



Linux  
Professional  
Institute

# LPIC-1

Version 5.0  
English

# 102

# Table of Contents

<b>TOPIC 105: SHELLS AND SHELL SCRIPTING</b>	<b>1</b>
<b>105.1 Customize and use the shell environment</b>	<b>2</b>
105.1 Lesson 1	4
Introduction	4
Shell Types: Interactive vs. Non-Interactive and Login vs. Non-Login	5
Guided Exercises	18
Explorational Exercises	20
Summary	22
Answers to Guided Exercises	24
Answers to Explorational Exercises	26
105.1 Lesson 2	28
Introduction	28
Variables: Assignment and Reference	28
Local or Shell Variables	32
Global or Environment Variables	34
Guided Exercises	45
Explorational Exercises	48
Summary	50
Answers to Guided Exercises	52
Answers to Explorational Exercises	56
105.1 Lesson 3	58
Introduction	58
Creating Aliases	58
Creating Functions	62
Guided Exercises	73
Explorational Exercises	76
Summary	77
Answers to Guided Exercises	79
Answers to Explorational Exercises	84
<b>105.2 Customize or write simple scripts</b>	<b>85</b>
105.2 Lesson 1	87
Introduction	87
Script Structure and Execution	88
Variables	90
Arithmetic Expressions	93
Conditional Execution	94
Script Output	95
Guided Exercises	98

Explorational Exercises	99
Summary	100
Answers to Guided Exercises	101
Answers to Explorational Exercises	102
105.2 Lesson 2	103
Introduction	103
Extended Tests	103
Loop Constructs	108
A More Elaborate Example	111
Guided Exercises	115
Explorational Exercises	117
Summary	118
Answers to Guided Exercises	119
Answers to Explorational Exercises	121
<b>TOPIC 106: USER INTERFACES AND DESKTOPS</b>	<b>122</b>
<b>106.1 Install and configure X11</b>	<b>123</b>
106.1 Lesson 1	124
Introduction	124
X Window System Architecture	125
X Server Configuration	128
Wayland	133
Guided Exercises	135
Explorational Exercises	136
Summary	137
Answers to Guided Exercises	138
Answers to Explorational Exercises	139
<b>106.2 Graphical Desktops</b>	<b>140</b>
106.2 Lesson 1	141
Introduction	141
X Window System	142
Desktop Environment	142
Popular Desktop Environments	144
Desktop Interoperability	146
Non-Local Access	147
Guided Exercises	149
Explorational Exercises	150
Summary	151
Answers to Guided Exercises	152
Answers to Explorational Exercises	153
<b>106.3 Accessibility</b>	<b>154</b>

106.3 Lesson 1	155
Introduction	155
Accessibility Settings	155
Keyboard and Mouse Assist	156
Visual Impairments	158
Guided Exercises	160
Explorational Exercises	161
Summary	162
Answers to Guided Exercises	163
Answers to Explorational Exercises	164
<b>TOPIC 107: ADMINISTRATIVE TASKS</b>	<b>165</b>
<b>107.1 Manage user and group accounts and related system files</b>	<b>166</b>
107.1 Lesson 1	168
Introduction	168
Adding User Accounts	168
Modifying User Accounts	170
Deleting User Accounts	172
Adding, Modifying and Deleting Groups	172
The Skeleton Directory	173
The <code>/etc/login.defs</code> File	173
The <code>passwd</code> Command	174
The <code>chage</code> Command	175
Guided Exercises	177
Explorational Exercises	178
Summary	179
Answers to Guided Exercises	181
Answers to Explorational Exercises	183
107.1 Lesson 2	185
Introduction	185
<code>/etc/passwd</code>	186
<code>/etc/group</code>	186
<code>/etc/shadow</code>	187
<code>/etc/gshadow</code>	188
Filter the Password and Group Databases	188
Guided Exercises	190
Explorational Exercises	192
Summary	193
Answers to Guided Exercises	194
Answers to Explorational Exercises	196
<b>107.2 Automate system administration tasks by scheduling jobs</b>	<b>198</b>

107.2 Lesson 1	200
Introduction	200
Schedule Jobs with Cron	200
User Crontabs	201
System Crontabs	202
Particular Time Specifications	203
Crontab Variables	203
Creating User Cron Jobs	204
Creating System Cron Jobs	205
Configure Access to Job Scheduling	206
An Alternative to Cron	206
Guided Exercises	209
Explorational Exercises	211
Summary	212
Answers to Guided Exercises	213
Answers to Explorational Exercises	215
107.2 Lesson 2	217
Introduction	217
Schedule Jobs with at	217
List Scheduled Jobs with atq	218
Delete Jobs with atrm	219
Configure Access to Job Scheduling	219
Time Specifications	220
An Alternative to at	220
Guided Exercises	221
Explorational Exercises	222
Summary	223
Answers to Guided Exercises	224
Answers to Explorational Exercises	225
<b>107.3 Localisation and internationalisation</b>	<b>227</b>
107.3 Lesson 1	229
Introduction	229
Time Zones	230
Daylight Savings Time	234
Language and Character Encoding	234
Encoding Conversion	238
Guided Exercises	239
Explorational Exercises	240
Summary	241
Answers to Guided Exercises	242

Answers to Explorational Exercises .....	243
<b>TOPIC 108: ESSENTIAL SYSTEM SERVICES</b> .....	<b>244</b>
<b>108.1 Maintain system time</b> .....	<b>245</b>
108.1 Lesson 1 .....	247
Introduction .....	247
Local Versus Universal Time .....	248
Date .....	248
Hardware Clock .....	250
timedatectl .....	250
Setting Time Zone Without timedatectl .....	252
Setting Date and Time Without timedatectl .....	253
Guided Exercise .....	255
Explorational Exercises .....	257
Summary .....	258
Answers to Guided Exercises .....	260
Answers to Explorational Exercises .....	262
108.1 Lesson 2 .....	263
Introduction .....	263
timedatectl .....	264
NTP Daemon .....	266
NTP Configuration .....	266
pool.ntp.org .....	267
ntpdate .....	268
ntpq .....	268
chrony .....	269
Guided Exercise .....	274
Explorational Exercise .....	276
Summary .....	277
Answers Guided Exercise .....	278
Answers to Explorational Exercises .....	280
<b>108.2 System logging</b> .....	<b>281</b>
108.2 Lesson 1 .....	283
Introduction .....	283
System Logging .....	283
Guided Exercises .....	304
Explorational Exercises .....	306
Summary .....	307
Answers to Guided Exercises .....	308
Answers to Explorational Exercises .....	311
108.2 Lesson 2 .....	312

Introduction	312
Basics of systemd	312
The System Journal: systemd-journal	313
Guided Exercises	331
Explorational Exercises	333
Summary	334
Answers to Guided Exercises	335
Answers to Explorational Exercises	337
<b>108.3 Mail Transfer Agent (MTA) basics</b>	<b>338</b>
108.3 Lesson 1	339
Introduction	339
Local and Remote MTA	339
Linux MTAs	341
The mail Command and Mail User Agents (MUA)	346
Delivery Customization	347
Guided Exercises	349
Explorational Exercises	350
Summary	351
Answers to Guided Exercises	352
Answers to Explorational Exercises	353
<b>108.4 Manage printers and printing</b>	<b>354</b>
108.4 Lesson 1	355
Introduction	355
The CUPS Service	356
Installing a Printer	360
Managing Printers	362
Submitting Print Jobs	363
Managing Print Jobs	366
Removing Printers	367
Guided Exercises	368
Explorational Exercises	369
Summary	370
Answers to Guided Exercises	372
Answers to Explorational Exercises	373
<b>TOPIC 109: NETWORKING FUNDAMENTALS</b>	<b>375</b>
<b>109.1 Fundamentals of internet protocols</b>	<b>376</b>
109.1 Lesson 1	377
Introduction	377
IP (Internet Protocol)	377
Guided Exercises	386

Explorational Exercises	387
Summary	388
Answers to Guided Exercises	389
Answers to Explorational Exercises	390
109.1 Lesson 2	391
Introduction	391
Transmission Control Protocol (TCP)	393
User Datagram Protocol (UDP)	393
Internet Control Message Protocol (ICMP)	393
IPv6	393
Guided Exercises	397
Explorational Exercises	398
Summary	399
Answers to Guided Exercises	400
Answers to Explorational Exercises	401
<b>109.2 Persistent network configuration</b>	<b>402</b>
109.2 Lesson 1	403
Introduction	403
The Network Interface	403
Interface Names	405
Interface Management	406
Local and Remote Names	408
Guided Exercises	412
Explorational Exercises	413
Summary	414
Answers to Guided Exercises	415
Answers to Explorational Exercises	416
109.2 Lesson 2	417
Introduction	417
NetworkManager	417
systemd-networkd	421
Guided Exercises	425
Explorational Exercises	426
Summary	427
Answers to Guided Exercises	428
Answers to Explorational Exercises	429
<b>109.3 Basic network troubleshooting</b>	<b>430</b>
109.3 Lesson 1	432
Introduction	432
About the ip Command	432



Netmask and Routing Review .....	433
Configuring an Interface .....	434
The Routing Table .....	437
Guided Exercises .....	440
Explorational Exercises .....	441
Summary .....	442
Answers to Guided Exercises .....	443
Answers to Explorational Exercises .....	445
109.3 Lesson 2 .....	447
Introduction .....	447
Testing Connections With ping .....	447
Tracing Routes .....	448
Finding MTUs With tracepath .....	450
Creating Arbitrary Connections .....	451
Viewing Current Connections and Listeners .....	453
Guided Exercises .....	455
Explorational Exercises .....	456
Summary .....	457
Answers to Guided Exercises .....	458
Answers to Explorational Exercises .....	460
<b>109.4 Configure client side DNS .....</b>	<b>462</b>
109.4 Lesson 1 .....	463
Introduction .....	463
Name Resolution Process .....	463
DNS Classes .....	464
Name Resolution Tools .....	467
Guided Exercises .....	473
Explorational Exercises .....	474
Summary .....	475
Answers to Guided Exercises .....	476
Answers to Explorational Exercises .....	477
<b>TOPIC 110: SECURITY .....</b>	<b>478</b>
<b>110.1 Perform security administration tasks .....</b>	<b>479</b>
110.1 Lesson 1 .....	481
Introduction .....	481
Checking for Files with the SUID and SGID Set .....	481
Password Management and Aging .....	484
Discovering Open Ports .....	487
Limits on Users Logins, Processes and Memory Usage .....	494
Dealing with Logged in Users .....	496

Basic sudo Configuration and Usage .....	499
Guided Exercises .....	504
Explorational Exercises .....	507
Summary .....	508
Answers to Guided Exercises .....	510
Answers to Explorational Exercises .....	514
<b>110.2 Setup host security .....</b>	<b>515</b>
110.2 Lesson 1 .....	516
Introduction .....	516
Improve Authentication Security with Shadow Passwords .....	516
How to Use a Superdaemon to Listen for Incoming Network Connections .....	518
Checking Services for Unnecessary Daemons .....	523
TCP Wrappers as Sort of a Simple Firewall .....	525
Guided Exercises .....	526
Explorational Exercises .....	527
Summary .....	528
Answers to Guided Exercises .....	530
Answers to Explorational Exercises .....	531
<b>110.3 Securing data with encryption .....</b>	<b>532</b>
110.3 Lesson 1 .....	534
Introduction .....	534
Basic OpenSSH Client Configuration and Usage .....	535
The Role of OpenSSH Server Host Keys .....	540
SSH Port Tunnels .....	542
Guided Exercises .....	546
Explorational Exercises .....	548
Summary .....	549
Answers to Guided Exercises .....	550
Answers to Explorational Exercises .....	552
110.3 Lesson 2 .....	553
Introduction .....	553
Perform Basic GnuPG Configuration, Usage and Revocation .....	553
Use GPG to Encrypt, Decrypt, Sign and Verify Files .....	559
Guided Exercises .....	564
Explorational Exercises .....	566
Summary .....	567
Answers to Guided Exercises .....	568
Answers to Explorational Exercises .....	570
<b>Imprint .....</b>	<b>571</b>



**Linux  
Professional  
Institute**

## **Topic 105: Shells and Shell Scripting**



## 105.1 Customize and use the shell environment

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 105.1](#)

### Weight

4

### Key knowledge areas

- Set environment variables (e.g. `PATH`) at login or when spawning a new shell.
- Write Bash functions for frequently used sequences of commands.
- Maintain skeleton directories for new user accounts.
- Set command search path with the proper directory.

### Partial list of the used files, terms and utilities

- `.`
- `source`
- `/etc/bash.bashrc`
- `/etc/profile`
- `env`
- `export`
- `set`
- `unset`
- `~/.bash_profile`
- `~/.bash_login`

- `~/.profile`
- `~/.bashrc`
- `~/.bash_logout`
- `function`
- `alias`



# 105.1 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	105 Shells and Shell Scripting
<b>Objective:</b>	105.1 Customize and use the shell environment
<b>Lesson:</b>	1 of 3

## Introduction

The shell is arguably the most powerful tool in a Linux system and can be defined as an interface between the user and the kernel of the operating system. It interprets commands entered by the user. Therefore, all system administrators must be skilled in using the shell. As we may certainly know by now, the Bourne Again Shell (*Bash*) is the *de facto* shell for the vast majority of Linux distributions.

Once started, the first thing *Bash* — or any other shell for that matter — does is executing a series of startup scripts. These scripts customize the session's environment. There are both system wide and user specific scripts. We can put our personal preferences or settings that best fit our users' needs in these scripts in the form of variables, aliases and functions.

The exact series of startup files depends on a very important parameter: the type of shell. Let us have a look at the variety of shells that exist.

# Shell Types: Interactive vs. Non-Interactive and Login vs. Non-Login

To start with, let us clarify the concepts of *interactive* and *login* in the context of shells:

## Interactive / Non-interactive Shells

This kind of shell refers to the interaction that takes place between the user and the shell: The user provides input by typing commands into the terminal using the keyboard; the shell provides output by printing messages on the screen.

## Login / Non-login Shells

This kind of shell refers to the event of a user accessing a computer system providing its credentials, such as username and password.

Both interactive and non-interactive shells can be either login or non-login and any possible combination of these types has its specific uses.

*Interactive login shells* are executed when users log into the system and are used to customize users' configurations according to their needs. A good example of this type of shell would be that of a group of users belonging to the same department who need a particular variable set in their sessions.

By *interactive non-login shells* we refer to any other shells opened by the user after logging into the system. Users use these shells during sessions to carry out maintenance and administrative tasks such as setting variables, the time, copying files, writing scripts, etc.

On the other hand, *non-interactive shells* do not require any kind of human interaction. Thus, these shells do not ask the user for input and their output — if any — is in most cases written to a log.

*Non-interactive login shells* are quite rare and impractical. Their uses are virtually non-existent and we will only comment on them for the sake of insight into shell behaviour. Some odd examples include forcing a script to be run from a login shell with `/bin/bash --login <some_script>` or piping the standard output (*stdout*) of a command into the standard input (*stdin*) of an ssh connection:

```
<some_command> | ssh <some_user>@<some_server>
```

As for *non-interactive non-login shell* there is neither interaction nor login on behalf of the user, so we are referring here to the use of automated scripts. These scripts are mostly used to carry out

repetitive administrative and maintenance tasks such as those included in cronjobs. In such cases, `bash` does not read any startup files.

## Opening a Terminal

When we are in a desktop environment, we can either open a terminal application or switch to one of the system consoles. Therefore, a new shell is either a `pts` shell when opened from a terminal emulator in the GUI or a `tty` shell when run from a system console. In the first case we are not dealing with a terminal but with a terminal emulator. As part of graphical sessions, terminal emulators like *gnome-terminal* or *konsole* are very feature-rich and user-friendly as compared to text-based user interface terminals. Less feature-rich terminal emulators include — amongst others — *XTerm* and *sakura*.

Using the `Ctrl + Alt + F1-F6` combos we can go to the console logins which open an interactive text-based login shell. `Ctrl + Alt + F7` will take the session back into the Desktop.

**NOTE** `tty` stands for teletypewriter; `pts` stands for pseudo terminal slave. For more information: `man tty` and `man pts`.

## Launching Shells with `bash`

After logging in, type `bash` into a terminal to open a new shell. Technically, this shell is a child process of the current shell.

While starting the `bash` child process, we can specify various switches to define which kind of shell we want to start. Here some important `bash` invocation options:

### **`bash -l` or `bash --login`**

will invoke a login shell.

### **`bash -i`**

will invoke an interactive shell.

### **`bash --noprofile`**

with login shells will ignore both the system-wide startup file `/etc/profile` and the user-level startup files `~/.bash_profile`, `~/.bash_login` and `~/.profile`.

### **`bash --norc`**

with interactive shells will ignore both the system-wide startup file `/etc/bash.bashrc` and the user-level startup file `~/.bashrc`.



## **bash --rcfile <file>**

with interactive shells will take `<file>` as the startup file ignoring system wide `/etc/bash.bashrc` and user-level `~/ .bashrc`.

We will discuss the various startup files below.

## **Launching Shells with su and sudo**

Through the use of these two similar programs we can obtain specific types of shells:

### **su**

Change user ID or become superuser (`root`). With this command we can invoke both login and non-login shells:

- `su - user2`, `su -l user2` or `su --login user2` will start an interactive login shell as `user2`.
- `su user2` will start an interactive non-login shell as `user2`.
- `su - root` or `su -` will start an interactive login shell as `root`.
- `su root` or `su` will start an interactive non-login shell as `root`.

### **sudo**

Execute commands as another user (including the supersuser). Because this command is mainly used to gain root privileges temporarily, the user using it must be in the `sudoers` file. To add users to `sudoers` we need to become `root` and then run:

```
root@debian:~# usermod -aG sudo user2
```

Just as `su`, `sudo` allows us to invoke both login and non-login shells:

- `sudo su - user2`, `sudo su -l user2` or `sudo su --login user2` will start an interactive login shell as `user2`.
- `sudo su user2` will start an interactive non-login shell as `user2`.
- `sudo -u user2 -s` will start an interactive non-login shell as `user2`.
- `sudo su - root` or `sudo su -` will start an interactive login shell as `root`.
- `sudo -i` will start an interactive login shell as `root`.
- `sudo -i <some_command>` will start an interactive login shell as `root`, run the command and return to the original user.

- `sudo su root` or `sudo su` will start an interactive non-login shell as `root`.
- `sudo -s` or `sudo -u root -s` will start a non-login shell as `root`.

When using either `su` or `sudo`, it is important to consider our particular case scenario for starting a new shell: Do we need the target user's environment or not? If so, we would use the options which invoke login shells; if not, those which invoke non-login shells.

## What Shell Type Do We Have?

In order to find out what type of shell we are working at, we can type `echo $0` into the terminal and get the following output:

### Interactive login

`-bash` or `-su`

### Interactive non-login

`bash` or `/bin/bash`

### Non-interactive non-login (scripts)

`<name_of_script>`

## How Many Shells Do We Have?

To see how many `bash` shells we have up and running in the system, we can use the command `ps aux | grep bash`:

```
user2@debian:~$ ps aux | grep bash
user2      5270  0.1  0.1 25532  5664 pts/0    Ss   23:03   0:00 bash
user2      5411  0.3  0.1 25608  5268 tty1      S+   23:03   0:00 -bash
user2      5452  0.0  0.0 16760   940 pts/0    S+   23:04   0:00 grep --color=auto bash
```

`user2` at `debian` has logged into a GUI (or X Window System) session and opened *gnome-terminal*, then she has pressed `Ctrl + Alt + F1` to go into a `tty` terminal session. Finally, she has gone back to the GUI session by pressing `Ctrl + Alt + F7` and typed in the command `ps aux | grep bash`. Thus, the output shows an interactive non-login shell via the terminal emulator (`pts/0`) and an interactive login shell via the proper text-based terminal (`tty1`). Note also how the last field of each line (the command) is `bash` for the former and `-bash` for the latter.

## Where Shells Get their Configuration From: Startup Files

Well, now that we know the shell types that we can find in a Linux system, it is high time that we saw what startup files get executed by what shell. Note that system wide or global scripts are placed in the `/etc/` directory, whereas local or user-level ones are found in the user's home (`~`). Also, when there is more than one file to be searched, once one is found and run the others are ignored. Explore and study these files yourself with your favorite text editor or by typing `less <startup_file>`.

### NOTE

Startup files can be divided into Bash specific (those limited only to `bash` configurations and commands) and general ones (relating to most shells).

## Interactive Login Shell

### Global Level

#### `/etc/profile`

This is the system-wide `.profile` file for the Bourne shell and Bourne compatible shells (`bash` included). Through a series of `if` statements this file sets a number of variables such as `PATH` and `PS1` accordingly as well as sourcing — if they exist — both the file `/etc/bash.bashrc` and those in the directory `/etc/profile.d`.

#### `/etc/profile.d/*`

This directory may contain scripts that get executed by `/etc/profile`.

### Local Level

#### `~/.bash_profile`

This Bash specific file is used for configuring the user environment. It can also be used to source both `~/.bash_login` and `~/.profile`.

#### `~/.bash_login`

Also Bash specific, this file will only be executed if there is no `~/.bash_profile` file. Its name suggests that it should be used to run commands needed on login.

#### `~/.profile`

This file is not Bash specific and gets sourced only if neither `~/.bash_profile` nor `~/.bash_login` exist — which is normally the case. Thus, the main purpose of `~/.profile` is that of checking if a Bash shell is being run and — if so — sourcing `~/.bashrc` if it exists. It usually sets the variable `PATH` so that it includes the user's private `~/bin` directory if it exists.

## ~/ .bash\_logout

If it exists, this Bash specific file does some clean-up operations when exiting the shell. This can be convenient in such cases as those in remote sessions.

## Exploring Interactive Login Shell Configuration Files

Let us show some of these files in action by modifying `/etc/profile` and `/home/user2/.profile`. We will append to each a line reminding us the file being executed:

```
root@debian:~# echo 'echo Hello from /etc/profile' >> /etc/profile
root@debian:~# echo 'echo Hello from ~/.profile' >> ~/.profile
```

**NOTE** Two redirection operators `>>` append the output of a command into an existing file — not overwriting it. If the file does not exist, though, it will be created.

Thus, through the output of their respective `echo` commands we will know when each of these files is read and executed. To prove it, let us see what happens when `user2` logs in via `ssh` from another machine:

```
user2@debian:~$ ssh user2@192.168.1.6
user2@192.168.1.6's password:
Linux debian 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov 27 19:57:19 2018 from 192.168.1.10
Hello from /etc/profile
Hello from /home/user2/.profile
```

As the last two lines show, it worked. Also, note three things:

- The global file was run first.
- There were no `.bash_profile` or `.bash_login` files in the home directory of `user2`.
- The tilde (`~`) expanded to the absolute path of the file (`/home/user2/.profile`).

## Interactive Non-Login Shell

### Global Level

#### `/etc/bash.bashrc`

This is the system-wide `.bashrc` file for interactive `bash` shells. Through its execution `bash` makes sure it is being run interactively, checks the window size after each command (updating the values of `LINES` and `COLUMNS` if necessary) and sets some variables.

### Local Level

#### `~/.bashrc`

In addition to carrying out similar tasks to those described for `/etc/bash.bashrc` at a user level (such as checking the window size or if being run interactively), this Bash specific file usually sets some history variables and sources `~/.bash_aliases` if it exists. Apart from that, this file is normally used to store users' specific aliases and functions.

Likewise, it is also worthwhile noting that `~/.bashrc` is read if `bash` detects its `<stdin>` is a network connection (as it was the case with the *Secure Shell* (SSH) connection in the example above).

## Exploring Interactive Non-Login Shell Configuration Files

Let us now modify `/etc/bash.bashrc` and `/home/user2/.bashrc`:

```
root@debian:~# echo 'echo Hello from /etc/bash.bashrc' >> /etc/bash.bashrc
root@debian:~# echo 'echo Hello from ~/.bashrc' >> ~/.bashrc
```

And this is what happens when `user2` starts a new shell:

```
user2@debian:~$ bash
Hello from /etc/bash.bashrc
Hello from /home/user2/.bashrc
```

Again, the two files were read and executed.

### WARNING

Remember, because of the order in which files are run, local files take precedence over global ones.

## Non-Interactive Login Shell

A non-interactive shell with the `-l` or `--login` options is forced to behave like a login shell and so the startup files to be run will be the same as those for interactive login shells.

To prove it, let us write a simple script and make it executable. We will not include any *shebangs* because we will be invoking the *bash executable* (`/bin/bash` with the login option) from the command line.

1. We create the script `test.sh` containing the line `echo 'Hello from a script'` so that we can prove the script runs successfully:

```
user2@debian:~$ echo "echo 'Hello from a script'" > test.sh
```

2. We make our script executable:

```
user2@debian:~$ chmod +x ./test.sh
```

3. Finally, we invoke `bash` with the `-l` option to run the script:

```
user2@debian:~$ bash -l ./test.sh
Hello from /etc/profile
Hello from /home/user2/.profile
Hello from a script
```

It works! Before running the script, the login took place and both `/etc/profile` and `~/.profile` were executed.

**NOTE** We will learn about *shebangs* and all other aspects of shell scripting in future lessons.

Let us now have the standard output (*stdout*) of the `echo` command into the standard input (*stdin*) of an ssh connection by means of a pipe (`|`):

```
user2@debian:~$ echo "Hello-from-a-noninteractive-login-shell" | ssh user2@192.168.1.6
Pseudo-terminal will not be allocated because stdin is not a terminal.
user2@192.168.1.6's password:
Linux debian 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27) x86_64

The programs included with the Debian GNU/Linux system are free software;
```

```
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Hello from /etc/profile
Hello from /home/user2/.profile
-bash: line 1: Hello-from-a-noninteractive-login-shell: command not found
```

Again, `/etc/profile` and `~/.profile` are executed. Other than that, the first and last lines of the output are quite telling as far as the behaviour of the shell is concerned.

### Non-Interactive Non-Login Shell

Scripts do not read any of the files listed above but look for the environment variable `BASH_ENV`, expand its value if needed and use it as the name of a startup file to read and execute commands. We will learn more about *environment variables* in the next lesson.

As mentioned above, typically `/etc/profile` and `~/.profile` make sure that both `/etc/bash.bashrc` and `~/.bashrc` get executed after a successful login. The output of the following command shows this phenomenon:

```
root@debian:~# su - user2
Hello from /etc/bash.bashrc
Hello from /etc/profile
Hello from /home/user2/.bashrc
Hello from /home/user2/.profile
```

Bearing in mind the lines we previously appended to the startup scripts—and invoking an interactive-login shell at user-level with `su - user2`—the four lines of output can be explained as follows:

1. `Hello from /etc/bash.bashrc` means `/etc/profile` has sourced `/etc/bash.bashrc`.
2. `Hello from /etc/profile` means `/etc/profile` has been fully read and executed.
3. `Hello from /home/user2/.bashrc` means `~/.profile` has sourced `~/.bashrc`.
4. `Hello from /home/user2/.profile` means `~/.profile` has been fully read and executed.

Note how with `su - <username>` (also `su -l <username>` and `su --login <username>`) we guarantee the invocation of a login shell, whereas `su <username>` would have only invoked `/etc/bash.bashrc` and `~/.bashrc`.

## Sourcing Files

In the previous sections we have discussed that some startup scripts include or execute other scripts. This mechanism is called sourcing and is explained in this section.

### Sourcing Files with `.`

The dot (`.`) is normally found in startup files.

In the `.profile` file of our Debian server we can find — for instance — the following block:

```
# include .bashrc if it exists
if [ -f "$HOME/.bashrc" ]; then
. "$HOME/.bashrc"
fi
```

We already saw how the execution of a script may lead to that of another. Thus, the `if` statement guarantees that the file `$HOME/.bashrc` — if it exists (`-f`) — will be sourced (i.e. read and executed) at login:

```
. "$HOME/.bashrc"
```

#### NOTE

As we will learn in the next lesson, `$HOME` is an environment variable that expands to the absolute path of the user's home directory.

Furthermore, we can use the `.` whenever we have modified a startup file and want to make the changes effective without a reboot. For example we can:

- add an alias to `~/.bashrc`:

```
user2@debian:~$ echo "alias hi='echo We salute you.'" >> ~/.bashrc
```

#### WARNING

When sending the output of one command into a file, remember not to mistake append (`>>`) with overwrite (`>`).

- output the last line of `~/.bashrc` to check everything went fine:

```
user2@debian:~$ tail -n 1 !$
tail -n 1 ~/.bashrc
alias hi='echo We salute you.'
```



**NOTE**

!`$` expands to the last argument from the previous command, in our case: `~/ .bashrc`.

- source the file by hand:

```
user2@debian:~$ . ~/ .bashrc
```

- and invoke the alias to prove it works:

```
user2@debian:~$ hi
We salute you.
```

**NOTE**

Refer to the next lesson to learn about *aliases* and *variables*.

## Sourcing Files with `source`

The `source` command is a synonym for `..`. So to source `~/ .bashrc` we can also do it like this:

```
user2@debian:~$ source ~/ .bashrc
```

## The Origin of Shell Startup Files: `SKEL`

`SKEL` is a variable whose value is the absolute path to the `skel` directory. This directory serves as a template for the file system structure of users' home directories. It includes the files that will be inherited by any new user accounts created (including, of course, the configuration files for shells). `SKEL` and other related variables are stored in `/etc/adduser.conf`, which is the configuration file for `adduser`:

```
user2@debian:~$ grep SKEL /etc/adduser.conf
# The SKEL variable specifies the directory containing "skeletal" user
SKEL=/etc/skel
# If SKEL_IGNORE_REGEX is set, adduser will ignore files matching this
SKEL_IGNORE_REGEX="dpkg-(old|new|dist|save)"
```

`SKEL` is set to `/etc/skel`; thus, the startup scripts that configure our shells lie therein:

```
user2@debian:~$ ls -a /etc/skel/
.  ..  .bash_logout  .bashrc  .profile
```

**WARNING**

Remember, files starting with a `.` are hidden, so we must use `ls -a` to see them when listing directory contents.

Let us now create a directory in `/etc/skel` for all new users to store their personal scripts in:

1. As `root` we move into `/etc/skel`:

```
root@debian:~# cd /etc/skel/
root@debian:/etc/skel#
```

2. We list its contents:

```
root@debian:/etc/skel# ls -a
.  ..  .bash_logout  .bashrc  .profile
```

3. We create our directory and check all went as expected:

```
root@debian:/etc/skel# mkdir my_personal_scripts
root@debian:/etc/skel# ls -a
.  ..  .bash_logout  .bashrc  my_personal_scripts  .profile
```

4. Now we delete `user2` together with its home directory:

```
root@debian:~# deluser --remove-home user2
Looking for files to backup/remove ...
Removing files ...
Removing user `user2' ...
Warning: group `user2' has no more members.
Done.
```

5. We add `user2` again so that it gets a new home directory:

```
root@debian:~# adduser user2
Adding user `user2' ...
Adding new group `user2' (1001) ...
Adding new user `user2' (1001) with group `user2' ...
Creating home directory `/home/user2' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
```

```
passwd: password updated successfully
Changing the user information for user2
Enter the new value, or press ENTER for the default
  Full Name []:
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n] y
```

6. Finally, we login as `user2` and list all the files in `/home/user2` to see if everything went as expected:

```
root@debian:~# su - user2
user2@debian:~$ pwd
/home/user2
user2@debian:~$ ls -a
.  ..  .bash_history  .bash_logout  .bashrc  my_personal_scripts  .profile
```

It did.

## Guided Exercises

- Study how the shells have been started under the column “Shell Started with...” and complete with the required information:

Shell Started with...	Interactive?	Login?	Result of echo \$0
sudo ssh user2@machine2			
Ctrl + Alt + F2			
su - user2			
gnome-terminal			
A regular user uses <i>konsole</i> to start an instance of <i>sakura</i>			
A script named <code>test.sh</code> containing the command <code>echo \$0</code>			

- Write the `su` and `sudo` commands to launch the specified shell:

### Interactive-login shell as user2

su:

sudo:

### Interactive login shell as root

su:

sudo:

### Interactive non-login shell as root

su:

sudo:

### Interactive non-login shell as user2

su:

sudo:

3. What startup file gets read when the shell under “Shell Type” is started?

Shell Type	/etc/profile	/etc/bash.bashrc	~/.profile	~/.bashrc
Interactive-login shell as user2				
Interactive login shell as root				
Interactive non-login shell as root				
Interactive non-login shell as user2				

## Explorational Exercises

1. In Bash we can write a simple `Hello world!` function by including the following code in an empty file:

```
function hello() {  
    echo "Hello world!"  
}
```

- What should we do next to make the function available to the shell?

- Once it is available to the current shell, how would you invoke it?

- To automate things, in what file would you put the function and its invocation so that it gets executed when `user2` opens a terminal from an X Window session? What type of shell is it?

- In what file would you put the function and its invocation so that it is run when `root` launches a new interactive shell irrespective of whether it is login or not?

2. Have a look at the following basic, `Hello world!` bash script:

```
#!/bin/bash  
  
#hello_world: a simple bash script to discuss interaction in scripts.  
  
echo "Hello world!"
```

- Suppose we make the script executable and run it. Would that be an interactive script? Why?

- What makes a script interactive?

3. Imagine you have changed the values of some variables in `~/ .bashrc` and want those changes to take effect without a reboot. From your home directory, how could you achieve that in two

different ways?

4. John has just started an X Window session on a Linux server. He opens a terminal emulator to carry out some administrative tasks but, surprisingly, the session freezes and he needs to open a text shell.

- How can he open that `tty` shell?

- What startup files will get sourced?

5. Linda is a user of a Linux server. She kindly asks the administrator to have a `~/.bash_login` file so she can have the time and date printed on the screen when she logs in. Other users like the idea and follow suit. The administrator has a hard time creating the file for all other users on the server so he decides to add a new policy and have `~/.bash_login` created for all potential new users. How can the administrator accomplish that task?

# Summary

In this lesson we learned:

- Shells set users' environment in a Linux system.
- *Bash* is the number one shell across GNU/Linux distributions.
- The first job a shell carries out is that of reading and executing one or various startup files.
- The concepts of *interaction* and *login* as related to shells.
- How to launch different types of shells with `bash`, `su`, `sudo` and `Ctrl + Alt + F1-F6`.
- How to check the type of shell with `echo $0`.
- The local startup files `~/.bash_profile`, `~/.profile`, `~/.bash_login`, `~/.bash_logout` and `~/.bashrc`.
- The global startup files `/etc/profile`, `/etc/profile.d/*`, `/etc/bash.bashrc`.
- Local files take precedence over global ones.
- How to redirect the output of a command with `>` (overwrite) and `>>` (append).
- The meaning of the `skel` directory.
- How to source files.

Commands used in this lesson:

## **bash**

Create a new shell.

## **su**

Create a new shell.

## **sudo**

Create a new shell.

## **usermod**

Modify a user account.

## **echo**

Display a line of text.



**ps**

Report a snapshot of the current processes.

**less**

A pager for long files.

**ssh**

Start an Open SSH connection (remotely).

**chmod**

Change mode bits of a file, for example make it executable.

**grep**

Print lines matching a pattern.

**ls**

List directory contents.

**cd**

Change directory.

**mkdir**

Make directory.

**deluser**

Delete user.

**adduser**

Add a new user.

**.**

Source a file.

**source**

Source a file.

**tail**

Output the last part of files.

## Answers to Guided Exercises

1. Study how the shells have been started under the column “Shell Started with...” and complete with the required information:

Shell Started with...	Interactive?	Login?	Result of echo \$0
sudo ssh user2@machine2	Yes	Yes	-bash
Ctrl + Alt + F2	Yes	Yes	-bash
su - user2	Yes	Yes	-bash
gnome-terminal	Yes	No	bash
A regular user uses <i>konsole</i> to start an instance of <i>sakura</i>	Yes	No	/bin/bash
A script named test.sh containing the command echo \$0	No	No	./test.sh

2. Write the su and sudo commands to launch the specified shell:

### Interactive-login shell as user2

#### su

su - user2, su -l user2 or su --login user2

#### sudo

sudo su - user2, sudo su -l user2 or sudo su --login user2

### Interactive login shell as root

#### su

su - root or su -

#### sudo

sudo su - root, sudo su - or sudo -i

**Interactive non-login shell as root****su**`su root` or `su`**sudo**`sudo su root`, `sudo su`, `sudo -s` or `sudo -u root -s`**Interactive non-login shell as user2****su**`su user2`**sudo**`sudo su user2` or `sudo -u user2 -s`

## 3. What startup file gets read when the shell under “Shell Type” is started?

Shell Type	/etc/profile	/etc/bash.bashrc	~/.profile	~/.bashrc
Interactive-login shell as user2	Yes	Yes	Yes	Yes
Interactive login shell as root	Yes	Yes	No	No
Interactive non-login shell as root	No	Yes	No	No
Interactive non-login shell as user2	No	Yes	No	Yes

# Answers to Explorational Exercises

1. In Bash we can write a simple `Hello world!` function by including the following code in an empty file:

```
function hello() {  
    echo "Hello world!"  
}
```

- What should we do next to make the function available to the shell?

To make the function available to the current shell, we must source the file which includes it.

- Once it is available to the current shell, how would you invoke it?

We will invoke it by typing its name into the terminal.

- To automate things, in what file would you put the function and its invocation so that it gets executed when `user2` opens a terminal from an X Window session? What type of shell is it?

The best file to put it is `/home/user2/.bashrc`. The invoked shell would be an interactive non-login one.

- In what file would you put the function and its invocation so that it is run when `root` launches a new interactive shell irrespective of whether it is login or not?

In `/etc/bash.bashrc` since this file gets executed for all interactive shells — whether login or not.

2. Have a look at the following basic, `Hello world!` bash script:

```
#!/bin/bash  
  
#hello_world: a simple bash script to discuss interaction in scripts.  
  
echo "Hello world!"
```

- Suppose we make the script executable and run it. Would that be an interactive script? Why?

No, as there is no human interaction and no commands being typed in by the user.

- What makes a script interactive?

The fact that it requires user input.

3. Imagine you have changed the values of some variables in `~/ .bashrc` and want those changes to take effect without a reboot. From your home directory, how could you achieve that in two different ways?

```
$ source .bashrc
```

OR

```
$ . .bashrc
```

4. John has just started an X Window session on a Linux server. He opens a terminal emulator to carry out some administrative tasks but, surprisingly, the session freezes and he needs to open a text shell.

- How can he open that `tty` shell?

He could do that by pressing `Ctrl + Alt + F1-F6` to enter one of the six `tty` shells.

- What startup files will get sourced?

```
/etc/profile
```

```
/home/john/.profile
```

5. Linda is a user of a Linux server. She kindly asks the administrator to have a `~/ .bash_login` file so she can have the time and date printed on the screen when she logs in. Other users like the idea and follow suit. The administrator has a hard time creating the file for all other users on the server so he decides to add a new policy and have `~/ .bash_login` created for all potential new users. How can the administrator accomplish that task?

He could achieve that by putting `.bash_login` into the `/etc/skel` directory.



## 105.1 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	105 Shells and Shell Scripting
<b>Objective:</b>	105.1 Customize and use the shell environment
<b>Lesson:</b>	2 of 3

### Introduction

Think of a variable as an imaginary box in which to temporarily place a piece of information. The same as with its initialization scripts, Bash classifies variables as either *shell/local* (those which live only within the limits of the shell in which they were created) or *environment/global* (those that are inherited by children shells and/or processes). In fact, in the previous lesson we had a look at shells and their configuration or initialization scripts. It is now convenient to point out that the power of these startup files lies in the fact that they allow us to use variables — as well as aliases and functions — which help us create and customize the shell environment of our choice.

### Variables: Assignment and Reference

A variable can be defined as a name containing a value.

In Bash, giving a value to a name is called *variable assignment* and it is the way in which we create or set variables. On the other hand, the process of accessing the value contained in the name is called *variable referencing*.

The syntax for assigning variables is:

```
<variable_name>=<variable_value>
```

For example:

```
$ distro=zorinos
```

The variable `distro` equals `zorinos`, that is to say, there is a portion of memory holding the value `zorinos` — with `distro` being the pointer to it.

Note, however, that there can not be any space on either side of the equal sign when assigning a variable:

```
$ distro =zorinos
-bash: distro: command not found
$ distro= zorinos
-bash: zorinos: command not found
```

Because of our mistake, Bash read `distro` and `zorinos` as commands.

To reference a variable (that is, to check its value) we use the `echo` command preceding the variable name with a `$` sign:

```
$ echo $distro
zorinos
```

## Variable Names

When choosing the name of variables, there are certain rules that we must take into consideration.

The name of a variable may contain letters (`a-z,A-Z`), numbers (`0-9`) and underscores (`_`):

```
$ distro=zorinos
$ echo $distro
zorinos
$ DISTRO=zorinos
$ echo $DISTRO
zorinos
$ distro_1=zorinos
```

```
$ echo $distro_1
zorinos
$ _distro=zorinos
$ echo $_distro
zorinos
```

It may not start with a number or Bash will get confused:

```
$ 1distro=zorinos
-bash: 1distro=zorinos: command not found
```

It may not contain spaces (not even using quotes); by convention, underscores are used instead:

```
$ "my distro"=zorinos
-bash: my: command not found
$ my_distro=zorinos
$ echo $my_distro
zorinos
```

## Variable Values

Concerning the reference or value of variables it is also important to consider a number of rules.

Variables may contain any alphanumerical characters (a-z,A-Z,0-9) as well as most other characters (?,!,\*,.,/, etc.):

```
$ distro=zorin12.4?
$ echo $distro
zorin12.4?
```

Variable values must be enclosed in quotes if they contain single spaces:

```
$ distro=zorin 12.4
-bash: 12.4: command not found
$ distro="zorin 12.4"
$ echo $distro
zorin 12.4
$ distro='zorin 12.4'
$ echo $distro
```



```
zorin 12.4
```

Variable values must also be enclosed in quotes if they contain such characters as those used for redirection (<,>) or the pipe symbol (|). The only thing the following command does is create an empty file named `zorin`:

```
$ distro=>zorin
$ echo $distro

$ ls zorin
zorin
```

This works, though, when we use the quotes:

```
$ distro=">zorin"
$ echo $distro
>zorin
```

However, single and double quotes are not always interchangeable. Depending on what we are doing with a variable (assigning or referencing), the use of one or the other has implications and will yield different results. In the context of variable assignment single quotes take all the characters of the variable value *literally*, whereas double quotes allow for variable substitution:

```
$ lizard=uromastyx
$ animal='My $lizard'
$ echo $animal
My $lizard
$ animal="My $lizard"
$ echo $animal
My uromastyx
```

On the other hand, when referencing a variable whose value includes some initial (or extra) spaces—sometimes combined with asterisks—it is mandatory that we use double quotes after the `echo` command to avoid *field splitting* and *pathname expansion*:

```
$ lizard="  genus  |  uromastyx"
$ echo $lizard
genus | uromastyx
$ echo "$lizard"
```

```
genus | uromastyx
```

If the reference of the variable contains a closing exclamation mark, this must be the last character in the string (as otherwise Bash will think we are referring to a `history` event):

```
$ distro=zorin.?!os
-bash: !os: event not found
$ distro=zorin.?!
$ echo $distro
zorin.?!
```

Any backslashes must be escaped with another backslash. Also, if a backslash is the last character in the string and we do not escape it Bash will interpret we want a line break and give us a new line:

```
$ distro=zorinos\
>
$ distro=zorinos\\
$ echo $distro
zorinos\
```

In the next two sections we will sum up the main differences between *local* and *environment* variables.

## Local or Shell Variables

Local or shell variables exist only in the shell in which they are created. By convention, local variables are written in lower-case letters.

For the sake of carrying out a few tests, let us create a local variable. As explained above, we choose an appropriate variable name and equate it to an appropriate value. For instance:

```
$ reptile=tortoise
```

Let us now use the `echo` command to reference our variable and check that everything went as expected:

```
$ echo $reptile
```

```
tortoise
```

In certain scenarios — such as when writing scripts — immutability can be an interesting feature of variables. If we want our variables to be immutable, we can either create them `readonly`:

```
$ readonly reptile=tortoise
```

Or turn them so after they have been created:

```
$ reptile=tortoise
$ readonly reptile
```

Now, if we try to change the value of `reptile`, Bash will refuse:

```
$ reptile=lizard
-bash: distro: readonly variable
```

#### NOTE

To list all readonly variables in our current session, type `readonly` or `readonly -p` into the terminal.

A useful command when dealing with local variables is `set`.

`set` outputs all of the currently assigned shell variables and functions. Since that can be a lot of lines (try it yourself!), it is recommended to use it in combination with a pager such as `less`:

```
$ set | less
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:force_fignore:histappend:interactive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_COMPAT_DIR=/etc/bash_completion.d
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=( [0]="4" [1]="4" [2]="12" [3]="1" [4]="release" [5]="x86_64-pc-linux-gnu" )
BASH_VERSION='4.4.12(1)-release'
(...)
```

Is our `reptile` variable there?

```
$ set | grep reptile
reptile=tortoise
```

Yes, there it is!

However, `reptile`—being a local variable—will not be inherited by any child processes spawned from the current shell:

```
$ bash
$ set | grep reptile
$
```

And, of course, we cannot echo out its value either:

```
$ echo $reptile
$
```

**NOTE** By typing the `bash` command into the terminal we open a new (child) shell.

To unset any variables (either local or global), we use the `unset` command:

```
$ echo $reptile
tortoise
$ unset reptile
$ echo $reptile
$
```

**NOTE** `unset` must be followed by the name of the variable alone (not preceded by the `$` symbol).

## Global or Environment Variables

Global or environment variables exist for the current shell as well as for all subsequent processes spawned from it. By convention, environment variables are written in uppercase letters:

```
$ echo $SHELL
```

```
/bin/bash
```

We can recursively pass the value of these variables to other variables and the value of the latter will ultimately expand to that of the former:

```
$ my_shell=$SHELL
$ echo $my_shell
/bin/bash
$ your_shell=$my_shell
$ echo $your_shell
/bin/bash
$ our_shell=$your_shell
$ echo $our_shell
/bin/bash
```

In order for a local shell variable to become an environment variable, the `export` command must be used:

```
$ export reptile
```

With `export reptile` we have turned our local variable into an environment variable so that child shells can recognize it and use it:

```
$ bash
$ echo $reptile
tortoise
```

Likewise, `export` can be used to set and export a variable, all at once:

```
$ export amphibian=frog
```

Now we can open a new instance of Bash and successfully reference the new variable:

```
$ bash
$ echo $amphibian
frog
```

#### NOTE

With `export -n <VARIABLE-NAME>` the variable will be turned back into a local

shell variable.

The `export` command will also give us a list of all existing environment variables when typed on its own (or with the `-p` option):

```
$ export
declare -x HOME="/home/user2"
declare -x LANG="en_GB.UTF-8"
declare -x LOGNAME="user2"
(...)
declare -x PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
declare -x PWD="/home/user2"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x SSH_CLIENT="192.168.1.10 49330 22"
declare -x SSH_CONNECTION="192.168.1.10 49330 192.168.1.7 22"
declare -x SSH_TTY="/dev/pts/0"
declare -x TERM="xterm-256color"
declare -x USER="user2"
declare -x XDG_RUNTIME_DIR="/run/user/1001"
declare -x XDG_SESSION_ID="8"
declare -x reptile="tortoise"
```

**NOTE** The command `declare -x` is equivalent to `export`.

Two more commands that can be used to print a list of all environment variables are `env` and `printenv`:

```
$ env
SSH_CONNECTION=192.168.1.10 48678 192.168.1.7 22
LANG=en_GB.UTF-8
XDG_SESSION_ID=3
USER=user2
PWD=/home/user2
HOME=/home/user2
SSH_CLIENT=192.168.1.10 48678 22
SSH_TTY=/dev/pts/0
MAIL=/var/mail/user2
TERM=xterm-256color
SHELL=/bin/bash
SHLVL=1
LOGNAME=user2
XDG_RUNTIME_DIR=/run/user/1001
```

```
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
_=/usr/bin/env
```

On top of being a synonym for `env`, we can sometimes use `printenv` in a similar way as we use the `echo` command to check for the value of a variable:

```
$ echo $PWD
/home/user2
$ printenv PWD
/home/user2
```

Note, however, that with `printenv` the variable name is not preceded by `$`.

#### NOTE

`PWD` stores the path of the present working directory. We will learn about this and other common environment variables later.

## Running a Program in a Modified Environment

`env` can be used to modify the shell environment at a program's time of execution.

To start a new Bash session with as empty an environment as possible — clearing most variables (as well as functions and aliases) — we will use `env` with the `-i` option:

```
$ env -i bash
```

Now most of our environment variables are gone:

```
$ echo $USER
$
```

And only a few remain:

```
$ env
LS_COLORS=
PWD=/home/user2
SHLVL=1
_=/usr/bin/printenv
```

We can also use `env` to set a particular variable for a particular program.

In our previous lesson, when discussing *non-interactive non-login shells*, we saw how scripts do not read any standard startup files but instead they look for the value of the `BASH_ENV` variable and use it as their startup file if it exists.

Let us demonstrate this process:

1. We create our own startup file called `.startup_script` with the following content:

```
CROCODILIAN=caiman
```

2. We write a Bash script named `test_env.sh` with the following content:

```
#!/bin/bash  
  
echo $CROCODILIAN
```

3. We set the executable bit for our `test_env.sh` script:

```
$ chmod +x test_env.sh
```

4. Finally, we use `env` to set `BASH_ENV` to `.startup_script` for `test_env.sh`:

```
$ env BASH_ENV=/home/user2/.startup_script ./test_env.sh  
caiman
```

The `env` command is implicit even if we get rid of it:

```
$ BASH_ENV=/home/user2/.startup_script ./test_env.sh  
caiman
```

## NOTE

If you do not understand the line `#!/bin/bash` or the `chmod +x` command, do not panic! We will learn everything necessary about shell scripting in future lessons. For now, just remember that to execute a script from within its own directory we use `./some_script`.

## Common Environment Variables

Time now for reviewing some of the most relevant environment variables that are set in Bash



configuration files.

## DISPLAY

Related to the X server, this variable's value is normally made up of three elements:

- The hostname (the absence of it means `localhost`) where the X server is running.
- A colon as a delimiter.
- A number (it is normally `0` and refers to the computer's display).

```
$ printenv DISPLAY
:0
```

An empty value for this variable means a server without an X Window System. An extra number—as in `my.xserver:0:1`—would refer to the screen number if more than one exists.

## HISTCONTROL

This variable controls what commands get saved into the `HISTFILE` (see below). There are three possible values:

### `ignorespace`

Commands starting with a space will not be saved.

### `ignoredups`

A command which is the same as the previous one will not be saved.

### `ignoreboth`

Commands which fall into any of the two previous categories will not be saved.

```
$ echo $HISTCONTROL
ignoreboth
```

## HISTSIZE

This sets the number of commands to be stored in memory while the shell session lasts.

```
$ echo $HISTSIZE
1000
```

## HISTFILESIZE

This sets the number of commands to be saved in HISTFILE both at the start and at the end of the session:

```
$ echo $HISTFILESIZE
2000
```

## HISTFILE

The name of the file which stores all commands as they are typed. By default this file is located at `~/.bash_history`:

```
$ echo $HISTFILE
/home/user2/.bash_history
```

### NOTE

To view the contents of HISTFILE, we simply type `history`. Alternatively, we can specify the number of commands we want to see by passing an argument (the number of the most recent commands) to `history` as in `history 3`.

## HOME

This variable stores the absolute path of the current user's home directory and it is set when the user logs in.

This piece of code — from `~/.profile` — is self-explanatory (it sources "`$HOME/.bashrc`" if it exists):

```
# include .bashrc if it exists
if [ -f "$HOME/.bashrc" ]; then
. "$HOME/.bashrc"
fi
```

### NOTE

If you do not quite understand the `if` statement, do not worry: just refer to the lessons about shell scripting.

Remember that `~` is equivalent to `$HOME`:

```
$ echo ~; echo $HOME
/home/carol
/home/carol
```

**NOTE** | Commands can be concatenated with a semicolon (;).

We can also prove this with an `if` statement:

```
$ if [ ~ == "$HOME" ]; then echo "true"; else echo "false"; fi
true
```

**NOTE** | Remember: The equals sign `=` is used for variable assignment. `==` is used to test for equality.

## HOSTNAME

This variable stores the TCP/IP name of the host computer:

```
$ echo $HOSTNAME
debian
```

## HOSTTYPE

This stores the architecture of the host computer's processor:

```
$ echo $HOSTTYPE
x86_64
```

## LANG

This variable saves the locale of the system:

```
$ echo $LANG
en_UK.UTF-8
```

## LD\_LIBRARY\_PATH

This variable consists of a colon-separated set of directories where shared libraries are shared by programs:

```
$ echo $LD_LIBRARY_PATH
/usr/local/lib
```

## MAIL

This variable stores the file in which Bash searches for email:

```
$ echo $MAIL
/var/mail/carol
```

Another common value for this variable is `/var/spool/mail/$USER`.

## MAILCHECK

This variable stores a numeric value which indicates in seconds the frequency with which Bash checks for new mail:

```
$ echo $MAILCHECK
60
```

## PATH

This environment variable stores the list of directories where Bash looks for executable files when told to run any program. In our example machine this variable is set through the system-wide `/etc/profile` file:

```
if [ "`id -u`" -eq 0 ]; then
    PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
    PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
fi
export PATH
```

Through the `if` statement the identity of the user is tested and—depending on the result of the test (root or otherwise)—we will obtain one `PATH` or the other. Finally the chosen `PATH` is propagated with `export`.

Observe two things regarding the value of `PATH`:

- Directory names are written using absolute paths.
- The colon is used as a delimiter.

If we wanted to include the folder `/usr/local/sbin` in the `PATH` for regular users, we will modify the line so that it looks like this:

```
(...)
else
    PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/usr/local/sbin"
```

```
fi
export PATH
```

Now we can see how the value of the variable changes when we login as a regular user:

```
# su - carol
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/usr/local/sbin
```

**NOTE**

We could have also added `/usr/local/sbin` to the user's `PATH` at the command line by typing either `PATH=/usr/local/sbin:$PATH` or `PATH=$PATH:/usr/local/sbin`—the former making `/usr/local/sbin` the first directory to be searched for executable files; the latter making it the last.

**PS1**

This variable stores the value of the Bash prompt. In the following chunk of code (also from `/etc/profile`), the `if` statement tests for the identity of the user and gives them a very discrete prompt accordingly ( `#` for `root` or `$` for regular users):

```
if [ "`id -u`" -eq 0 ]; then
    PS1='# '
else
    PS1='$ '
fi
```

**NOTE** The `id` of `root` is `0`. Become `root` and test it yourself with `id -u`.

Other Prompt Variables include:

**PS2**

Normally set to `>` and used as a continuation prompt for long multiline commands.

**PS3**

Used as the prompt for the `select` command.

**PS4**

Normally set to `+` and used for debugging.

## SHELL

This variable stores the absolute path of the current shell:

```
$ echo $SHELL
/bin/bash
```

## USER

This stores the name of the current user:

```
$ echo $USER
carol
```

## Guided Exercises

1. Observe the variable assignment under the “Command(s)” column and indicate if the resulting variable is “Local” or “Global”:

Command(s)	Local	Global
<code>debian=mother</code>		
<code>ubuntu=deb-based</code>		
<code>mint=ubuntu-based;</code> <code>export mint</code>		
<code>export suse=rpm-based</code>		
<code>zorin=ubuntu-based</code>		

2. Study the “Command” and the “Output” and explain the meaning:

Command	Output	Meaning
<code>echo \$HISTCONTROL</code>	<code>ignoreboth</code>	
<code>echo ~</code>	<code>/home/carol</code>	
<code>echo \$DISPLAY</code>	<code>reptilium:0:2</code>	
<code>echo \$MAILCHECK</code>	<code>60</code>	
<code>echo \$HISTFILE</code>	<code>/home/carol/.bash_history</code>	

3. Variables are being set incorrectly under the “Wrong Command” column. Provide the missing information under “Right Command” and “Variable Reference” so that we get the “Expected Output”:

Wrong Command	Right Command	Variable Reference	Expected Output
<code>lizard =chameleon</code>			<code>chameleon</code>
<code>cool</code> <code>lizard=chameleon</code>			<code>chameleon</code>
<code>lizard=cha me leon</code>			<code>cha me leon</code>
<code>lizard=/**</code> <code>chameleon **/</code>			<code>/** chameleon **/</code>

Wrong Command	Right Command	Variable Reference	Expected Output
<code>win_path=C:\path\to\dir\</code>			<code>C:\path\to\dir\</code>

4. Consider the purpose and write the appropriate command:

Purpose	Command
Set the language of the current shell to Spanish UTF-8 ( <code>es_ES.UTF-8</code> ).	
Print the name of the current working directory.	
Reference the environment variable which stores the information about ssh connections.	
Set <code>PATH</code> to include <code>/home/carol/scripts</code> as the last directory to search for executables.	
Set the value of <code>my_path</code> to <code>PATH</code> .	
Set the value of <code>my_path</code> to that of <code>PATH</code> .	

5. Create a local variable named `mammal` and assign it the value `gnu`:

6. Using variable substitution, create another local variable named `var_sub` with the appropriate value so that when referenced via `echo $var_sub` we obtain: `The value of mammal is gnu`:

7. Turn `mammal` into an environment variable:

8. Search for it with `set` and `grep`:

9. Search for it with `env` and `grep`:

10. Create, in two consecutive commands, an environment variable named `BIRD` whose value is `penguin`:



11. Create, in a single command, an environment variable named `NEW_BIRD` whose value is `yellow-eyed penguin`:

12. Assuming you are `user2`, create a folder named `bin` in your home directory:

13. Type the command to add the `~/bin` folder to your `PATH` so that it is the first directory `bash` searches for binaries:

14. To guarantee the value of `PATH` remains unaltered across reboots, what piece of code—in the form of an `if` statement—would you put into `~/ .profile`?

## Explorational Exercises

1. `let`: more than arithmetic expression evaluation:

- Do a manpage or web search for `let` and its implications when setting variables and create a new local variable named `my_val` whose value is `10` — as a result of adding 5 and 5:

- Now create another variable named `your_val` whose value is `5` — as a result of dividing the value of `my_val` into 2:

2. The result of a command in a variable? Of course, that is possible; it is called *command substitution*. Investigate it and study the following function named `music_info`:

```
music_info(){
latest_music=`ls -l1t ~/Music | head -n 6`
echo -e "Your latest 5 music files:\n$latest_music"
}
```

The result of the command `ls -l1t ~/Music | head -n 6` becomes the value of the variable `latest_music`. Then the variable `latest_music` is referenced in the `echo` command (which outputs the total number of bytes occupied by the `Music` folder and the latest five music files stored in the `Music` folder — one per line).

Which of the following is a valid synonym for

```
latest_music=`ls -l1t ~/Music | head -n 6`
```

Option A:

```
latest_music=$(ls -l1t ~/Music| head -n 6)
```

Option B:

```
latest_music="(ls -l1t ~/Music| head -n 6)"
```

Option C:

```
latest_music=((ls -l1t ~/Music| head -n 6))
```

# Summary

In this lesson we learned:

- Variables are a very important part of the shell environment as they are used by the shell itself as well as by other programs.
- How to assign and reference variables.
- The differences between *local* and *global* (or *environment*) variables.
- How to make variables *readonly*.
- How to turn a local variable into an environment variable with the `export` command.
- How to list all environment variables.
- How to run a program in a modified environment.
- How to make variables persistent with the help of startup scripts.
- Some common environment variables: `DISPLAY`, `HISTCONTROL`, `HISTSIZE`, `HISTFILESIZE`, `HISTFILE`, `HOME`, `HOSTNAME`, `HOSTTYPE`, `LANG`, `LD_LIBRARY_PATH`, `MAIL`, `MAILCHECK`, `PATH`, `PS1` (and other prompt variables), `SHELL` and `USER`.
- The meaning of the tilde (`~`).
- The very basics of `if` statements.

Commands used in this lesson:

## **echo**

Reference a variable.

## **ls**

List directory contents.

## **readonly**

Make variables immutable. List all readonly variables in current session.

## **set**

List all variables and functions in current session.

## **grep**

Print lines matching a pattern.

**bash**

Launch a new shell

**unset**

Unset variables.

**export**

Turn a local variable into an environment variable. List environment variables.

**env**

List environment variables. Run a program in a modified environment.

**printenv**

List environment variables. Reference a variable.

**chmod**

Change mode bits of a file, for example make it executable.

**history**

List previous commands.

**su**

Change user ID or become superuser.

**id**

Print user ID.

## Answers to Guided Exercises

1. Observe the variable assignment under the “Command(s)” column and indicate if the resulting variable is “Local” or “Global”:

Command(s)	Local	Global
<code>debian=mother</code>	Yes	No
<code>ubuntu=deb-based</code>	Yes	No
<code>mint=ubuntu-based;</code> <code>export mint</code>	No	Yes
<code>export suse=rpm-based</code>	No	Yes
<code>zorin=ubuntu-based</code>	Yes	No

2. Study the “Command” and the “Output” and explain the meaning:

Command	Output	Meaning
<code>echo \$HISTCONTROL</code>	<code>ignoreboth</code>	Both duplicate commands and those starting with a space will not be saved in history.
<code>echo ~</code>	<code>/home/carol</code>	The HOME of carol is <code>/home/carol</code> .
<code>echo \$DISPLAY</code>	<code>reptilium:0:2</code>	<code>reptilium</code> machine has a X server running and we are using the second screen of the display.
<code>echo \$MAILCHECK</code>	<code>60</code>	Mail will be checked every minute.
<code>echo \$HISTFILE</code>	<code>/home/carol/.bash_history</code>	history will be saved in <code>/home/carol/.bash_history</code> .

3. Variables are being set incorrectly under the “Wrong Command” column. Provide the missing information under “Right Command” and “Variable Reference” so that we get the “Expected Output”:

Wrong Command	Right Command	Variable Reference	Expected Output
<code>lizard =chameleon</code>	<code>lizard=chameleon</code>	<code>echo \$lizard</code>	<code>chameleon</code>
<code>cool lizard=chameleon</code>	<code>cool_lizard=chameleon (for example)</code>	<code>echo \$cool_lizard</code>	<code>chameleon</code>
<code>lizard=cha me leon</code>	<code>lizard="cha me leon"</code> or <code>lizard='cha me leon'</code>	<code>echo \$lizard</code>	<code>cha me leon</code>
<code>lizard=/** chameleon **/</code>	<code>lizard="/** chameleon **/"</code> or <code>lizard='/** chameleon **/'</code>	<code>echo "\$lizard"</code>	<code>/** chameleon **/</code>
<code>win_path=C:\path\to\dir\ o\dir\</code>	<code>win_path=C:\\path\ \to\\dir\\</code>	<code>echo \$win_path</code>	<code>C:\path\to\dir\ o\dir\</code>

4. Consider the purpose and write the appropriate command:

Purpose	Command
Set the language of the current shell to Spanish UTF-8 (es_ES.UTF-8).	<code>LANG=es_ES.UTF-8</code>
Print the name of the current working directory	<code>echo \$PWD</code> or <code>pwd</code>
Reference the environment variable which stores the information about ssh connections	<code>echo \$SSH_CONNECTION</code>
Set <code>PATH</code> to include <code>/home/carol/scripts</code> as the last directory to search for executables.	<code>PATH=\$PATH:/home/carol/scripts</code>
Set the value of <code>my_path</code> to <code>PATH</code> .	<code>my_path=PATH</code>
Set the value of <code>my_path</code> to that of <code>PATH</code> .	<code>my_path=\$PATH</code>

5. Create a local variable named `mammal` and assign it the value `gnu`:

```
mammal=gnu
```

6. Using variable substitution, create another local variable named `var_sub` with the appropriate value so that when referenced via `echo $var_sub` we obtain The value of `mammal` is `gnu`:

```
var_sub="The value of mammal is $mammal"
```

7. Turn `mammal` into an environment variable:

```
export mammal
```

8. Search for it with `set` and `grep`:

```
set | grep mammal
```

9. Search for it with `env` and `grep`:

```
env | grep mammal
```

10. Create, in two consecutive commands, an environment variable named `BIRD` whose value is `penguin`:

```
BIRD=penguin; export BIRD
```

11. Create, in a single command, an environment variable named `NEW_BIRD` whose value is `yellow-eyed penguin`:

```
export NEW_BIRD="yellow-eyed penguin"
```

OR

```
export NEW_BIRD='yellow-eyed penguin'
```

12. Assuming you are `user2`, use `mkdir` to create a folder named `bin` in your home directory:

```
mkdir ~/bin
```

OR

```
mkdir /home/user2/bin
```



OR

```
mkdir $HOME/bin
```

13. Type the command to add the `~/bin` folder to your `PATH` so that it is the first directory bash searches for binaries:

```
PATH="$HOME/bin:$PATH"
```

`PATH=~/bin:$PATH` or `PATH=/home/user2/bin:$PATH` are equally valid.

14. To guarantee the value of `PATH` remains unaltered across reboots, what piece of code—in the form of an `if` statement—would you put into `~/.profile`?

```
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

# Answers to Explorational Exercises

1. `let`: more than arithmetic expression evaluation:

- Do a manpage or web search for `let` and its implications when setting variables and create a new local variable named `my_val` whose value is `10` — as a result of adding 5 and 5:

```
let "my_val = 5 + 5"
```

or

```
let 'my_val = 5 + 5'
```

- Now create another variable named `your_val` whose value is `5` — as a result of dividing the value of `my_val` into 2:

```
let "your_val = $my_val / 2"
```

or

```
let 'your_val = $my_val / 2'
```

2. The result of a command in a variable? Of course, that is possible; it is called *command substitution*. Investigate it and study the following function named `music_info`:

```
music_info(){
latest_music=`ls -l1t ~/Music | head -n 6`
echo -e "Your latest 5 music files:\n$latest_music"
}
```

The result of the command `ls -l1t ~/Music | head -n 6` becomes the value of the variable `latest_music`. Then the variable `latest_music` is referenced in the `echo` command (which outputs the total number of bytes occupied by the `Music` folder and the latest five music files stored in the `Music` folder — one per line).

Which of the following is a valid synonym for

```
latest_music=`ls -l1t ~/Music | head -n 6`
```

It is option A:

```
latest_music=$(ls -l1t ~/Music| head -n 6)
```



## 105.1 Lesson 3

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	105 Shells and Shell Scripting
<b>Objective:</b>	105.1 Customize and use the shell environment
<b>Lesson:</b>	3 of 3

### Introduction

After going through shells, startup scripts and variables in the previous lessons, we will round the whole topic of customizing the shell off by having a look at two very interesting shell elements: *aliases* and *functions*. In fact the entire group of variables, aliases and functions—and their influence on one another—is what makes up the shell environment.

The main strength of these two flexible and time-saving shell facilities has to do with the concept of encapsulation: they offer the possibility of putting together—under a single command—a series of repetitive or recurrent commands.

### Creating Aliases

An alias is a substitute name for another command(s). It can run like a regular command, but instead executes another command according to the alias definition.

The syntax for declaring aliases is quite straightforward. Aliases are declared by writing the keyword `alias` followed by the alias assignment. In turn, the alias assignment consists of the *alias name*, an *equal sign* and one or more *commands*:

```
alias alias_name=command(s)
```

For example:

```
$ alias oldshell=sh
```

This awkward alias will start an instance of the original `sh` shell when the user types `oldshell` into the terminal:

```
$ oldshell
$
```

The power of aliases lies in that they allow us to write short versions of long commands:

```
$ alias ls='ls --color=auto'
```

**NOTE** For information about `ls` and its colors, type `man dir_colors` into the terminal.

Likewise, we can create aliases for a series of concatenated commands—the semicolon (`;`) is used as a delimiter. We can, for instance, have an alias that gives us information about the location of the `git` executable and its version:

```
$ alias git_info='which git;git --version'
```

To invoke an alias, we type its name into the terminal:

```
$ git_info
/usr/bin/git
git version 2.7.4
```

The `alias` command will produce a listing of all available aliases in the system:

```
$ alias
alias git-info='which git;git --version'
alias ls='ls --color=auto'
alias oldshell='sh'
```

The `unalias` command removes aliases. We can, for instance, `unalias git-info` and see how it disappears from the listing:

```
$ unalias git-info
$ alias
alias ls='ls --color=auto'
alias oldshell='sh'
```

As we saw with the `alias hi='echo We salute you.'` in a previous lesson, we must enclose commands in quotes (either single or double) when—as a result of having arguments or parameters—they contain spaces:

```
$ alias greet='echo Hello world!'
$ greet
Hello world!
```

Commands with spaces include also those with options:

```
$ alias ll='ls -al'
```

Now `ll` will list all files—including the hidden ones (`a`)—in the long format (`l`).

We can reference variables in aliases:

```
$ reptile=uromastyx
$ alias greet='echo Hello $reptile!'
$ greet
Hello uromastyx!
```

The variable can also be assigned within the alias:

```
$ alias greet='reptile=tortoise; echo Hello $reptile!'
$ greet
Hello tortoise!
```

We can escape an alias with `\`:

```
$ alias where?='echo $PWD'
```

```
$ where?
/home/user2
$ \where?
-bash: where?: command not found
```

Escaping an alias is useful when an alias has the same name as a regular command. In this case, the alias takes precedence over the original command, which, however, is still be accessible by escaping the alias.

Likewise, we can put an alias inside another alias:

```
$ where?
/home/user2
$ alias my_home=where?
$ my_home
/home/user2
```

On top of that, we can also put a function inside an alias as you will be shown below.

## Expansion and Evaluation of Quotes in Aliases

When using quotes with environment variables, single quotes make the expansion dynamic:

```
$ alias where?='echo $PWD'
$ where?
/home/user2
$ cd Music
$ where?
/home/user2/Music
```

However, with double quotes the expansion is done statically:

```
$ alias where?="echo $PWD"
$ where?
/home/user2
$ cd Music
$ where?
/home/user2
```

## Persistence of Aliases: Startup Scripts

Just as with variables, for our aliases to gain persistence, we must put them into initialization scripts that are sourced at startup. As we already know, a good file for users to put their personal aliases in is `~/.bashrc`. You will probably find some aliases there already (most of them commented out and ready to be used by removing the leading `#`):

```
$ grep alias .bashrc
# enable color support of ls and also add handy aliases
  alias ls='ls --color=auto'
  #alias dir='dir --color='
  #alias vdir='vdir --color='
  #alias grep='grep --color='
  #alias fgrep='fgrep --color='
  #alias egrep='egrep --color='
# some more ls aliases
#ll='ls -al'
#alias la='ls -A'
#alias l='ls -CF'
# ~/.bash_aliases, instead of adding them here directly.
if [ -f ~/.bash_aliases ]; then
  . ~/.bash_aliases
```

As you can read in the last three lines, we are offered the possibility of having our own alias-dedicated file — `~/.bash_aliases` — and have it sourced by `.bashrc` with every system start. So we can go for that option and create and populate such file:

```
#####
# .bash_aliases:
# a file to be populated by the user's personal aliases (and sourced by ~/.bashrc).
#####
alias git_info='which git;git --version'
alias greet='echo Hello world!'
alias ll='ls -al'
alias where?='echo $PWD'
```

## Creating Functions

Compared to aliases, functions are more programmatic and flexible, specially when it comes to exploiting the full potential of *Bash* special built-in variables and positional parameters. They are also great to work with flow control structures such as loops or conditionals. We can think of a



function as a command which includes logic through blocks or collections of other commands.

## Two Syntaxes for Creating Functions

There are two valid syntaxes to define functions.

### Using the keyword `function`

On the one hand, we can use the keyword `function`, followed by the name of the function and the commands between curly brackets:

```
function function_name {  
  command #1  
  command #2  
  command #3  
  .  
  .  
  .  
  command #n  
}
```

### Using `()`

On the other, we can leave out the keyword `function` and use two brackets right after the name of the function instead:

```
function_name() {  
  command #1  
  command #2  
  command #3  
  .  
  .  
  .  
  command #n  
}
```

It is commonplace to put functions in files or scripts. However, they can also be written directly into the shell prompt with each command on a different line — note PS2(>) indicating a new line after a line break:

```
$ greet() {  
> greeting="Hello world!"
```

```
> echo $greeting
> }
```

Whatever the case — and irrespective of the syntax we choose —, if we decide to skip line breaks and write a function in just one line, commands must be separated by semicolons (note the semicolon after the last command too):

```
$ greet() { greeting="Hello world!"; echo $greeting; }
```

`bash` did not complain when we pressed Enter, so our function is ready to be invoked. To invoke a function, we must type its name into the terminal:

```
$ greet
Hello world!
```

Just as with variables and aliases, if we want functions to be persistent across system reboots we have to put them into shell initialization scripts such as `/etc/bash.bashrc` (global) or `~/.bashrc` (local).

### WARNING

After adding aliases or functions to any startup script file, you should source such files with either `.` or `source` for the changes to take effect if you do not want to logout and back in again or reboot the system.

## Bash Special Built-in Variables

The *Bourne Again Shell* comes with a set of special variables which are particularly useful for functions and scripts. These are special because they can only be referenced — not assigned. Here is a list of the most relevant ones:

### \$?

This variable's reference expands to the result of the last command run. A value of `0` means success:

```
$ ps aux | grep bash
user2      420  0.0  0.4  21156  5012 pts/0    Ss   17:10   0:00 -bash
user2      640  0.0  0.0  12784   936 pts/0    S+   18:04   0:00 grep bash
$ echo $?
0
```

A value other than 0 means error:

```
user1@debian:~$ ps aux |rep bash
-bash: rep: command not found
user1@debian:~$ echo $?
127
```

**\$\$**

It expands to the shell PID (process ID):

```
$ ps aux | grep bash
user2      420  0.0  0.4  21156  5012 pts/0    Ss   17:10   0:00 -bash
user2      640  0.0  0.0  12784   936 pts/0    S+   18:04   0:00 grep bash
$ echo $$
420
```

**\$!**

It expands to the PID of the last background job:

```
$ ps aux | grep bash &
[1] 663
$ user2      420  0.0  0.4  21156  5012 pts/0    Ss+  17:10   0:00 -bash
user2      663  0.0  0.0  12784   972 pts/0    S    18:08   0:00 grep bash
^C
[1]+  Done                  ps aux | grep bash
$ echo $!
663
```

**NOTE** Remember, the ampersand (&) is used to start processes in the background.

## Positional parameters \$0 through \$9

They expand to the parameters or arguments being passed to the function (alias or script) — \$0 expanding to the name of the script or shell.

Let us create a function to demonstrate positional parameters — note PS2 (>) indicating new lines after line breaks:

```
$ special_vars() {
> echo $0
> echo $1
```

```
> echo $2
> echo $3
}
```

Now, we will invoke the function (`special_vars`) passing three parameters to it (`debian`, `ubuntu`, `zorin`):

```
$ special_vars debian ubuntu zorin
-bash
debian
ubuntu
zorin
```

It worked as expected.

Although passing positional parameters to aliases is technically possible, it is not at all functional since—with aliases—positional parameters are always passed at the end:

#### WARNING

```
$ alias great_editor='echo $1 is a great text editor'
$ great_editor emacs
is a great text editor emacs
```

Other Bash special built-in variables include:

**\$#**

It expands to the number of arguments passed to the command.

**\$@, \$\***

They expand to the arguments passed to the command.

**\$\_**

It expands to the last parameter or the name of the script (amongst other things; see `man bash` to find out more!):

## Variables in Functions

Of course, variables can be used within functions.

To prove it, this time we will create a new empty file called `funed` and put the following function

into it:

```
editors() {  
  
    editor=emacs  
  
    echo "My editor is: $editor. $editor is a fun text editor."  
}
```

As you may have guessed by now, we must source the file first to be able to invoke the function:

```
$ . funed
```

And now we can test it:

```
$ editors  
My editor is emacs. emacs is a fun text editor.
```

As you can appreciate, for the `editors` function to work properly, the `editor` variable must first be set. The scope of that variable is local to the current shell and we can reference it as long as the session lasts:

```
$ echo $editor  
emacs
```

Together with local variables we can also include environment variables in our function:

```
editors() {  
  
    editor=emacs  
  
    echo "The text editor of $USER is: $editor."  
}  
  
editors
```

Note how this time we decided to call the function from within the file itself (`editors` in the last line). That way, when we source the file, the function will also be invoked — all at once:

```
$ . funed
The text editor of user2 is: emacs.
```

## Positional Parameters in Functions

Something similar occurs with positional parameters.

We can pass them to functions from within the file or script (note the last line: `editors tortoise`):

```
editors() {
    editor=emacs

    echo "The text editor of $USER is: $editor."
    echo "Bash is not a $1 shell."
}

editors tortoise
```

We source the file and prove it works:

```
$ . funed
The text editor of user2 is: emacs.
Bash is not a tortoise shell.
```

And we can also pass positional parameters to functions at the command line. To prove it, we get rid of the last line of the file:

```
editors() {
    editor=emacs

    echo "The text editor of $USER is: $editor."
    echo "Bash is not a $1 shell."
}
```

Then, we have to source the file:

```
$ . funed
```

Finally, we invoke the function with `tortoise` as the positional parameter `$1` at the command line:

```
$ editors tortoise
The text editor of user2 is: emacs.
Bash is not a tortoise shell.
```

## Functions in Scripts

Functions are mostly found in Bash scripts.

Turning our `funed` file into a script (we will name it `funed.sh`) is really a piece of cake:

```
#!/bin/bash

editors() {

editor=emacs

echo "The text editor of $USER is: $editor."
echo "Bash is not a $1 shell."
}

editors tortoise
```

That is it! We only added two lines:

- The first line is the *shebang* and defines what program is going to interpret the script: `#!/bin/bash`. Curiously enough, that program is `bash` itself.
- The last line is simply the invocation of the function.

Now there is only one thing left — we have to make the script executable:

```
$ chmod +x funed.sh
```

And now it is ready to be executed:

```
$ ./funed.sh
The text editor of user2 is: emacs.
Bash is not a tortoise shell.
```

**NOTE** You will learn all about *shell scripting* in the next few lessons.

## A Function within an Alias

As said above, we can put a function inside an alias:

```
$ alias great_editor='gr8_ed() { echo $1 is a great text editor; unset -f gr8_ed; }; gr8_ed'
```

This long alias value deserves an explanation. Let us break it down:

- First there is the function itself: `gr8_ed() { echo $1 is a great text editor; unset -f gr8_ed; }`
- The last command in the function — `unset -f gr8_ed` — unsets the function so that it does not remain in the present `bash` session after the alias is called.
- Last but not least, to have a successful alias invocation, we must first invoke the function too: `gr8_ed`.

Let us invoke the alias and prove it works:

```
$ great_editor emacs
emacs is a great text editor
```

As shown in `unset -f gr8_ed` above, the `unset` command is not only used to unset variables, but also functions. In fact, there are specific switches or options:

### **unset -v**

for variables

### **unset -f**

for functions

If used without switches, `unset` will try to unset a variable first and — if it fails — then it will try to unset a function.



## A Function within a Function

Now say we want to communicate two things to `user2` every time she logs into the system:

- Say hello and recommend/praise a text editor.
- Since she is starting to put a lot of Matroska video files in its `$HOME/Video` folder, we also want to give her a warning.

To accomplish that purpose, we have put the following two functions into `/home/user2/.bashrc`:

The first function (`check_vids`) does the checking on `.mkv` files and the warning:

```
check_vids() {
    ls -l ~/Video/*.mkv > /dev/null 2>&1
    if [ "$?" = "0" ];then
        echo -e "Remember, you must not keep more than 5 video files in your Video
folder.\nThanks."
    else
        echo -e "You do not have any videos in the Video folder. You can keep up to 5.\nThanks."
    fi
}
```

`check_vids` does three things:

- It lists the `mkv` files in `~/Video` sending the output—and any errors—to the so-called *bit-bucket* (`/dev/null`).
- It tests the output of the previous command for success.
- Depending on the result of the test, it echoes one of two messages.

The second function is a modified version of our `editors` function:

```
editors() {
    editor=emacs

    echo "Hi, $USER!"
    echo "$editor is more than a text editor!"

    check_vids
}
```

```
editors
```

It is important to observe two things:

- The last command of `editors` invokes `check_vids` so both functions get chained: The greeting, praise and the checking and warning are executed in sequence.
- `editors` itself is the entry point to the sequence of functions so it is invoked in the last line (`editors`).

Now, let us log in as `user2` and prove it works:

```
# su - user2
Hi, user2!
emacs is more than a text editor!
Remember, you must not keep more than 5 video files in your Video folder.
Thanks.
```

## Guided Exercises

1. Complete the table with “Yes” or “No” considering the capabilities of aliases and functions:

Feature	Aliases?	Functions?
Local variables can be used		
Environment variables can be used		
Can be escaped with <code>\</code>		
Can be recursive		
Very productive when used with positional parameters		

2. Enter the command that list all aliases in your system:

3. Write an alias named `logg` that lists all `ogg` files in `~/Music` — one per line:

4. Invoke the alias to prove it works:

5. Now, modify the alias so that it echoes out the session’s user and a colon before the listing:

6. Invoke it again to prove this new version also works:

7. List all aliases again and check your `logg` alias appears in the listing:

8. Remove the alias:

9. Study the columns “Alias Name” and “Aliased Command(s)” and assign the aliases to their values correctly:

Alias Name	Aliased Command(s)	Alias Assignment
b	bash	
bash_info	which bash + echo "\$BASH_VERSION"	
kernel_info	uname -r	
greet	echo Hi, \$USER!	
computer	pc=slimbook + echo My computer is a \$pc	

10. As root, write a function called `my_fun` in `/etc/bash.bashrc`. The function must say hello to the user and tell them what their path is. Invoke it so that the user gets both messages every time they log in:

11. Login as `user2` to check it works:

12. Write the same function in just one line:

13. Invoke the function:

14. Unset the function:

15. This is a modified version of the `special_vars` function:

```
$ special_vars2() {  
> echo $#  
> echo $_  
> echo $1  
> echo $4  
> echo $6  
> echo $7  
> echo $_  
> echo $@  
> echo $?
```

```
> }
```

This is the command we use to invoke it:

```
$ special_vars2 crying cockles and mussels alive alive oh
```

Guess the outputs:

Reference	Value
echo \$#	
echo \$_	
echo \$1	
echo \$4	
echo \$6	
echo \$7	
echo \$_	
echo \$@	
echo \$?	

16. Based on the sample function (`check_vids`) in the section “A Function within a Function”, write a function named `check_music` to include into a `bash` startup script that accepts positional parameters so that we can modify easily:
- the type of file being checked: `ogg`
  - the directory in which files are saved: `~/Music`
  - the type of file being kept: `music`
  - the number of files being saved: `7`

---

## Explorational Exercises

1. Read-only functions are those whose contents we cannot modify. Do a research on *readonly functions* and complete the following table:

Function Name	Make it readonly	List all readonly Functions
my_fun		

2. Search the web for how to modify `PS1` and anything else you may need to write a function called `fyi` (to be placed in a startup script) which gives the user the following information:
  - name of the user
  - home directory
  - name of the host
  - operating system type
  - search path for executables
  - mail directory
  - how often mail is checked
  - how many shells deep the current session is
  - prompt (you should modify it so that it shows `<user>@<host-date>`)

---

# Summary

In this lesson you learned:

- Both aliases and functions are important features of the shell that allow us to encapsulate recurrent blocks of code.
- Aliases are useful to have shorter versions of long and/or complicated commands.
- Functions are procedures that implement logic and allow us to automate tasks, specially when used in scripts.
- The syntax to write aliases and functions.
- How to concatenate various commands by means of the semicolon (;).
- How to properly use quotes with aliases.
- How to make aliases and functions persistent.
- Bash special built-in variables: `?`, `$$`, `$!`, positional parameters (`$0-$9`),  `$#`,  `$@`,  `$*` and  `$_`.
- How to use variables and positional parameters with functions.
- How to use functions in scripts.
- How to invoke a function from an alias.
- How to invoke a function from another function.
- The basics for creating a bash script.

Commands and keywords used in this lesson:

## **alias**

Create aliases.

## **unalias**

Remove aliases.

## **cd**

Change directory.

## **grep**

Print lines matching a pattern.

## **function**

Shell keyword to create functions.

.

Source a file.

### **source**

Source a file.

### **ps**

Report a snapshot of the current processes.

### **echo**

Display a line of text.

### **chmod**

Change mode bits of a file, for example make it executable.

### **unset**

Unset variables and functions.

### **su**

Change user ID or become superuser.



## Answers to Guided Exercises

1. Complete the table with “Yes” or “No” considering the capabilities of aliases and functions:

Feature	Aliases?	Functions?
Local variables can be used	Yes	Yes
Environment variables can be used	Yes	Yes
Can be escaped with \	Yes	No
Can be recursive	Yes	Yes
Very productive when used with positional parameters	No	Yes

2. Enter the command that list all aliases in your system:

```
alias
```

3. Write an alias named `logg` that lists all `ogg` files in `~/Music` — one per line:

```
alias logg='ls -1 ~/Music/*ogg'
```

4. Invoke the alias to prove it works:

```
logg
```

5. Now, modify the alias so that it echoes out the session’s user and a colon before the listing:

```
alias logg='echo $USER:; ls -1 ~/Music/*ogg'
```

6. Invoke it again to prove this new version also works:

```
logg
```

7. List all aliases again and check your `logg` alias appears in the listing:

```
alias
```

8. Remove the alias:

```
unalias logg
```

9. Study the columns “Alias Name” and “Aliased Command(s)” and assign the aliases to their values correctly:

Alias Name	Aliased Command(s)	Alias Assignment
b	bash	alias b=bash
bash_info	which bash + echo "\$BASH_VERSION"	alias bash_info='which bash; echo "\$BASH_VERSION"'
kernel_info	uname -r	alias kernel_info='uname -r'
greet	echo Hi, \$USER!	alias greet='echo Hi, \$USER'
computer	pc=slimbook + echo My computer is a \$pc	alias computer='pc=slimbook; echo My computer is a \$pc'

**NOTE** | Single quotes can also be replaced by double ones.

10. As root, write a function called my\_fun in /etc/bash.bashrc. The function must say hello to the user and tell them what their path is. Invoke it so that the user gets both messages every time they log in:

Option A:

```
my_fun() {
  echo Hello, $USER!
  echo Your path is: $PATH
}
my_fun
```

Option B:

```
function my_fun {  
echo Hello, $USER!  
echo Your path is: $PATH  
}  
my_fun
```

11. Login as `user2` to check it works:

```
su - user2
```

12. Write the same function in just one line:

Option A:

```
my_fun() { echo "Hello, $USER!"; echo "Your path is: $PATH"; }
```

Option B:

```
function my_fun { echo "Hello, $USER!"; echo "Your path is: $PATH"; }
```

13. Invoke the function:

```
my_fun
```

14. Unset the function:

```
unset -f my_fun
```

15. This is a modified version of the `special_vars` function:

```
$ special_vars2() {  
> echo $#  
> echo $_  
> echo $1  
> echo $4  
> echo $6
```

```
> echo $7
> echo $_
> echo $@
> echo $?
> }
```

This is the command we use to invoke it:

```
$ special_vars2 crying cockles and mussels alive alive oh
```

Guess the outputs:

Reference	Value
echo \$#	7
echo \$_	7
echo \$1	crying
echo \$4	mussels
echo \$6	alive
echo \$7	oh
echo \$_	oh
echo \$@	crying cockles and mussels alive alive oh
echo \$?	0

16. Based on the sample function (`check_vids`) in section “A Function within a Function”, write a function named `check_music` to include into a `bash` startup script that accepts positional parameters so that we can modify easily:
- the type of file being checked: `ogg`
  - the directory in which files are saved: `~/Music`
  - the type of file being kept: `music`
  - the number of files being saved: `7`

```
check_music() {
    ls -1 ~/$1/*.$2 > ~/.mkv.log 2>&1
    if [ "$?" = "0" ];then
```

```
    echo -e "Remember, you must not keep more than $3 $4 files in your $1
folder.\nThanks."
    else
        echo -e "You do not have any $4 files in the $1 folder. You can keep up to
$3.\nThanks."
    fi
}

check_music Music ogg 7 music
```

## Answers to Explorational Exercises

1. Read-only functions are those whose contents we cannot modify. Do a research on *readonly functions* and complete the following table:

Function Name	Make it readonly	List all readonly Functions
my_fun	readonly -f my_fun	readonly -f

2. Search the web for how to modify PS1 and anything else you may need to write a function called `fyi` (to be placed in a startup script) which gives the user the following information:
- name of the user
  - home directory
  - name of the host
  - operating system type
  - search path for executables
  - mail directory
  - how often mail is checked
  - how many shells deep the current session is
  - prompt (you should modify it so that it shows `<user>@<host-date>`)

```
fyi() {
    echo -e "For your Information:\n
    Username: $USER
    Home directory: $HOME
    Host: $HOSTNAME
    Operating System: $OSTYPE
    Path for executable files: $PATH
    Your mail directory is $MAIL and is searched every $MAILCHECK seconds.
    The current level of your shell is: $SHLVL"
    PS1="\u@\h-\d "
}

fyi
```



**Linux  
Professional  
Institute**

## 105.2 Customize or write simple scripts

### Reference to LPI objectives

[LPIC-1 5.0, Exam 102, Objective 105.2](#)

### Weight

4

### Key knowledge areas

- Use standard sh syntax (loops, tests).
- Use command substitution.
- Test return values for success or failure or other information provided by a command.
- Execute chained commands.
- Perform conditional mailing to the superuser.
- Correctly select the script interpreter through the shebang (#!) line.
- Manage the location, ownership, execution and suid-rights of scripts.

### Partial list of the used files, terms and utilities

- `for`
- `while`
- `test`
- `if`
- `read`
- `seq`
- `exec`

- `||`
- `&&`





## 105.2 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	105 Shells and Shell Scripting
<b>Objective:</b>	105.2 Customize or write simple scripts
<b>Lesson:</b>	1 of 2

### Introduction

The Linux shell environment allows the use of files — called *scripts* — containing commands from any available program in the system combined with shell builtin commands to automate a user's and/or a system's custom tasks. Indeed, many of the operating system maintenance tasks are performed by scripts consisting of sequences of commands, decision structures and conditional loops. Although scripts are most of the time intended for tasks related to the operating system itself, they are also useful for user oriented tasks, like mass renaming of files, data collecting and parsing, or any otherwise repetitive command line activities.

Scripts are nothing more than text files that behave like programs. An actual program — the interpreter — reads and executes the instructions listed in the script file. The interpreter can also start an interactive session where commands — including scripts — are read and executed as they are entered, as is the case with Linux shell sessions. Script files can group those instructions and commands when they grow too complex to be implemented as an alias or a custom shell function. Furthermore, script files can be maintained like conventional programs and, being just text files, they can be created and modified with any simple text editor.

## Script Structure and Execution

Basically, a script file is an ordered sequence of commands that must be executed by a corresponding command interpreter. How an interpreter reads a script file varies and there are distinct ways of doing so in a Bash shell session, but the default interpreter for a script file will be the one indicated in the first line of the script, just after the characters `#!` (known as *shebang*). In a script with instructions for the Bash shell, the first line should be `#!/bin/bash`. By indicating this line, the interpreter for all the instructions in the file will be `/bin/bash`. Except for the first line, all other lines starting with the hash character `#` will be ignored, so they can be used to place reminders and comments. Blank lines are also ignored. A very concise shell script file can therefore be written as follows:

```
#!/bin/bash

# A very simple script

echo "Cheers from the script file! Current time is: "

date +%H:%M
```

This script has only two instructions for the `/bin/bash` interpreter: the builtin command `echo` and the command `date`. The most basic way to run a script file is to execute the interpreter with the script path as the argument. So, assuming the previous example was saved in a script file named `script.sh` in the current directory, it will be read and interpreted by Bash with the following command:

```
$ bash script.sh
Cheers from the script file! Current time is:
10:57
```

Command `echo` will automatically add a new line after displaying the content, but the option `-n` will suppress this behaviour. Thus, using `echo -n` in the script will make the output of both commands to appear in the same line:

```
$ bash script.sh
Cheers from the script file! Current time is: 10:57
```

Although not required, the suffix `.sh` helps to identify shell scripts when listing and searching for files.

**TIP**

Bash will call whatever command is indicated after the `#!` as the interpreter for the script file. It may be useful, for example, to employ the shebang for other scripting languages, like *Python* (`#!/usr/bin/python`), *Perl* (`#!/usr/bin/perl`) or *awk* (`#!/usr/bin/awk`).

If the script file is intended to be executed by other users in the system, it is important to check if the proper reading permissions are set. The command `chmod o+r script.sh` will give reading permission to all users in the system, allowing them to execute `script.sh` by placing the path to the script file as the argument of command `bash`. Alternatively, the script file may have the execution bit permission set so the file can be executed as a conventional command. The execution bit is activated on the script file with the command `chmod`:

```
$ chmod +x script.sh
```

With the execution bit enabled, the script file named `script.sh` in the current directory can be executed directly with the command `./script.sh`. Scripts placed in a directory listed in the `PATH` environment variable will also be accessible without their complete path.

**WARNING**

A script performing restricted actions may have its SUID permission activated, so ordinary users can also run the script with root privileges. In this case, it is very important to ensure that no user other than root has the permission to write in the file. Otherwise, an ordinary user could modify the file to perform arbitrary and potentially harmful operations.

The placement and indentation of commands in script files are not too rigid. Every line in a shell script will be executed as an ordinary shell command, in the same sequence as the line appears in the script file, and the same rules that apply to the shell prompt also apply to each script line individually. It is possible to place two or more commands in the same line, separated by semicolons:

```
echo "Cheers from the script file! Current time is:" ; date +%H:%M
```

Although this format might be convenient at times, its usage is optional, as sequential commands can be placed one command per line and they will be executed just as they were separated by semicolons. In other words, the semicolon can be replaced by a new line character in Bash script files.

When a script is executed, the commands contained therein are not executed directly in the current session, but instead they are executed by a new Bash process, called a *sub-shell*. It prevents the script from overwriting the current session's environment variables and from

leaving unattended modifications in the current session. If the goal is to run the script's contents in the current shell session, then it should be executed with `source script.sh` or `. script.sh` (note that there is a space between the dot and the script name).

As it happens with the execution of any other command, the shell prompt will only be available again when the script ends its execution and its exit status code will be available in the `?` variable. To change this behaviour, so the current shell also ends when the script ends, the script—or any other command—can be preceded by the `exec` command. This command will also replace the exit status code of the current shell session with its own.

## Variables

Variables in shell scripts behave in the same way as in interactive sessions, given that the interpreter is the same. For example, the format `SOLUTION=42` (without spaces around the equal sign) will assign the value `42` to the variable named `SOLUTION`. By convention, uppercase letters are used for variable names, but it's not mandatory. Variable names can not, however, start with non-alphabetical characters.

In addition to the ordinary variables created by the user, Bash scripts also have a set of special variables called *parameters*. Unlike ordinary variables, parameter names start with a non-alphabetical character that designates its function. Arguments passed to a script and other useful information are stored in parameters like `$0`, `$*`, `$?`, etc, where the character following the dollar sign indicates the information to be fetched:

**`$*`**

All the arguments passed to the script.

**`$@`**

All the arguments passed to the script. If used with double quotes, as in `"$@"`, every argument will be enclosed by double quotes.

**`$#`**

The number of arguments.

**`$0`**

The name of the script file.

**`$!`**

PID of the last executed program.

**\$\$**

PID of the current shell.

**\$?**

Numerical exit status code of the last finished command. For POSIX standard processes, a numerical value of `0` means that the last command was successfully executed, which also applies to shell scripts.

A *positional parameter* is a parameter denoted by one or more digits, other than the single digit `0`. For example, the variable `$1` corresponds to the first argument given to the script (positional parameter one), `$2` corresponds to the second argument, and so on. If the position of a parameter is greater than nine, it must be referenced with curly braces, as in `${10}`, `${11}`, etc.

Ordinary variables, on the other hand, are intended to store manually inserted values or the output generated by other commands. Command `read`, for example, can be used inside the script to ask the user for input during script execution:

```
echo "Do you want to continue (y/n)?"
read ANSWER
```

The returned value will be stored in the `ANSWER` variable. If the name of the variable is not supplied, the variable name `REPLY` will be used by default. It is also possible to use command `read` to read more than one variable simultaneously:

```
echo "Type your first name and last name:"
read NAME SURNAME
```

In this case, each space separated term will be assigned to the variables `NAME` and `SURNAME` respectively. If the number of given terms is greater than the number of variables, the exceeding terms will be stored in the last variable. `read` itself can display the message to the user with option `-p`, making the `echo` command redundant in this case:

```
read -p "Type your first name and last name:" NAME SURNAME
```

Scripts performing system tasks will often require information provided by other programs. The *backtick notation* can be used to store the output of a command in a variable:

```
$ OS=`uname -o`
```

In the example, the output of command `uname -o` will be stored in the `OS` variable. An identical result will be produced with `$()`:

```
$ OS=$(uname -o)
```

The length of a variable, that is, the quantity of characters it contains, is returned by prepending a hash `#` before the name of the variable. This feature, however, requires the use of the curly braces syntax to indicate the variable:

```
$ OS=$(uname -o)
$ echo $OS
GNU/Linux
$ echo ${#OS}
9
```

Bash also features one-dimensional array variables, so a set of related elements can be stored with a single variable name. Every element in an array has a numerical index, which must be used to write and read values in the corresponding element. Different from ordinary variables, arrays must be declared with the Bash builtin command `declare`. For example, to declare a variable named `SIZES` as an array:

```
$ declare -a SIZES
```

Arrays can also be implicitly declared when populated from a predefined list of items, using the parenthesis notation:

```
$ SIZES=( 1048576 1073741824 )
```

In the example, the two large integer values were stored in the `SIZES` array. Array elements must be referenced using curly braces and square brackets, otherwise Bash will not change or display the element correctly. As array indexes start at 0, the content of the first element is in `${SIZES[0]}`, the second element is in `${SIZES[1]}`, and so on:

```
$ echo ${SIZES[0]}
1048576
$ echo ${SIZES[1]}
1073741824
```

Unlike reading, changing an array element's content is performed without the curly braces (e.g., `SIZES[0]=1048576`). As with ordinary variables, the length of an element in an array is returned with the hash character (e.g., `${#SIZES[0]}` for the length of the first element in `SIZES` array). The total number of elements in an array is returned if `@` or `*` are used as the index:

```
$ echo ${#SIZES[@]}
2
$ echo ${#SIZES[*]}
2
```

Arrays can also be declared using the output of a command as the initial elements through command substitution. The following example shows how to create a Bash array whose elements are the current system's supported filesystems:

```
$ FS=( $(cut -f 2 < /proc/filesystems) )
```

The command `cut -f 2 < /proc/filesystems` will display all the filesystems currently supported by the running kernel (as listed in the second column of the file `/proc/filesystems`), so the array `FS` now contains one element for each supported filesystem. Any text content can be used to initialize an array as, by default, any terms delimited by *space*, *tab* or *newline* characters will become an array element.

**TIP** Bash treats each character of an environment variable's `$IFS` (*Input Field Separator*) as a delimiter. To change the field delimiter to newline characters only, for example, the `IFS` variable should be reset with command `IFS=$'\n'`.

## Arithmetic Expressions

Bash provides a practical method to perform integer arithmetic operations with the builtin command `expr`. Two numerical variables, `$VAL1` and `$VAL2` for example, can be added together with the following command:

```
$ SUM=`expr $VAL1 + $VAL2`
```

The resulting value of the example will be available in the `$SUM` variable. Command `expr` can be replaced by `$(( ))`, so the previous example can be rewritten as `SUM=$(( $VAL1 + $VAL2 ))`. Power expressions are also allowed with the double asterisks operator, so the previous array declaration `SIZES=( 1048576 1073741824 )` could be rewritten as `SIZES=( $( (1024**2) ) $( (1024**3) ) )`.

Command substitution can also be used in arithmetic expressions. For example, the file `/proc/meminfo` has detailed information about the system memory, including the number of free bytes in RAM:

```
$ FREE=$(( 1000 * `sed -nre '2s/[^[:digit:]]//gp' < /proc/meminfo` ))
```

The example shows how the command `sed` can be used to parse the contents of `/proc/meminfo` inside the arithmetic expression. The second line of the `/proc/meminfo` file contains the amount of free memory in thousands of bytes, so the arithmetic expression multiplies it by 1000 to obtain the number of free bytes in RAM.

## Conditional Execution

Some scripts usually are not intended to execute all the commands in the script file, but just those commands that match a predefined criteria. For example, a maintenance script may send a warning message to the administrator's email only if the execution of a command fails. Bash provides specific methods of assessing the success of command execution and general conditional structures, more similar to those found in popular programming languages.

By separating commands with `&&`, the command to the right will be executed only if the command to the left did not encounter an error, that is, if its exit status was equal to `0`:

```
COMMAND A && COMMAND B && COMMAND C
```

The opposite behaviour occurs if commands are separated with `||`. In this case, the following command will be executed only if the previous command did encounter an error, that is, if its returning status code differs from `0`.

One of the most important features of all programming languages is the ability to execute commands depending on previously defined conditions. The most straightforward way to conditionally execute commands is to use the Bash builtin command `if`, which executes one or more commands only if the command given as argument returns a `0` (success) status code. Another command, `test`, can be used to assess many different special criteria, so it is mostly used in conjunction with `if`. In the following example, the message `Confirmed: /bin/bash is executable.` will be displayed if the file `/bin/bash` exists and it is executable:

```
if test -x /bin/bash ; then
  echo "Confirmed: /bin/bash is executable."
fi
```



Option `-x` makes command `test` return a status code 0 only if the given path is an executable file. The following example shows another way to achieve the exact same result, as the square brackets can be used as a replacement for `test`:

```
if [ -x /bin/bash ] ; then
    echo "Confirmed: /bin/bash is executable."
fi
```

The `else` instruction is optional to the `if` structure and can, if present, define a command or sequence of commands to execute if the conditional expression is not true:

```
if [ -x /bin/bash ] ; then
    echo "Confirmed: /bin/bash is executable."
else
    echo "No, /bin/bash is not executable."
fi
```

`if` structures must always end with `fi`, so the Bash interpreter knows where the conditional commands end.

## Script Output

Even when the purpose of a script only involves file-oriented operations, it is important to display progress related messages in the standard output, so the user is kept informed of any issues and can eventually use those messages to generate operation logs.

The Bash builtin command `echo` is commonly used to display simple strings of text, but it also provides some extended features. With option `-e`, command `echo` is able to display special characters using escaped sequences (a backslash sequence designating a special character). For example:

```
#!/bin/bash

# Get the operating system's generic name
OS=$(uname -o)

# Get the amount of free memory in bytes
FREE=$(( 1000 * `sed -nre '2s/^[[:digit:]]*//gp' < /proc/meminfo` ))

echo -e "Operating system:\t$OS"
```

```
echo -e "Unallocated RAM:\t$(( $FREE / 1024**2 )) MB"
```

Whilst the use of quotes is optional when using `echo` with no options, it is necessary to add them when using option `-e`, otherwise the special characters may not render correctly. In the previous script, both `echo` commands use the tabulation character `\t` to align the text, resulting in the following output:

```
Operating system:      GNU/Linux
Unallocated RAM:      1491 MB
```

The newline character `\n` can be used to separate the output lines, so the exact same output is obtained by combining the two `echo` commands into only one:

```
echo -e "Operating system:\t$OS\nUnallocated RAM:\t$(( $FREE / 1024**2 )) MB"
```

Although fit to display most text messages, command `echo` may not be well suited to display more specific text patterns. Bash builtin command `printf` gives more control over how to display the variables. Command `printf` uses the first argument as the format of the output, where placeholders will be replaced by the following arguments in the order they appear in the command line. For example, the message of the previous example could be generated with the following `printf` command:

```
printf "Operating system:\t%s\nUnallocated RAM:\t%d MB\n" $OS $(( $FREE / 1024**2 ))
```

The placeholder `%s` is intended for text content (it will be replaced by the `$OS` variable) and the `%d` placeholder is intended to integer numbers (it will be replaced by the resulting number of free megabytes in RAM). `printf` does not append a newline character at the end of the text, so the newline character `\n` should be placed at the end of the pattern if needed. The entire pattern should be interpreted as a single argument, so it must be enclosed in quotes.

#### TIP

The format of placeholder substitution performed by `printf` can be customized using the same format used by the `printf` function from the C programming language. The full reference for the `printf` function can be found in its manual page, accessed with the command `man 3 printf`.

With `printf`, the variables are placed outside the text pattern, which makes it possible to store the text pattern in a separate variable:

```
MSG='Operating system:\t%s\nUnallocated RAM:\t%d MB\n'
```

```
printf "$MSG" $OS $(( $FREE / 1024**2 ))
```

This method is particularly useful to display distinct output formats, depending on the user's requirements. It makes it easier, for example, to write a script that uses a distinct text pattern if the user requires a CSV (*Comma Separated Values*) list rather than a default output message.

## Guided Exercises

1. The `-s` option for the `read` command is useful for entering passwords, as it will not show the content being typed on the screen. How could the `read` command be used to store the user's input in the variable `PASSWORD` while hiding the typed content?

2. The only purpose of the command `whoami` is to display the username of the user who called it, so it is mostly used inside scripts to identify the user who is running it. Inside a Bash script, how could the output of the `whoami` command be stored in the variable named `WHO`?

3. What Bash operator should be between the commands `apt-get dist-upgrade` and `systemctl reboot` if the root user wants to execute `systemctl reboot` only if `apt-get dist-upgrade` finished successfully?

## Explorational Exercises

1. After trying to run a newly created Bash script, a user receives the following error message:

```
bash: ./script.sh: Permission denied
```

Considering that the file `./script.sh` was created by the same user, what would be the probable cause of this error?

2. Suppose a script file named `do.sh` is executable and the symbolic link named `undo.sh` points to it. From within the script, how could you identify if the calling filename was `do.sh` or `undo.sh`?

3. In a system with a properly configured email service, the command `mail -s "Maintenance Error" root <<<"Scheduled task error"` sends the notice email message to the root user. Such a command could be used in unattended tasks, like *cronjobs*, to inform the system administrator about an unexpected issue. Write an *if* construct that will execute the aforementioned `mail` command if the exit status of the previous command—whatever it was—is unsuccessful.

## Summary

This lesson covers the basic concepts for understanding and writing Bash shell scripts. Shell scripts are a core part of any Linux distribution as they offer a very flexible way to automate user and system tasks performed in the shell environment. The lesson goes through the following steps:

- Shell script structure and correct script file permissions
- Script parameters
- Using variables to read user input and to store the output of commands
- Bash arrays
- Simple tests and conditional execution
- Output formatting

The commands and procedures addressed were:

- Bash builtin notation for command substitution, array expansion and arithmetic expressions
- Conditional command execution with the `||` and `&&` operators
- `echo`
- `chmod`
- `exec`
- `read`
- `declare`
- `test`
- `if`
- `printf`

## Answers to Guided Exercises

1. The `-s` option for the `read` command is useful for entering passwords, as it will not show the content being typed on the screen. How could the `read` command be used to store the user's input in the variable `PASSWORD` while hiding the typed content?

```
read -s PASSWORD
```

2. The only purpose of the command `whoami` is to display the username of the user who called it, so it is mostly used inside scripts to identify the user who is running it. Inside a Bash script, how could the output of the `whoami` command be stored in the variable named `WHO`?

```
WHO=`whoami` or WHO=$(whoami)
```

3. What Bash operator should be between the commands `apt-get dist-upgrade` and `systemctl reboot` if the root user wants to execute `systemctl reboot` only if `apt-get dist-upgrade` finished successfully?

The operator `&&`, as in `apt-get dist-upgrade && systemctl reboot`.

# Answers to Explorational Exercises

1. After trying to run a newly created Bash script, a user receives the following error message:

```
bash: ./script.sh: Permission denied
```

Considering that the file `./script.sh` was created by the same user, what would be the probable cause of this error?

The `./script.sh` file does not have the execution permission enabled.

2. Suppose a script file named `do.sh` is executable and the symbolic link named `undo.sh` points to it. From within the script, how could you identify if the calling filename was `do.sh` or `undo.sh`?

The special variable `$0` contains the filename used to call the script.

3. In a system with a properly configured email service, the command `mail -s "Maintenance Error" root <<<"Scheduled task error"` sends the notice email message to the root user. Such a command could be used in unattended tasks, like *cronjobs*, to inform the system administrator about an unexpected issue. Write an *if* construct that will execute the aforementioned `mail` command if the exit status of the previous command—whatever it was—is unsuccessful.

```
if [ "$?" -ne 0 ]; then mail -s "Maintenance Error" root <<<"Scheduled task error"; fi
```





## 105.2 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	105 Shells and Shell Scripting
<b>Objective:</b>	105.2 Customize or write simple scripts
<b>Lesson:</b>	2 of 2

### Introduction

Shell scripts are generally intended to automate operations related to files and directories, the same operations that could be performed manually at the command line. Yet, the coverage of shell scripts are not only restricted to a user's documents, as the configuration and interaction with many aspects of a Linux operating system are also accomplished through script files.

The Bash shell offers many helpful builtin commands to write shell scripts, but the full power of these scripts relies on the combination of Bash builtin commands with the many command line utilities available on a Linux system.

### Extended Tests

Bash as a script language is mostly oriented to work with files, so the Bash builtin command `test` has many options to evaluate the properties of filesystem objects (essentially files and directories). Tests that are focused on files and directories are useful, for example, to verify if the files and directories required to perform a particular task are present and can be read. Then, associated with an *if* conditional construct, the proper set of actions are executed if the test is successful.

The `test` command can evaluate expressions using two different syntaxes: test expressions can be given as an argument to command `test` or they can be placed inside square brackets, where command `test` is given implicitly. Thus, the test to evaluate if `/etc` is a valid directory can be written as `test -d /etc` or as `[ -d /etc ]`:

```
$ test -d /etc
$ echo $?
0
$ [ -d /etc ]
$ echo $?
0
```

As confirmed by the exit status codes in the special variable `?` — a value of 0 means the test was successful — both forms evaluated `/etc` as a valid directory. Assuming the path to a file or directory was stored in the variable `$VAR`, the following expressions can be used as arguments to `test` or inside square brackets:

**-a "\$VAR"**

Evaluate if the path in `VAR` exists in the filesystem and it is a file.

**-b "\$VAR"**

Evaluate if the path in `VAR` is a special block file.

**-c "\$VAR"**

Evaluate if the path in `VAR` is a special character file.

**-d "\$VAR"**

Evaluate if the path in `VAR` is a directory.

**-e "\$VAR"**

Evaluate if the path in `VAR` exists in the filesystem.

**-f "\$VAR"**

Evaluate if the path in `VAR` exists and it is a regular file.

**-g "\$VAR"**

Evaluate if the path in `VAR` has the SGID permission.

**-h "\$VAR"**

Evaluate if the path in `VAR` is a symbolic link.

**-L "\$VAR"**

Evaluate if the path in VAR is a symbolic link (like -h).

**-k "\$VAR"**

Evaluate if the path in VAR has the *sticky* bit permission.

**-p "\$VAR"**

Evaluate if the path in VAR is a *pipe* file.

**-r "\$VAR"**

Evaluate if the path in VAR is readable by the current user.

**-s "\$VAR"**

Evaluate if the path in VAR exists and it is not empty.

**-S "\$VAR"**

Evaluate if the path in VAR is a socket file.

**-t "\$VAR"**

Evaluate if the path in VAR is open in a terminal.

**-u "\$VAR"**

Evaluate if the path in VAR has the SUID permission.

**-w "\$VAR"**

Evaluate if the path in VAR is writable by the current user.

**-x "\$VAR"**

Evaluate if the path in VAR is executable by the current user.

**-O "\$VAR"**

Evaluate if the path in VAR is owned by the current user.

**-G "\$VAR"**

Evaluate if the path in VAR belongs to the effective group of the current user.

**-N "\$VAR"**

Evaluate if the path in VAR has been modified since the last time it was accessed.

**"\$VAR1" -nt "\$VAR2"**

Evaluate if the path in VAR1 is newer than the path in VAR2, according to their modification dates.

**"\$VAR1" -ot "\$VAR2"**

Evaluate if the path in VAR1 is older than VAR2.

**"\$VAR1" -ef "\$VAR2"**

This expression evaluates to True if the path in VAR1 is a hardlink to VAR2.

It is recommended to use the double quotes around a tested variable because, if the variable happens to be empty, it could cause a syntax error for the test command. The test options require an operand argument and an unquoted empty variable would cause an error due to a missing required argument. There are also tests for arbitrary text variables, described as follows:

**-z "\$TXT"**

Evaluate if variable TXT is empty (zero size).

**-n "\$TXT" or test "\$TXT"**

Evaluate if variable TXT is not empty.

**"\$TXT1" = "\$TXT2" or "\$TXT1" == "\$TXT2"**

Evaluate if TXT1 and TXT2 are equal.

**"\$TXT1" != "\$TXT2"**

Evaluate if TXT1 and TXT2 are not equal.

**"\$TXT1" < "\$TXT2"**

Evaluate if TXT1 comes before TXT2, in alphabetical order.

**"\$TXT1" > "\$TXT2"**

Evaluate if TXT1 comes after TXT2, in alphabetical order.

Distinct languages may have different rules for alphabetical ordering. To obtain consistent results, regardless of the localization settings of the system where the script is being executed, it is recommended to set the environment variable LANG to C, as in LANG=C, before doing operations involving alphabetical ordering. This definition will also keep the system messages in the original language, so it should be used only within script scope.

Numerical comparisons have their own set of test options:

**\$NUM1 -lt \$NUM2**

Evaluate if NUM1 is less than NUM2.

**\$NUM1 -gt \$NUM2**

Evaluate if NUM1 is greater than NUM2.

**\$NUM1 -le \$NUM2**

Evaluate if NUM1 is less or equal to NUM2.

**\$NUM1 -ge \$NUM2**

Evaluate if NUM1 is greater or equal to NUM2.

**\$NUM1 -eq \$NUM2**

Evaluate if NUM1 is equal to NUM2.

**\$NUM1 -ne \$NUM2**

Evaluate if NUM1 is not equal to NUM2.

All tests can receive the following modifiers:

**! EXPR**

Evaluate if the expression EXPR is false.

**EXPR1 -a EXPR2**

Evaluate if both EXPR1 and EXPR2 are true.

**EXPR1 -o EXPR2**

Evaluate if at least one of the two expressions are true.

Another conditional construct, `case`, can be seen as a variation of the `if` construct. The instruction `case` will execute a list of given commands if a specified item—the content of a variable, for example—can be found in a list of items separated by *pipes* (the vertical bar `|`) and terminated by `)`. The following sample script shows how the `case` construct can be used to indicate the corresponding software packaging format for a given Linux distribution:

```
#!/bin/bash

DISTRO=$1

echo -n "Distribution $DISTRO uses "
case "$DISTRO" in
```

```

    debian | ubuntu | mint)
    echo -n "the DEB"
;;
    centos | fedora | opensuse )
    echo -n "the RPM"
;;
    *)
    echo -n "an unknown"
;;
esac
echo " package format."

```

Each list of patterns and associated commands must be terminated with `;;`, `;&`, or `;;&`. The last pattern, an asterisk, will match if no other previous pattern corresponded beforehand. The `esac` instruction (*case* backwards) terminates the `case` construct. Assuming the previous sample script was named `script.sh` and it is executed with `opensuse` as the first argument, the following output will be generated:

```

$ ./script.sh opensuse
Distribution opensuse uses the RPM package format.

```

**TIP**

Bash has an option called `nocasematch` that enables case-insensitive pattern matching for the `case` construct and other conditional commands. The `shopt` builtin command toggles the values of settings controlling optional shell behaviour: `shopt -s` will enable (*set*) the given option and `shopt -u` will disable (*unset*) the given option. Therefore, placing `shopt -s nocasematch` before the `case` construct will enable case-insensitive pattern matching. Options modified by `shopt` will only affect the current session, so modified options inside scripts running in a sub-shell — which is the standard way to run a script — do not affect the options of the parent session.

The searched item and the patterns undergo tilde expansion, parameter expansion, command substitution and arithmetic expansion. If the searched item is specified with quotes, they will be removed before matching is attempted.

## Loop Constructs

Scripts are often used as a tool to automate repetitive tasks, performing the same set of commands until a stop criteria is verified. Bash has three loop instructions — `for`, `until` and `while` — designed for slightly distinct loop constructions.

The `for` construct walks through a given list of items — usually a list of words or any other space-

separated text segments — executing the same set of commands on each one of those items. Before each iteration, the `for` instruction assigns the current item to a variable, which can then be used by the enclosed commands. The process is repeated until there are no more items left. The syntax of the `for` construct is:

```
for VARNAME in LIST
do
    COMMANDS
done
```

`VARNAME` is an arbitrary shell variable name and `LIST` is any sequence of separated terms. The valid delimiting characters splitting items in the list are defined by the `IFS` environment variable, which are the characters *space*, *tab* and *newline* by default. The list of commands to be executed is delimited by the `do` and `done` instructions, so commands can occupy as much lines as needed.

In the following example, the `for` command will take each item from the provided list—a sequence of numbers—and assign it to the `NUM` variable, one item at a time:

```
#!/bin/bash

for NUM in 1 1 2 3 5 8 13
do
    echo -n "$NUM is "
    if [ $(( $NUM % 2 )) -ne 0 ]
    then
        echo "odd."
    else
        echo "even."
    fi
done
```

In the example, a nested `if` construct is used in conjunction with an arithmetic expression to evaluate if the number in the current `NUM` variable is even or odd. Assuming the previous sample script was named `script.sh` and it is in the current directory, the following output will be generated:

```
$ ./script.sh
1 is odd.
1 is odd.
2 is even.
3 is odd.
```

```
5 is odd.  
8 is even.  
13 is odd.
```

Bash also supports an alternative format to `for` constructs, with the double parenthesis notation. This notation resembles the `for` instruction syntax from the C programming language and it is particularly useful for working with arrays:

```
#!/bin/bash  
  
SEQ=( 1 1 2 3 5 8 13 )  
  
for (( IDX = 0; IDX < ${#SEQ[*]}; IDX++ ))  
do  
    echo -n "${SEQ[$IDX]} is "  
    if [ $(( ${SEQ[$IDX]} % 2 )) -ne 0 ]  
    then  
        echo "odd."  
    else  
        echo "even."  
    fi  
done
```

This sample script will generate the exact same output as the previous example. However, instead of using the `NUM` variable to store one item at a time, the `IDX` variable is employed to track the current array index in ascending order, starting from 0 and continuously adding to it while it is under the number of items in the `SEQ` array. The actual item is retrieved from its array position with `${SEQ[$IDX]}`.

In the same fashion, the `until` construct executes a command sequence until a test command—like the `test` command itself—terminates with status 0 (success). For example, the same loop structure from the previous example can be implemented with `until` as follows:

```
#!/bin/bash  
  
SEQ=( 1 1 2 3 5 8 13 )  
  
IDX=0  
  
until [ $IDX -eq ${#SEQ[*]} ]  
do
```



```

echo -n "${SEQ[$IDX]} is "
if [ $(( ${SEQ[$IDX]} % 2 )) -ne 0 ]
then
    echo "odd."
else
    echo "even."
fi
IDX=$(( $IDX + 1 ))
done

```

`until` constructs may require more instructions than `for` constructs, but it can be more suitable to non-numerical stop criteria provided by `test` expressions or any other command. It is important to include actions that ensure a valid stop criteria, like the increment of a counter variable, otherwise the loop may run indefinitely.

The instruction `while` is similar to the instruction `until`, but `while` keeps repeating the set of commands if the test command terminates with status 0 (success). Therefore, the instruction `until [ $IDX -eq ${#SEQ[*]} ]` from the previous example is equivalent to `while [ $IDX -lt ${#SEQ[*]} ]`, as the loop should repeat while the array index is *less than* the total of items in the array.

## A More Elaborate Example

Imagine a user wants to periodically sync a collection of their files and directories with another storage device, mounted at an arbitrary mount point in the filesystem, and a full featured backup system is considered overkill. Since this is an activity intended to be performed periodically, it is a good application candidate for automating with a shell script.

The task is straight forward: sync every file and directory contained in a list, from an origin directory informed as the first script argument to a destination directory informed as the second script argument. To make it easier to add or remove items from the list, it will be kept in a separated file, `~/ .sync .list`, one item per line:

```

$ cat ~/.sync.list
Documents
To do
Work
Family Album
.config
.ssh
.bash_profile

```

```
.vimrc
```

The file contains a mixture of files and directories, some with blank spaces in their names. This is a suitable scenario for the builtin Bash command `mapfile`, which will parse any given text content and create an array variable from it, placing each line as an individual array item. The script file will be named `sync.sh`, containing the following script:

```
#!/bin/bash

set -ef

# List of items to sync
FILE=~/.sync.list

# Origin directory
FROM=$1

# Destination directory
TO=$2

# Check if both directories are valid
if [ ! -d "$FROM" -o ! -d "$TO" ]
then
    echo Usage:
    echo "$0 <SOURCEDIR> <DESTDIR>"
    exit 1
fi

# Create array from file
mapfile -t LIST < $FILE

# Sync items
for (( IDX = 0; IDX < ${#LIST[*]}; IDX++ ))
do
    echo -e "$FROM/${LIST[$IDX]} \u2192 $TO/${LIST[$IDX]}";
    rsync -qa --delete "$FROM/${LIST[$IDX]}" "$TO";
done
```

The first action the script does is to redefine two shell parameters with command `set`: option `-e` will exit execution immediately if a command exits with a non-zero status and option `-f` will disable file name globbing. Both options can be shortened as `-ef`. This is not a mandatory step, but helps diminish the probability of unexpected behaviour.

The actual application oriented instructions of the script file can be divided in three parts:

### 1. *Collect and check script parameters*

The `FILE` variable is the path to the file containing the list of items to be copied: `~/sync.list`. Variables `FROM` and `TO` are the origin and destination paths, respectively. Since these last two parameters are user provided, they go through a simple validation test performed by the `if` construct: if any of the two is not a valid directory — assessed by the test `[ ! -d "$FROM" -o ! -d "$TO" ]` — the script will show a brief help message and then terminate with an exit status of 1.

### 2. *Load list of files and directories*

After all the parameters are defined, an array containing the list of items to be copied is created with the command `mapfile -t LIST < $FILE`. Option `-t` of `mapfile` will remove the trailing newline character from each line before including it in the array variable named `LIST`. The contents of the file indicated by the variable `FILE` — `~/sync.list` — is read via input redirection.

### 3. *Perform the copy and inform the user*

A `for` loop using double parenthesis notation traverses the array of items, with the `IDX` variable keeping track of the index increment. The `echo` command will inform the user of each item being copied. The escaped unicode character — `\u2192` — for the *right arrow* character is present in the output message, so the `-e` option of command `echo` must be used. Command `rsync` will selectively copy only the modified file pieces from the origin, thus its use is recommended for such tasks. `rsync` options `-q` and `-a`, condensed in `-qa`, will inhibit `rsync` messages and activate the *archive* mode, where all file properties are preserved. Option `--delete` will make `rsync` delete an item in the destination that does not exist in the origin anymore, so it should be used with care.

Assuming all the items in the list exist in the home directory of user `carol`, `/home/carol`, and the destination directory `/media/carol/backup` points to a mounted external storage device, the command `sync.sh /home/carol /media/carol/backup` will generate the following output:

```
$ sync.sh /home/carol /media/carol/backup
/home/carol/Documents → /media/carol/backup/Documents
/home/carol/"To do" → /media/carol/backup/"To do"
/home/carol/Work → /media/carol/backup/Work
/home/carol/"Family Album" → /media/carol/backup/"Family Album"
/home/carol/.config → /media/carol/backup/.config
/home/carol/.ssh → /media/carol/backup/.ssh
```

```
/home/carol/.bash_profile → /media/carol/backup/.bash_profile  
/home/carol/.vimrc → /media/carol/backup/.vimrc
```

The example also assumes the script is executed by root or by user `carol`, as most of the files would be unreadable by other users. If `script.sh` is not inside a directory listed in the `PATH` environment variable, then it should be specified with its full path.

## Guided Exercises

1. How could command `test` be used to verify if the file path stored in the variable `FROM` is newer than a file whose path is stored in the variable `T0`?

2. The following script should print a number sequence from 0 to 9, but instead it indefinitely prints 0. What should be done in order to get the expected output?

```
#!/bin/bash

COUNTER=0

while [ $COUNTER -lt 10 ]
do
    echo $COUNTER
done
```

3. Suppose a user wrote a script that requires a sorted list of usernames. The resulting sorted list is presented as the following on his computer:

```
carol
Dave
emma
Frank
Grace
henry
```

However, the same list is sorted as the following on his colleague's computer:

```
Dave
Frank
Grace
carol
emma
henry
```

What could explain the differences between the two sorted lists?



## Explorational Exercises

1. How could all of the script's command line arguments be used to initialize a Bash array?

2. Why is it that, counter intuitively, the command `test 1 > 2` evaluates as true?

3. How would a user temporarily change the default field separator to the newline character only, while still being able to revert it to its original content?

## Summary

This lesson takes a deeper look at the tests available for the `test` command and at other conditional and loop constructs, necessary to write more elaborate shell scripts. A simple file synchronization script is given as an example of a practical shell script application. The lesson goes through the following steps:

- Extended tests for the `if` and `case` conditional constructs.
- Shell loop constructs: `for`, `until` and `while`.
- Iterating through arrays and parameters.

The commands and procedures addressed were:

### **test**

Perform a comparison between items supplied to the command.

### **if**

A logic construct used in scripts to evaluate something as either true or false, then branch command execution based on results.

### **case**

Evaluate several values against a single variable. Script command execution is then carried out depending on the `case` command's result.

### **for**

Repeat execution of a command based on a given criteria.

### **until**

Repeat the execution of a command until an expression evaluates to false.

### **while**

Repeat the execution of a command while a given expression evaluates to true.



## Answers to Guided Exercises

1. How could command `test` be used to verify if the file path stored in the variable `FROM` is newer than a file whose path is stored in the variable `T0`?

The command `test "$FROM" -nt "$T0"` will return a status code of 0 if the file in the `FROM` variable is newer than the file in the `T0` variable.

2. The following script should print a number sequence from 0 to 9, but instead it indefinitely prints 0. What should be done in order to get the expected output?

```
#!/bin/bash

COUNTER=0

while [ $COUNTER -lt 10 ]
do
    echo $COUNTER
done
```

The `COUNTER` variable should be incremented, which could be done with the arithmetic expression `COUNTER=$(( $COUNTER + 1 ))`, to eventually reach the stop criteria and end the loop.

3. Suppose a user wrote a script that requires a sorted list of usernames. The resulting sorted list is presented as the following in his computer:

```
carol
Dave
emma
Frank
Grace
henry
```

However, the same list is sorted as the following in his colleague computer:

```
Dave
Frank
Grace
carol
emma
```

henry

What could explain the differences between the two sorted lists?

The sorting is based on the current system's locale. To avoid the inconsistencies, sorting tasks should be performed with the LANG environment variable set to C.

## Answers to Explorational Exercises

1. How could all of the script's command line arguments be used to initialize a Bash array?

The commands `PARAMS=( $* )` or `PARAMS=( "$@" )` will create an array called `PARAMS` with all the arguments.

2. Why is it that, counter intuitively, the command `test 1 > 2` evaluates as true?

Operator `>` is intended to be used with string tests, no numerical tests.

3. How would a user temporarily change the default field separator to the newline character only, while still being able to revert it to its original content?

A copy of the `IFS` variable can be stored in another variable: `OLDIFS=$IFS`. Then the new line separator is defined with `IFS=$'\n'` and the `IFS` variable can be reverted back with `IFS=$OLDIFS`.



## **Topic 106: User Interfaces and Desktops**



## 106.1 Install and configure X11

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 106.1](#)

### Weight

2

### Key knowledge areas

- Understanding of the X11 architecture.
- Basic understanding and knowledge of the X Window configuration file.
- Overwrite specific aspects of Xorg configuration, such as keyboard layout.
- Understand the components of desktop environments, such as display managers and window managers.
- Manage access to the X server and display applications on remote X servers.
- Awareness of Wayland.

### Partial list of the used files, terms and utilities

- `/etc/X11/xorg.conf`
- `/etc/X11/xorg.conf.d/`
- `~/.xsession-errors`
- `xhost`
- `xauth`
- `DISPLAY`
- `X`



# 106.1 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	106 User Interfaces and Desktops
<b>Objective:</b>	106.1 Install and configure X11
<b>Lesson:</b>	1 of 1

## Introduction

The X Window System is a software stack that is used to display text and graphics on a screen. The overall look and design of an X client is not dictated by the X Window System, but is instead handled by each individual X client, a *window manager* (e.g. Window Maker, Tab Window Manager), or a complete *desktop environment* such as KDE, GNOME, or Xfce. Desktop environments will be covered in a later lesson. This lesson will focus on the underlying architecture and common tools for the X Window System that an administrator would use to configure X.

The X Window System is cross-platform and runs on various operating systems such as Linux, the BSDs, Solaris and other Unix-like systems. There are also implementations available for Apple's macOS and Microsoft Windows.

The primary version of the X protocol used in modern Linux distributions is *X.org* version 11, commonly written as *X11*. The X protocol is the communication mechanism between the X client and X server. The differences between the X client and X server will be discussed below.

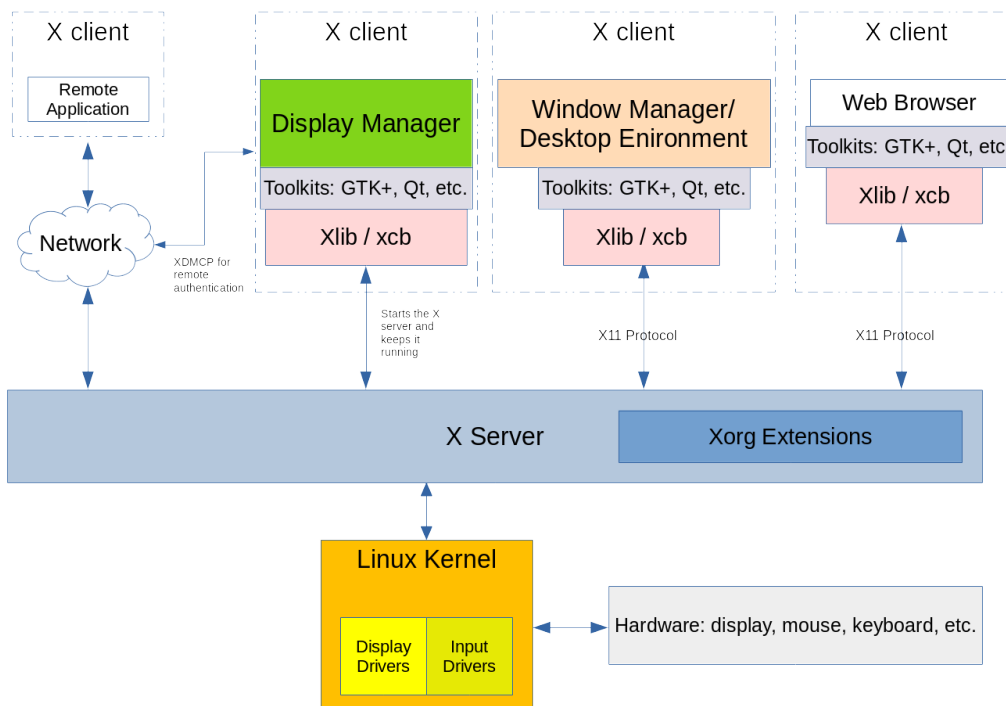
**NOTE** The X Window System's predecessor was a window system called *W* and was a

joint development effort between IBM, DEC and MIT. This software was born out of *Project Athena* in 1984. When the developers started work on a new display server, they chose the next letter in the English alphabet: “X”. The X Window System’s evolution is currently controlled by the *MIT X Consortium*.

## X Window System Architecture

The X Window System provides the mechanisms for drawing basic two dimensional shapes (and three dimensional shapes through extensions) on a display. It is divided into a client and a server, and in most installations where a graphical desktop is needed both of these components are on the same computer. The client component takes the form of an application, such as a terminal emulator, a game or a web browser. Each client application informs the X server about its window location and size on a computer screen. The client also handles what goes into that window, and the X server puts the requested drawing up on the screen. The X Window System also handles input from devices such as mice, keyboards, trackpads and more.

### Basic Structure of an X Window System



The X Window System is network-capable and multiple X clients from different computers on a network can make drawing requests to a single remote X server. The reasoning behind this is so that an administrator or user can have access to a graphical application on a remote system that

may not be available on their local system.

A key feature of the X Window System is that it is modular. Over the course of the X Window System's existence newer features have been developed and added to its framework. These new components were only added as extensions to the X server, leaving the core X11 protocol intact. These extensions are contained within *Xorg* library files. Examples of Xorg libraries include: `libXrandr`, `libXcursor`, `libX11`, `libxkbfile` as well as several others, each providing extended functionality to the X server.

A *display manager* provides a graphical login to a system. This system can either be a local computer or a computer across a network. The display manager is launched after the computer boots and will start an X server session for the authenticated user. The display manager is also responsible for keeping the X server up and running. Example display managers include GDM, SDDM and LightDM.

Each instance of a running X server has a *display name* to identify it. The display name contains the following:

```
hostname:displaynumber.screennumber
```

The display name also instructs a graphical application where it should be rendered and on which host (if using a remote X connection).

The `hostname` refers to the name of the system that will display the application. If a `hostname` is missing from the display name, then the local host is assumed.

The `displaynumber` references the collection of “screens” that are in use, whether that is a single laptop screen or multiple screens on a workstation. Each running X server session is given a display number starting at 0.

The default `screennumber` is 0. This can be the case if only one physical screen or multiple physical screens are configured to work as one screen. When all screens in a multi-monitor setup are combined into one logical screen, application windows can be moved freely between the screens. In situations where each screen is configured to work independently of one another, each screen will house the application windows that open within them and the windows can not be moved from one screen to another. Each independent screen will have its own number assigned to it. If there is only one logical screen in use, then the dot and the screen number are omitted.

The display name of a running X session is stored in the `DISPLAY` environment variable:

```
$ echo $DISPLAY
```



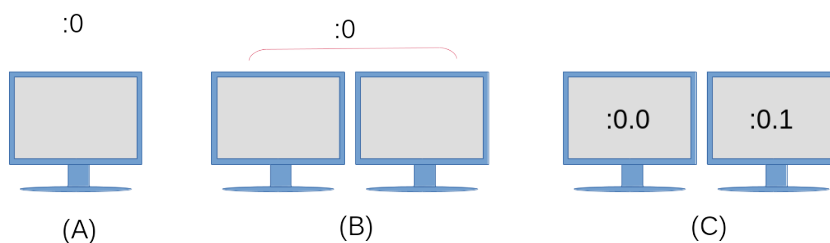
```
:0
```

The output details the following:

1. The X server in use is on the local system, hence there is nothing printed to the left of the colon.
2. The current X server session is the first as indicated by the `0` immediately following the colon.
3. There is only one logical screen in use, so a screen number is not visible.

To further illustrate this concept, refer to the following diagram:

### Example Display Configurations



#### (A)

A single monitor, with a single display configuration and only one screen.

#### (B)

Configured as a single display, with two physical monitors configured as one screen. Application windows can be moved freely between the two monitors.

#### (C)

A single display configuration (as indicated by the `:0`) however each monitor is an independent screen. Both screens will still share the same input devices such as a keyboard and mouse, however an application opened on screen `:0.0` can not be moved to screen `:0.1` and vice versa.

To start an application on a specific screen, assign the screen number to the `DISPLAY` environment variable prior to launching the application:

```
$ DISPLAY=:0.1 firefox &
```

This command would start the Firefox web browser on the screen to the right in the diagram above. Some toolkits also provide command line options to instruct an application to run on a specified screen. See `--screen` and `--display` in the `gtk-options(7)` man page for an example.

## X Server Configuration

Traditionally, the primary configuration file that is used to configure an X server is the `/etc/X11/xorg.conf` file. On modern Linux distributions, the X server will configure itself at runtime when the X server is started and so no `xorg.conf` file may exist.

The `xorg.conf` file is divided up into separate stanzas called *sections*. Each section begins with the term `Section` and following this term is the *section name* which refers to a component's configuration. Each `Section` is terminated by a corresponding `EndSection`. The typical `xorg.conf` file contains the following sections:

### InputDevice

Used to configure a specific model of keyboard or mouse.

### InputClass

In modern Linux distributions this section is typically found in a separate configuration file located under `/etc/X11/xorg.conf.d/`. The `InputClass` is used to configure a *class* of hardware devices such as keyboards and mice rather than a specific piece of hardware. Below is an example `/etc/X11/xorg.conf.d/00-keyboard.conf` file:

```
Section "InputClass"
    Identifier "system-keyboard"
    MatchIsKeyboard "on"
    Option "XkbLayout" "us"
    Option "XkbModel" "pc105"
EndSection
```

The option for `XkbLayout` determines the layout of the keys on a keyboard, such as Dvorak, left or right handed, QWERTY and language. The option for `XkbModel` is used to define the type of keyboard in use. A table of models, layouts and their descriptions can be found within `xkeyboard-config(7)`. The files associated with keyboard layouts can be found within `/usr/share/X11/xkb`. An example Greek Polytonic keyboard layout on a Chromebook computer would look like the following:

```
Section "InputClass"
    Identifier "system-keyboard"
    MatchIsKeyboard "on"
    Option "XkbLayout" "gr(polytonic)"
    Option "XkbModel" "chromebook"
EndSection
```

Alternatively, a keyboard's layout can be modified during a running X session with the `setxkbmap` command. Here is an example of this command setting up the Greek Polytonic layout on a Chromebook computer:

```
$ setxkbmap -model chromebook -layout "gr(polytonic)"
```

This setting will only last as long as the X session is in use. To make such changes permanent, modify the `/etc/X11/xorg.conf.d/00-keyboard.conf` file to include the settings required.

**NOTE** The `setxkbmap` command makes use of the *X Keyboard Extension* (XKB). This is an example of the additive functionality of the X Window System through its use of extensions.

Modern Linux distributions provide the `localectl` command via `systemd` which can also be used to modify a keyboard layout and will automatically create the `/etc/X11/xorg.conf.d/00-keyboard.conf` configuration file. Once more, here is an example setting up a Greek Polytonic keyboard on a Chromebook, this time using the `localectl` command:

```
$ localectl --no-convert set-x11-keymap "gr(polytonic)" chromebook
```

The `--no-convert` option is used here to prevent `localectl` from modifying the host's console keymap.

## Monitor

The `Monitor` section describes the physical monitor that is used and where it is connected. Here is an example configuration showing a hardware monitor connected to the second display port and is used as the primary monitor.

```
Section "Monitor"
    Identifier "DP2"
    Option "Primary" "true"
EndSection
```

## Device

The `Device` section describes the physical video card that is used. The section will also contain the kernel module used as the driver for the video card, along with its physical location on the motherboard.

```
Section "Device"
    Identifier "Device0"
    Driver     "i915"
    BusID     "PCI:0:2:0"
EndSection
```

## Screen

The `Screen` section ties the `Monitor` and `Device` sections together. An example `Screen` section could look like the following;

```
Section "Screen"
    Identifier "Screen0"
    Device    "Device0"
    Monitor   "DP2"
EndSection
```

## ServerLayout

The `ServerLayout` section groups all of the sections such as mouse, keyboard and screens into one X Window System interface.

```
Section "ServerLayout"
    Identifier "Layout-1"
    Screen    "Screen0" 0 0
    InputDevice "mouse1" "CorePointer"
    InputDevice "system-keyboard" "CoreKeyboard"
EndSection
```

### NOTE

Not all sections may be found within a configuration file. In instances where a section is missing, default values are provided by the running X server instance instead.

User-specified configuration files also reside in `/etc/X11/xorg.conf.d/`. Configuration files provided by the distribution are located in `/usr/share/X11/xorg.conf.d/`. The configuration files located within `/etc/X11/xorg.conf.d/` are parsed prior to the `/etc/X11/xorg.conf` file if it exists on the system.

The `xdpinfo` command is used on a computer to display information about a running X server instance. Below is sample output from the command:

```
$ xdpinfo
name of display:      :0
version number:      11.0
vendor string:       The X.Org Foundation
vendor release number: 12004000
X.Org version:       1.20.4
maximum request size: 16777212 bytes
motion buffer size:  256
bitmap unit, bit order, padding:  32, LSBFirst, 32
image byte order:    LSBFirst
number of supported pixmap formats:  7
supported pixmap formats:
    depth 1, bits_per_pixel 1, scanline_pad 32
    depth 4, bits_per_pixel 8, scanline_pad 32
    depth 8, bits_per_pixel 8, scanline_pad 32
    depth 15, bits_per_pixel 16, scanline_pad 32
    depth 16, bits_per_pixel 16, scanline_pad 32
    depth 24, bits_per_pixel 32, scanline_pad 32
    depth 32, bits_per_pixel 32, scanline_pad 32
keycode range:      minimum 8, maximum 255
focus:             None
number of extensions:  25
    BIG-REQUESTS
    Composite
    DAMAGE
    DOUBLE-BUFFER
    DRI3
    GLX
    Generic Event Extension
    MIT-SCREEN-SAVER
    MIT-SHM
    Present
    RANDR
    RECORD
    RENDER
    SECURITY
    SHAPE
    SYNC
    X-Resource
    XC-MISC
    XFIXES
    XFree86-VidModeExtension
    XINERAMA
    XInputExtension
```

```

XKEYBOARD
XTEST
XVideo
default screen number: 0
number of screens: 1

screen #0:
dimensions: 3840x1080 pixels (1016x286 millimeters)
resolution: 96x96 dots per inch
depths (7): 24, 1, 4, 8, 15, 16, 32
root window id: 0x39e
depth of root window: 24 planes
number of colormaps: minimum 1, maximum 1
default colormap: 0x25
default number of colormap cells: 256
preallocated pixels: black 0, white 16777215
options: backing-store WHEN MAPPED, save-unders NO
largest cursor: 3840x1080
current input event mask: 0xda0033
  KeyPressMask           KeyReleaseMask         EnterWindowMask
  LeaveWindowMask        StructureNotifyMask     SubstructureNotifyMask
  SubstructureRedirectMask PropertyChangeMask      ColormapChangeMask
number of visuals: 270
...

```

The more relevant portions of the output are in bold, such as the name of the display (which is the same as the contents of the `DISPLAY` environment variable), the versioning information of the X server in use, the number and listing of Xorg extensions in use, and further information about the screen itself.

## Creating a Basic Xorg Configuration File

Even though X will create its configuration after system startup on modern Linux installations, an `xorg.conf` file can still be used. To generate a permanent `/etc/X11/xorg.conf` file, run the following command:

```
$ sudo Xorg -configure
```

### NOTE

If there is already an X session running, you will need to specify a different `DISPLAY` in your command, for example:

```
$ sudo Xorg :1 -configure
```

On some Linux distributions, the `X` command can be used in place of `Xorg`, as `X` is a symbolic link to `Xorg`.

An `xorg.conf.new` file will be created in your current working directory. The contents of this file are derived from what the X server has found to be available in hardware and drivers on the local system. To use this file, it will need to be moved to the `/etc/X11/` directory and renamed to `xorg.conf`:

```
$ sudo mv xorg.conf.new /etc/X11/xorg.conf
```

**NOTE**

The following manual pages provide further information on components of the X Window System: `xorg.conf(5)`, `Xserver(1)`, `X(1)` and `Xorg(1)`.

## Wayland

Wayland is the newer display protocol designed to replace the X Window System. Many modern Linux distributions are using it as their default display server. It is meant to be lighter on system resources and have a smaller installation footprint than X. The project began in 2010 and is still undergoing active development, including work from active and former X.org developers.

Unlike the X Window System, there is no server instance that runs between the client and kernel. Instead, a client window works with its own code or that of a toolkit (such as `Gtk+` or `Qt`) to provide the rendering. In order to do the rendering, a request is made to the Linux kernel via the Wayland protocol. The kernel forwards the request via the Wayland protocol to the Wayland *compositor*, which handles device input, window management and composition. The compositor is the part of the system that combines the rendered elements into visual output on the screen.

Most modern toolkits such `Gtk+ 3` and `Qt 5` have been updated to allow for rendering to either an X Window System or a computer running Wayland. Not all standalone applications have been written to support rendering in Wayland as of yet. For applications and frameworks that are still targeting the X Window System to run, the application can run within *XWayland*. The *XWayland* system is a separate X server that runs within a Wayland client and thus renders a client window's contents within a standalone X server instance.

Just as the X Window System uses a `DISPLAY` environment variable to keep track of screens in use, the Wayland protocol uses a `WAYLAND_DISPLAY` environment variable. Below is sample output from a system running a Wayland display:

```
$ echo $WAYLAND_DISPLAY  
wayland-0
```

This environment variable is not available on systems running X.



## Guided Exercises

1. What command would you use to determine what Xorg extensions are available on a system?

2. You have just received a brand new 10-button mouse for your computer, however it will require extra configuration in order to get all of the buttons to function properly. Without modifying the rest of the X server configuration, what directory would you use to create a new configuration file for this mouse, and what specific configuration section would be used in this file?

3. What component of a Linux installation is responsible for keeping an X server running?

4. What command line switch is used with the `X` command to create a new `xorg.conf` configuration file?

## Explorational Exercises

1. What would the contents of the `DISPLAY` environment variable be on a system named `lab01` using a single display configuration? Assume that the `DISPLAY` environment variable is being viewed in a terminal emulator on the third independent screen.

2. What command can be used to create a keyboard configuration file for use by the X Window System?

3. On a typical Linux installation a user can switch to a virtual terminal by pressing the `Ctrl` + `Alt` + `F1-F6` keys on a keyboard. You have been asked to set up a kiosk system with a graphical interface and need this feature disabled to prevent unauthorized tampering with the system. You decide to create a `/etc/X11/xorg.conf.d/10-kiosk.conf` configuration file. Using a `ServerFlags` section (which is used to set global Xorg options on the server), what option would need to be specified? Review the `xorg(1)` man page to locate the option.

## Summary

This lesson covered the X Window System as it is used on Linux. The X Window System is used to draw images and text on screens, as they are defined in various configuration files. The X Window System is often used to configure input devices such as mice and keyboards. This lesson discussed the following points:

- The X Window System architecture at a high level.
- What configuration files are used to configure an X Windows System, and their location on the filesystem.
- How to use the `DISPLAY` environment variable on a system running X.
- A brief introduction to the Wayland display protocol.

The commands and configuration files addressed were:

- Modification of a keyboard's layout within an Xorg installation with `setxkbmap` and `localectl`.
- The `Xorg` command for creating a new `/etc/X11/xorg.conf` configuration file.
- The contents of the Xorg configuration files located at: `/etc/X11/xorg.conf`, `/etc/X11/xorg.conf.d/` and `/usr/share/X11/xorg.conf.d/`.
- The `xdpinfo` command for displaying general information about a running X server session.

## Answers to Guided Exercises

1. What command would you use to determine what Xorg extensions are available on a system?

```
$ xdpinfo
```

2. You have just received a brand new 10-button mouse for your computer, however it will require extra configuration in order to get all of the buttons to function properly. Without modifying the rest of the X server configuration, what directory would you use to create a new configuration file for this mouse, and what specific configuration section would be used in this file?

User-defined configurations should be located in `/etc/X11/xorg.conf.d/` and the specific section needed for this mouse configuration would be `InputDevice`.

3. What component of a Linux installation is responsible for keeping an X server running?

The display manager.

4. What command line switch is used with the `X` command to create a new `xorg.conf` configuration file?

```
-configure
```

Remember that the `X` command is a symbolic link to the `Xorg` command.

## Answers to Explorational Exercises

1. What would the contents of the `DISPLAY` environment variable be on a system named `lab01` using a single display configuration? Assume that the `DISPLAY` environment variable is being viewed in a terminal emulator on the third independent screen.

```
$ echo $DISPLAY
lab01:0.2
```

2. What command can be used to create a keyboard configuration file for use by the X Window System?

```
$ localectl
```

3. On a typical Linux installation a user can switch to a virtual terminal by pressing the `Ctrl` + `Alt` + `F1`-`F6` keys on a keyboard. You have been asked to set up a kiosk system with a graphical interface and need this feature disabled to prevent unauthorized tampering with the system. You decide to create a `/etc/X11/xorg.conf.d/10-kiosk.conf` configuration file. Using a `ServerFlags` section (which is used to set global Xorg options on the server), what option would need to be specified? Review the `xorg.conf(5)` man page to locate the option.

```
Section "ServerFlags"
    Option "DontVTSwitch" "True"
EndSection
```



## 106.2 Graphical Desktops

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 106.2](#)

### Weight

1

### Key knowledge areas

- Awareness of major desktop environments
- Awareness of protocols to access remote desktop sessions

### Partial list of the used files, terms and utilities

- KDE
- Gnome
- Xfce
- X11
- XDMCP
- VNC
- Spice
- RDP



## 106.2 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	106 User Interfaces and Desktops
<b>Objective:</b>	106.2 Graphical Desktops
<b>Lesson:</b>	1 of 1

### Introduction

Linux based operating systems are known for their advanced command line interface, but it can be intimidating for non-technical users. Intending to make computer usage more intuitive, the combination of high resolution displays with pointing devices gave birth to picture driven user interfaces. Whilst the command line interface requires prior knowledge about program names and their configuration options, with a *graphical user interface* (GUI) program functionality can be triggered by pointing to familiar visual elements, making the learning curve less steep. Furthermore, the graphical interface is best suited for multimedia and other visually oriented activities.

Indeed, the graphical user interface is almost a synonym to computer interface and most Linux distributions come with the graphical interface installed by default. There is not, however, a single monolithic program accountable for the full featured graphical desktops available in Linux systems. Instead, each graphical desktop is in fact a large collection of programs and their dependencies, varying from distribution's choices to the user's personal taste.

## X Window System

In Linux and other Unix-like operating systems where it's employed, the *X Window System* (also known as *X11* or just *X*) provides the low-level resources related to the graphical interface rendering and the user interaction with it, such as:

- The handling of the input events, like mouse movements or keystrokes.
- The ability to cut, copy and paste text content between separate applications.
- The programming interface other programs resort to draw the graphic elements.

Although the X Window System is in charge of controlling the graphic display (the video driver itself is part of X), it is not intended to draw complex visual elements on its own. Shapes, colors, shades and any other visual effects are generated by the application running on top of X. This approach gives applications a lot of room to create customized interfaces, but it can also lead to development overhead that is beyond the application scope and to inconsistencies in appearance and behaviour when compared to other program interfaces.

From the developer's perspective, the introduction of the *desktop environment* facilitates the GUI programming bound to the underlying application development, whereas from the user's perspective it gives a consistent experience among distinct applications. Desktop environments bring together programming interfaces, libraries and supporting programs that cooperate to deliver traditional but still evolving design concepts.

## Desktop Environment

The traditional desktop computer GUI consists of various windows—the term *window* is used here to refer to any autonomous screen area—associated with running processes. As the X Window System alone offers only basic interactive features, the full user experience depends on the components provided by the desktop environment.

Probably the most important component of a desktop environment, the *window manager* controls window placement and decorations. It is the window manager that adds the title bar to the window, the control buttons—generally associated with the minimize, maximize and close actions—and manages the switching between open windows.

### NOTE

The basic concepts found in graphic interfaces of desktop computers came from ideas taken from actual office workspaces. Metaphorically speaking, the computer screen is the desktop, where objects like documents and folders are placed. An application window with the content of a document mimics physical acts like filling in a form or painting a picture. As actual desktops, computer desktops also



have software accessories like notepads, clocks, calendars, etc., most of them based on their “real” counterparts.

All desktop environments provide a window manager that matches the look and feel of its *widget toolkit*. Widgets are informative or interactive visual elements, like buttons or text input fields, distributed inside the application window. The standard desktop components—like the application launcher, taskbar, etc.—and the window manager itself rely on such widget toolkits to assemble their interfaces.

Software libraries, like *GTK+* and *Qt*, provide widgets that programmers can use to build elaborate graphical interfaces for their applications. Historically, applications developed with *GTK+* didn't look like applications made with *Qt* and vice-versa, but the better theme support of today's desktop environments make the distinction less obvious.

Overall, *GTK+* and *Qt* offer the same features regarding widgets. Simple interactive elements can be indistinguishable, whereas composite widgets—such as the dialog window used by applications to open or save files—can, however, look quite different. Nonetheless, applications built with distinct toolkits can run simultaneously, regardless of the widget toolkit used by the other desktop components.

In addition to the basic desktop components, which could be considered individual programs on their own, desktop environments pursue the desktop metaphor by providing a minimal set of accessory applications developed under the same design guidelines. Variations of the following applications are commonly provided by all major desktop environments:

### **System related applications**

Terminal emulator, file manager, package installation manager, system configuration tools.

### **Communication and Internet**

Contacts manager, email client, web browser.

### **Office applications**

Calendar, calculator, text editor.

Desktop environments may include many other services and applications: the login screen greeter, session manager, inter-process communication, keyring agent, etc. They also incorporate features provided by third-party system services, such as *PulseAudio* for sound and *CUPS* for printing. These features do not need the graphical environment to work, but the desktop environment provides graphical frontends to facilitate the configuration and operation of such resources.

## Popular Desktop Environments

Many proprietary operating systems support only a single official desktop environment, which is tied to their particular release and it is not supposed to be changed. Unlike them, Linux-based operating systems support different desktop environment options that can be used in conjunction with X. Each desktop environment has its own features, but they usually share some common design concepts:

- An application launcher listing the builtin and third-party applications available in the system.
- Rules defining the default applications associated to file types and protocols.
- Configuration tools to customize the appearance and behaviour of the desktop environment.

*Gnome* is one of the most popular desktop environments, being the first choice in distributions like Fedora, Debian, Ubuntu, SUSE Linux Enterprise, Red Hat Enterprise Linux, CentOS, etc. In its version 3, Gnome brought major changes in its look and structure, moving away from the desktop metaphor and introducing the *Gnome Shell* as its new interface.



Figure 1. Gnome Shell Activities

The general purpose full-screen launcher *Gnome Shell Activities* replaced the traditional application launcher and taskbar. However, it is still possible to use Gnome 3 with the old look by choosing the *Gnome Classic* option in the login screen.

*KDE* is a large ecosystem of applications and development platform. Its latest desktop environment version, *KDE Plasma*, is used by default in openSUSE, Mageia, Kubuntu, etc. The employment of the Qt library is KDE's striking feature, giving it its unmistakable appearance and a plethora of original applications. KDE even provides a configuration tool to ensure visual

cohesion with GTK+ applications.

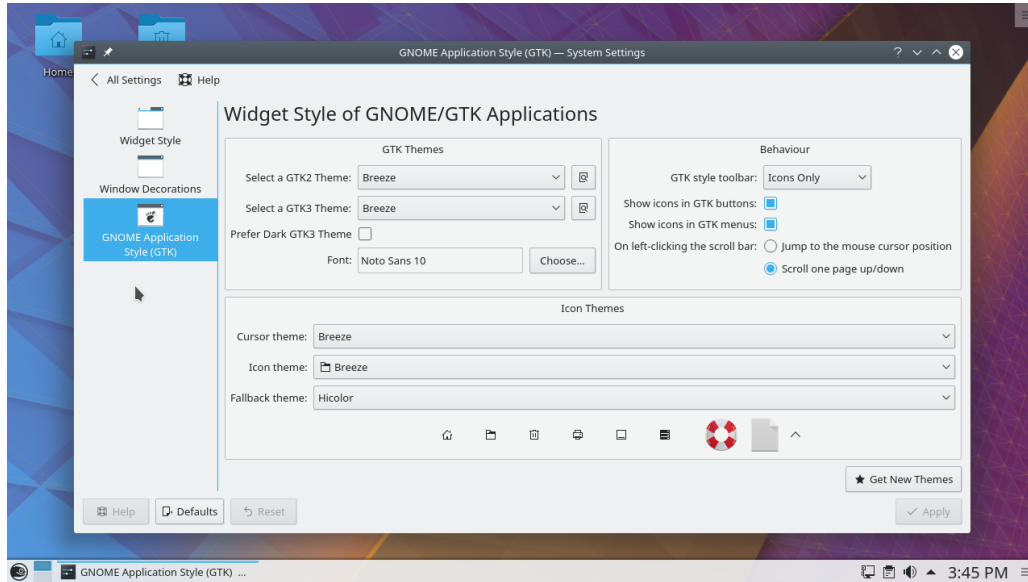


Figure 2. KDE GTK Settings

*Xfce* is a desktop environment that aims to be aesthetically pleasing while not consuming a lot of machine resources. Its structure is highly modularized, allowing a user to activate and to deactivate components according to the user's needs and preferences.

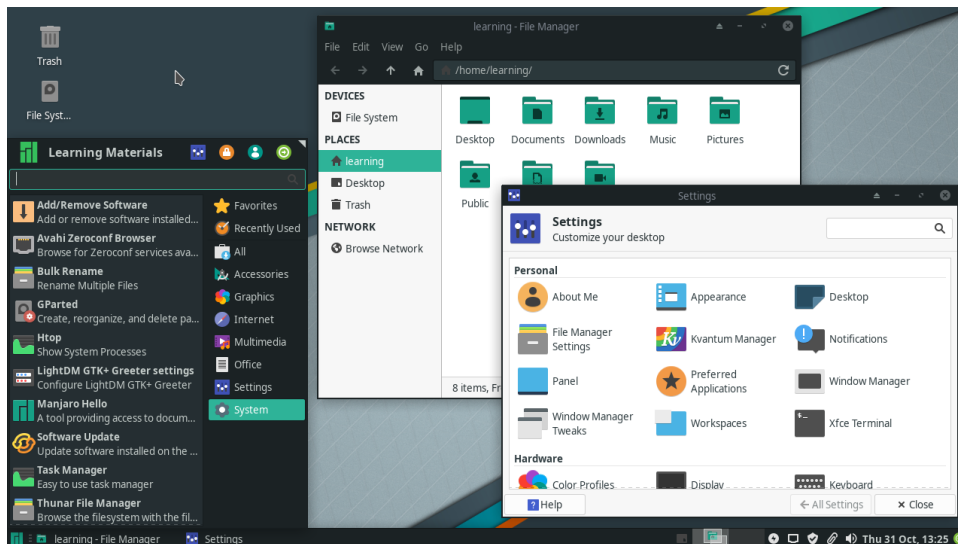


Figure 3. The Xfce desktop

There are many other desktop environments for Linux, usually provided by alternative distribution spins. The Linux Mint distribution, for example, provides two original desktop environments: *Cinnamon* (a fork of Gnome 3) and *MATE* (a fork of Gnome 2). *LXDE* is a desktop environment tailored to low resource consumption, which makes it a good choice for installation

on older equipment or single board computers. While not offering all the features of heavier desktop environments, LXDE offers all the basic features expected from a modern graphical user interface.

**TIP**

Keyboard shortcuts play an important role in the interaction with the desktop environment. Some keyboard shortcuts — such as `Alt` + `Tab` to switch between windows or `Ctrl` + `c` to copy text — are universal across desktop environments, but each desktop environment has its own keyboard shortcuts too. Keyboard shortcuts are found in the keyboard configuration tool provided by the desktop environment, where shortcuts can be added or modified.

## Desktop Interoperability

The diversity of desktop environments in Linux-based operating systems imposes a challenge: how to make them work correctly with third-party graphical applications or system services without having to implement specific support for each of them. Shared methods and specifications across desktop environments greatly improve user experience and settles many development issues, as graphical applications must interact with the current desktop environment regardless of the desktop environment they originally were designed for. In addition to that, it is important to keep general desktop settings if the user eventually changes their desktop environment choice.

A large body of specifications for desktop interoperability is maintained by the *freedesktop.org* organization. The adoption of the full specification is not mandatory, but many of them are widely used:

### Directories locations

Where the personal settings and other user-specific files are located.

### Desktop entries

Command line applications can run in the desktop environment through any terminal emulator, but it would be too confusing to make all of them available in the application launcher. Desktop entries are text files ending with `.desktop` which are used by the desktop environment to gather information about the available desktop applications and how to use them.

### Application autostart

Desktop entries indicating the application that should start automatically after the user has logged in.

## Drag and drop

How applications should handle drag and drop events.

## Trash can

The common location of files deleted by the file manager, as well as the methods to store and remove files from there.

## Icon themes

The common format for interchangeable icon libraries.

The ease of use provided by desktop environments has a downside compared to text interfaces such as the shell: the ability to provide remote access. While a remote machine command-line shell environment can be easily accessed with tools such as `ssh`, remote access to graphical environments requires different methods and may not achieve satisfactory performance on slower connections.

## Non-Local Access

The X Window Systems adopts a design based on autonomous *displays*, where the same *X display manager* can control more than one graphical desktop session at the same time. In essence, a display is analogous to a text terminal: both refer to a machine or software application used as an entry point to establish an independent operating system session. Although the most common setup involves a singular graphical session running in the local machine, other less conventional setups are also possible:

- Switch between active graphical desktop sessions in the same machine.
- More than one set of display devices (e.g. screen, keyboard, mouse) connected to the same machine, each one controlling its own graphical desktop session.
- Remote graphical desktop sessions, where the graphical interface is sent through the network to a remote display.

Remote desktop sessions are supported natively by X, which employs the *X Display Manager Control Protocol* (XDMCP) to communicate with remote displays. Due to its high bandwidth usage, XDMCP is rarely used through the internet or in low-speed LANs. Security issues are also a concern with XDMCP: the local display communicates with a privileged remote X display manager to execute remote procedures, so an eventual vulnerability could make it possible to execute arbitrary privileged commands on the remote machine.

Furthermore, XDMCP requires X instances running on both ends of the connection, which can make it unfeasible if the X Windows System is not available for all the machines involved. In

practice, other more efficient and less invasive methods are used to establish remote graphical desktop sessions.

*Virtual Network Computing* (VNC) is a platform-independent tool to view and control remote desktop environments using the *Remote Frame Buffer* protocol (RFB). Through it, events produced by the local keyboard and mouse are transmitted to the remote desktop, which in turn sends back any screen updates to be displayed locally. It is possible to run many VNC servers in the same machine, but each VNC server needs an exclusive TCP port in the network interface accepting incoming session requests. By convention, the first VNC server should use TCP port 5900, the second should use 5901, and so on.

The VNC server does not need special privileges to run. An ordinary user can, for example, log in to their remote account and start their own VNC server from there. Then, in the local machine, any VNC client application can be used to access the remote desktop (assuming the corresponding network ports are reachable). The `~/ .vnc/xstartup` file is a shell script executed by the VNC server when it starts and can be used to define which desktop environment the VNC server will make available for the VNC client. It is important to note that VNC does not provide modern encryption and authentication methods natively, so it should be used in conjunction with a third-party application that provides such features. Methods involving VPN and SSH tunnels are often used to secure VNC connections.

The *Remote Desktop Protocol* (RDP) is mainly used to remotely access the desktop of a *Microsoft Windows* operating system through the TCP 3389 network port. Although it uses Microsoft's proprietary RDP protocol, the client implementation used in Linux systems are open-source programs licensed under the *GNU General Public License* (GPL) and has no legal restrictions on use.

*Simple Protocol for Independent Computing Environments* (Spice) comprises a suite of tools aimed at accessing the desktop environment of *virtualised* systems, either in the local machine or in a remote location. In addition to that, the Spice protocol offers native features to integrate the local and remote systems, like the ability to access local devices (for example, the sound speakers and the connected USB devices) from the remote machine and file sharing between the two systems.

There are specific client commands to connect to each one of these remote desktop protocols, but the *Remmina* remote desktop client provides an integrated graphical interface that facilitates the connection process, optionally storing the connection settings for later use. Remmina has plugins for each individual protocol and there are plugins for XDMCP, VNC, RDP and Spice. The choice of the right tool depends on the operating systems involved, the quality of the network connection and what features of the remote desktop environment should be available.

## Guided Exercises

1. What type of application provides windowed shell sessions in the desktop environment?

2. Due to the variety of Linux desktop environments, the same application may have more than one version, each of them best suited for a particular widget toolkit. For example, the bittorrent client *Transmission* has two versions: *transmission-gtk* and *transmission-qt*. Which of the two should be installed to ensure maximum integration with KDE?

3. What Linux desktop environments are recommended for low-cost single board computers with little processing power?

## Explorational Exercises

1. There are two ways to copy and paste text in the X Window System: using the traditional `Ctrl + c` and `Ctrl + v` keystrokes (also available in the window menu) or to use the mouse middle-button click to paste the currently selected text. What's the appropriate way to copy and paste text from a terminal emulator?

2. Most desktop environments assign the `Alt + F2` keyboard shortcut to the *Run program* window, where programs can be executed in a command line fashion. In KDE, what command would execute the default terminal emulator?

3. What protocol is best suited to access a remote Windows desktop from a Linux desktop environment?



## Summary

This lesson is an overview of graphical desktops available for Linux systems. The X Window System alone provides only simple interface features, so desktop environments extend the user experience in the graphical windowed interface. The lesson goes through the following topics:

- Graphic interface and X Window System concepts.
- Desktop environments available for Linux.
- Similarities and differences between desktop environments.
- How to access a remote desktop environment.

The concepts and programs addressed were:

- X Window System.
- Popular desktop environments: KDE, Gnome, Xfce.
- Remote access protocols: XDMCP, VNC, RDP, Spice.

## Answers to Guided Exercises

1. What type of application provides windowed shell sessions in the desktop environment?

Any terminal emulator like Konsole, Gnome terminal, xterm, etc., will give access to a local interactive shell session.

2. Due to the variety of Linux desktop environments, the same application may have more than one version, each of them best suited for a particular widget toolkit. For example, the bittorrent client *Transmission* has two versions: *transmission-gtk* and *transmission-qt*. Which of the two should be installed to ensure maximum integration with KDE?

KDE is built on top of the Qt library, so the Qt version — *transmission-qt* — should be installed.

3. What Linux desktop environments are recommended for low-cost single board computers with little processing power?

Basic desktop environments that don't use too much visual effects, such as Xfce and LXDE.

## Answers to Explorational Exercises

1. There are two ways to copy and paste text in the X Window System: using the traditional `Ctrl + c` and `Ctrl + v` keystrokes (also available in the window menu) or to use the mouse middle-button click to paste the currently selected text. What's the appropriate way to copy and paste text from a terminal emulator?

Interactive shell sessions assign the `Ctrl + c` keystroke to stop program execution, so the middle-button method is recommended.

2. Most desktop environments assign the `Alt + F2` keyboard shortcut to the *Run program* window, where programs can be executed in a command line fashion. In KDE, what command would execute the default terminal emulator?

The command `konsole` executes KDE's terminal emulator, but generic terms like *terminal* also works.

3. What protocol is best suited to access a remote Windows Desktop from a Linux desktop environment?

The Remote Desktop Protocol (RDP), as it is natively supported by both Windows and Linux.



## 106.3 Accessibility

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 106.3](#)

### Weight

1

### Key knowledge areas

- Basic knowledge of visual settings and themes.
- Basic knowledge of assistive technology.

### Partial list of the used files, terms and utilities

- High Contrast/Large Print Desktop Themes.
- Screen Reader.
- Braille Display.
- Screen Magnifier.
- On-Screen Keyboard.
- Sticky/Repeat keys.
- Slow/Bounce/Toggle keys.
- Mouse keys.
- Gestures.
- Voice recognition.



# 106.3 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	106 User Interfaces and Desktops
<b>Objective:</b>	106.3 Accessibility
<b>Lesson:</b>	1 of 1

## Introduction

The Linux desktop environment has many settings and tools to adapt the user interface for people with disabilities. The ordinary human interface devices—screen, keyboard and mouse/touchpad—can be individually reconfigured to overcome visual impairments or reduced mobility.

It is possible, for example, to adjust the desktop color scheme to better serve color-blind people. Also, people suffering from repetitive strain injury may take advantage of alternative typing and pointing methods.

Some of these accessibility features are provided by the desktop environment itself, like Gnome or KDE, and others are provided by additional programs. In the latter case, it is important to pick the tool that best integrates the desktop environment, so that the aid is of better quality.

## Accessibility Settings

All major Linux distributions provide about the same accessibility features, which can be customized with a configuration module found in the settings manager that come with the

desktop environment. The accessibility settings module is called *Universal Access* in the Gnome desktop, whereas in KDE is under *System Settings, Personalization, Accessibility*. Other desktop environments, like *Xfce*, also call it *Accessibility* in their graphical settings manager. However, they offer a reduced set of functionalities compared to Gnome and KDE.

Gnome, for instance, can be configured to permanently show the Universal Access menu in the top-right corner of the screen, where the accessibility options can be quickly switched on. It can be used, for example, to replace the sound alert by a visual one, so users with hearing impairment will be able to perceive system alerts more easily. Although KDE does not have a similar quick-access menu, the visual alert feature is also available, but is called *visual bell* instead.

## Keyboard and Mouse Assist

The default keyboard and mouse behaviour can be modified to get around specific mobility difficulties. Key combinations, key auto repeat rate and unintended key presses can be significant obstacles for users with reduced hand mobility. These typing shortcomings are addressed by three keyboard related accessibility features: *Sticky keys*, *Bounce keys* and *Slow keys*.

The *Sticky keys* feature, found in the *Typing Assist* section of Gnome's Universal Access configuration, allows the user to type keyboard shortcuts one key at a time. When enabled, key combinations like `Ctrl + C` do not need to be held down at the same time. The user may first press the `Ctrl` key, release it and then press the `C` key. In KDE, this option is in the *Modifier Keys* tab of the Accessibility settings window. KDE also offers the *Locking Keys* option: if enabled, the `Alt`, `Ctrl` and Shift keys will stay "down" if the user presses them twice, similar to the Caps lock key behaviour. Like the Caps Lock feature, the user will need to press the corresponding key again to release it.

The *Bounce keys* feature tries to inhibit unintended key presses by placing a delay between them, that is, a new key press will be accepted only after a specified length of time has passed since the last key press. Users with hand tremors may find the Bounce keys feature helpful to avoid pressing a key multiple times when they only intend to press it once. In Gnome, this feature concerns only same key repetitions, while in KDE it concerns any other key press as well and is found in the *Keyboard Filters* tab.

The *Slow keys* feature also helps to avoid accidental key strokes. When enabled, Slow keys will require the user to hold down the key for a specified length of time before it is accepted. Depending on the user's needs, it may also be helpful to adjust the automatic repetition while holding a key down, available in the keyboard settings.

The Sticky keys and Slow keys accessibility features can be turned on and off with *Activation Gestures* performed in the keyboard. In KDE, the *Use gestures for activating sticky keys and slow keys* option should be checked to enable Activation Gestures, while in Gnome this feature is called

*Enable by Keyboard* in the *Typing Assist* configuration window. Once the Activation Gestures are enabled, the Sticky keys feature will be activated after pressing the Shift key five consecutive times. To activate the Slow keys feature, the Shift key must be held down for eight consecutive seconds.

Users who find it more comfortable to use keyboard over the mouse or touchpad can resort to keyboard shortcuts to get around in the desktop environment. Furthermore, a feature called *Mouse Keys* allows the user to control the mouse pointer itself with the numerical keypad, which is present in full-sized desktop keyboards and in larger laptops.

The numerical keypad is arranged into a square grid, so each number corresponds to a direction: **2** moves the cursor downwards, **4** moves the cursor to the left, **7** moves do cursor north-west, etc. By default, number **5** corresponds to the left mouse click.

Whilst in Gnome there is only a switch to enable the Mouse Keys option at the Universal Access settings window, in KDE the Mouse Keys settings are located at *System Settings, Mouse, Keyboard Navigation*, and options like speed and acceleration can be customized.

**TIP**

The Slow keys, Stick keys, Bounce keys and Mouse keys are accessibility features provided by AccessX, a resource within the X keyboard extension of the X Window System. AccessX settings can also be modified from the command line, with the `xkbset` command.

The mouse or touchpad can be used to generate keyboard input when keyboard usage is too uncomfortable or not possible. If the *Screen Keyboard* switch in Gnome's Universal Access settings is enabled, then an on-screen keyboard will appear every time the cursor is in a text field and new text is entered by clicking the keys with the mouse or touchscreen, much like the virtual keyboard of smartphones.

KDE and other desktop environments may not provide the screen keyboard by default, but the *onboard* package can be manually installed to provide a simple on-screen keyboard that can be used in any desktop environment. After installation, it will be available as a regular application in the application launcher.

The pointer behaviour can also be modified if clicking and dragging the mouse causes pain or is impracticable for any other reason. If the user is not able to click the mouse button quick enough to trigger a double-click event, for example, the time interval to press the mouse button a second time to double-click can be increased in the *Mouse Preferences* in the system configuration window.

If the user is not able to press one of the mouse buttons or none of the mouse buttons, then mouse clicks can be simulated using different techniques. In the *Click Assist* section of the *Gnome*

*Universal Access*, the *Simulate a right mouse click* option will generate a right-click if the user presses and holds the left mouse button. With the *Simulate clicking by hovering* option enabled, a click event will be triggered when the user holds the mouse still. In KDE, the *KMouseTool* application will provide these same features to assist mouse actions.

## Visual Impairments

Users with reduced eyesight may still be able to use the monitor screen to interact with the computer. Depending on the user's needs, many visual adjustments can be made to improve the otherwise hard to see details of the standard graphical desktop.

Gnome's *Seeing* section of *Universal Access* settings provide options that can help people with reduced eyesight:

### High Contrast

will make windows and buttons easier to see by drawing them in sharper colors.

### Large Text

will enlarge the standard screen font size.

### Cursor Size

allows to choose a bigger mouse cursor, making it easier to locate on the screen.

Some of these adjustments are not strictly related to accessibility features, thus can be found in the appearance section of the configuration utility provided by other desktop environments. A user having difficulties discerning between visual elements can choose a high-contrast theme to make it easier to identify buttons, overlapping windows, etc.

If appearance adjustments alone are not enough to improve screen readability, then a screen magnifier program can be used to zoom in on parts of the screen. This feature is called *Zoom* in Gnome's *Universal Access* settings, where options like magnification ratio, position of the magnifier and color adjustments can be customized.

In KDE, the *KMagnifier* program provides the same features, but it is available as an ordinary application through the application launcher. Other desktop environments may provide their own screen magnifiers. Xfce, for example, will zoom in and out the screen by rotating the mouse scrolling wheel while the Alt key is down.

Finally, users for whom the graphical interface is not an option can use a *screen reader* to interact with the computer. Regardless of the chosen desktop environment, the most popular screen reader for Linux systems is *Orca*, which is usually installed by default in most distributions. Orca



generates a synthesized voice to report screen events and to read the text under the mouse cursor. Orca also works with *refreshable braille displays*, special devices that display braille characters by raising small pins that can be felt with the fingertips. Not all desktop applications are fully adapted for screen readers and not all users will find it easy to use them, so it is important to provide as many screen reading strategies as possible for users to choose from.

## Guided Exercises

1. What accessibility feature could help a user to alternate between open windows using the keyboard, considering that the user is unable to press the `Alt` and `Tab` keys at the same time?

2. How could the *Bounce keys* accessibility feature help users whose involuntary hand tremors disturb their typing?

3. What is the most common Activation Gestures for the *Sticky keys* accessibility feature?

---

## Explorational Exercises

1. Accessibility features may not be provided by a single application and may vary from one desktop environment to another. In KDE, what application helps those with repetitive strain injuries by clicking the mouse whenever the mouse cursor pauses briefly?

2. What appearance aspects of the graphical environment can be modified to make it easier for people to read text on the screen?

3. In what ways can the *Orca* application help visually impaired users to interact with the desktop environment?

## Summary

This lesson covers general accessibility features available in Linux systems. All major desktop environments, specially Gnome and KDE, provide many builtin and third-party applications to assist people with visual impairments or reduced mobility. The lesson goes through the following topics:

- How to change accessibility settings.
- Alternative ways to use the keyboard and mouse.
- Desktop enhancements for the visually impaired.

The commands and procedures addressed were:

- Keyboard accessibility settings: Sticky keys, Slow keys, Bounce keys.
- Artificially generate mouse events.
- On-screen keyboard.
- Visual settings to enhance readability.
- High Contrast/Large Print Desktop Themes.
- Screen magnifiers.
- The Orca screen reader.

## Answers to Guided Exercises

1. What accessibility feature could help a user to alternate between open windows using the keyboard, considering that the user is unable to press the `Alt` and `Tab` keys at the same time?

The Sticky keys feature, which allows the user to type keyboard shortcuts one key at a time.

2. How could the *Bounce keys* accessibility feature help users whose involuntary hand tremors disturb their typing?

With Bounce keys enabled, a new key press will be accepted only after a specified length of time has passed since the last key press.

3. What is the most common Activation Gestures for the *Sticky keys* accessibility feature?

If Activation Gestures are enabled, the Sticky keys feature will be activated after pressing the `Shift` key five consecutive times.

## Answers to Explorational Exercises

1. Accessibility features may not be provided by a single application and may vary from one desktop environment to another. In KDE, what application helps those with repetitive strain injuries by clicking the mouse whenever the mouse cursor pauses briefly?

The *KMouseTool* application.

2. What appearance aspects of the graphical environment can be modified to make it easier for people to read text on the screen?

Setting a large screen font size in the desktop configuration will make all screen texts easier to read.

3. In what ways can the *Orca* application help visually impaired users to interact with the desktop environment?

Orca is a screen reader that generates a synthesized voice to report screen events and to read the text under the mouse cursor. It also works with devices called *refreshable braille displays*, so the user can identify the text with tactile patterns.



**Linux  
Professional  
Institute**

## **Topic 107: Administrative Tasks**



## 107.1 Manage user and group accounts and related system files

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 107.1](#)

### Weight

5

### Key knowledge areas

- Add, modify and remove users and groups.
- Manage user/group info in password/group databases.
- Create and manage special purpose and limited accounts.

### Partial list of the used files, terms and utilities

- `/etc/passwd`
- `/etc/shadow`
- `/etc/group`
- `/etc/skel/`
- `chage`
- `getent`
- `groupadd`
- `groupdel`
- `groupmod`
- `passwd`
- `useradd`



- `userdel`
- `usermod`



# 107.1 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	107 Administrative Tasks
<b>Objective:</b>	107.1 Manage user and group accounts and related system files
<b>Lesson:</b>	1 of 2

## Introduction

User and group administration is a very important part of any system administrator's job. Modern Linux distributions implement graphical interfaces that allow you to manage all the activities related to this key aspect quickly and easily. These interfaces are different from each other in terms of graphical layouts, but the features are the same. With these tools you can view, edit, add, and delete local users and groups. However for more advanced management you need to work through the command line.

## Adding User Accounts

In Linux, you can add a new user account with the `useradd` command. For example, acting with root privileges, you can create a new user account named `michael` with a default setting, using the following:

```
# useradd michael
```

When you run the `useradd` command, the user and group information stored in the password and group databases are updated for the newly created user account and, if specified, the home directory of the new user is created as well. A group with the same name of the new user account is also created.

Once you have created the new user, you can set its password using the `passwd` command. You can review its User ID (UID), Group ID (GID) and the groups it belongs to through the `id` and `groups` commands.

```
# passwd michael
Changing password for user michael.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
# id michael
uid=1000(michael) gid=100(michael) groups=100(michael)
# groups michael
michael : michael
```

**NOTE**

Remember that any user can review their UID, GID and the groups they belong to by simply using the `id` and `groups` commands without arguments, and that any user can change their password using the `passwd` command. However only users with root privileges can change *any* user's password.

The most important options which apply to the `useradd` command are:

**-c**

Create a new user account with custom comments (for example the user's full name).

**-d**

Create a new user account with a custom home directory.

**-e**

Create a new user account by setting a specific date on which it will be disabled.

**-f**

Create a new user account by setting the number of days after a password expires during which the user should update the password (otherwise the account will be disabled).

**-g**

Create a new user account with a specific GID.

**-G**

Create a new user account by adding it to multiple secondary groups.

**-k**

Create a new user account by copying the skeleton files from a specific custom directory (this option is only valid if the `-m` or `--create-home` option is specified).

**-m**

Create a new user account with its home directory (if it does not exist).

**-M**

Create a new user account without its home directory.

**-s**

Create a new user account with a specific login shell.

**-u**

Create a new user account with a specific UID.

See the manual pages for the `useradd` command for the complete list of options.

## Modifying User Accounts

Sometimes you need to change an attribute of an existing user account, such as the login name, login shell, password expiry date and so on. In such cases, you need to use the `usermod` command.

```
# usermod -s /bin/tcsh michael
# usermod -c "Michael User Account" michael
```

Just as with the `useradd` command, the `usermod` command requires root privileges.

In the examples above, the login shell of `michael` is changed first and then a brief description is added to this user account. Remember that you can modify multiple attributes at once, specifying them in a single command.

The most important options which apply to the `usermod` command are:

**-c**

Add a brief comment to the specified user account.

**-d**

Change the home directory of the specified user account. When used with the `-m` option, the contents of the current home directory are moved to the new home directory, which is created if it does not already exist.

**-e**

Set the expiration date of the specified user account.

**-f**

Set the number of days after a password expires during which the user should update the password (otherwise the account will be disabled).

**-g**

Change the primary group of the specified user account (the group must exist).

**-G**

Add secondary groups to the specified user account. Each group must exist and must be separated from the next by a comma, with no intervening whitespace. If used alone, this option removes all existing groups to which the user belongs, while when used with the `-a` option, it simply appends new secondary groups to the existing ones.

**-l**

Change the login name of the specified user account.

**-L**

Lock the specified user account. This puts an exclamation mark in front of the encrypted password within the `/etc/shadow` file, thus disabling access with a password for that user.

**-s**

Change the login shell of the specified user account.

**-u**

Change the UID of the specified user account.

**-U**

Unlock the specified user account. This removes the exclamation mark in front of the encrypted password with the `/etc/shadow` file.

See the manual pages for the `usermod` command for the complete list of options.

**TIP**

Remember that when you change the login name of a user account, you should

probably rename the home directory of that user and other user-related items such as mail spool files. Also remember that when you change the UID of a user account, you should probably fix the ownership of files and directories outside the user's home directory (the user ID is changed automatically for the user's mailbox and for all files owned by the user and located in the user's home directory).

## Deleting User Accounts

If you want to delete a user account, you can use the `userdel` command. In particular, this command updates the information stored in the account databases, deleting all entries referring to the specified user. The `-r` option also removes the user's home directory and all its contents, along with the user's mail spool. Other files, located elsewhere, must be searched for and deleted manually.

```
# userdel -r michael
```

As for `useradd` and `usermod`, you need root authority to delete user accounts.

## Adding, Modifying and Deleting Groups

Just as with user management, you can add, modify and delete groups using the `groupadd`, `groupmod` and `groupdel` commands with root privileges. If you want to create a new group named `developer`, you can run the following command:

```
# groupadd -g 1090 developer
```

The `-g` option of this command creates a group with a specific GID.

### WARNING

Remember that when you add a new user account, the primary group and the secondary groups to which it belongs *must* exist before launching the `useradd` command.

Later, if you want to rename the group from `developer` to `web-developer` and change its GID, you can run the following:

```
# groupmod -n web-developer -g 1050 developer
```

### TIP

Remember that if you change the GID using the `-g` option, you should change the GID of all files and directories that must continue to belong to the group.

Finally, if you want to delete the `web-developer` group, you can run the following:

```
# groupdel web-developer
```

You cannot delete a group if it is the primary group of a user account. Therefore, you must remove the user before removing the group. As for users, if you delete a group, the files belonging to that group remain in your filesystem and are not deleted or assigned to another group.

## The Skeleton Directory

When you add a new user account, even creating its home directory, the newly created home directory is populated with files and folders that are copied from the skeleton directory (by default `/etc/skel`). The idea behind this is simple: a system administrator wants to add new users having the same files and directories in their home folder. Therefore, if you want to customize the files and folders that are created automatically in the home directory of new user accounts, you must add these new files and folders to the skeleton directory.

### TIP

Note that if you want to list all the files and directories in the skeleton directory, you must use the `ls -al` command.

## The `/etc/login.defs` File

In Linux, the `/etc/login.defs` file specifies the configuration parameters that control the creation of users and groups. In addition, the commands shown in the previous sections take default values from this file.

The most important directives are:

### `UID_MIN` and `UID_MAX`

The range of user IDs that can be assigned to new ordinary users.

### `GID_MIN` and `GID_MAX`

The range of group IDs that can be assigned to new ordinary groups.

### `CREATE_HOME`

Specify whether a home directory should be created by default for new users.

### `USERGROUPS_ENAB`

Specify whether the system should by default create a new group for each new user account with the same name as the user, and whether deleting the user account should also remove the

user's primary group if it no longer contains members.

### **MAIL\_DIR**

The mail spool directory.

### **PASS\_MAX\_DAYS**

The maximum number of days a password may be used.

### **PASS\_MIN\_DAYS**

The minimum number of days allowed between password changes.

### **PASS\_MIN\_LEN**

The minimum acceptable password length.

### **PASS\_WARN\_AGE**

The number of warning days before a password expires.

#### **TIP**

When managing users and groups, always check this file to view and eventually change the default behavior of the system if needed.

## **The passwd Command**

This command is primarily used to change a user's password. As described before, any user can change their own password, but only root can change *any* user's password. This happens because the `passwd` command has the SUID bit set (an `s` in the place of the executable flag for the owner), which means that it executes with the privileges of the file's owner (thus root).

```
# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 42096 mag 17 2015 /usr/bin/passwd
```

Depending on the `passwd` options used, you can control specific aspects of password aging:

#### **-d**

Delete the password of a user account (thus disabling the user).

#### **-e**

Force the user account to change the password.

#### **-i**

Set the number of days of inactivity after a password expires during which the user should



update the password (otherwise the account will be disabled).

**-l**

Lock the user account (the encrypted password is prefixed with an exclamation mark in the `/etc/shadow` file).

**-n**

Set the minimum password lifetime.

**-S**

Output information about the password status of a specific user account.

**-u**

Unlock the user account (the exclamation mark is removed from the password field in the `/etc/shadow` file).

**-x**

Set the maximum password lifetime.

**-w**

Set the number of days of warning before the password expires during which the user is warned that the password must be changed.

**NOTE**

Groups can also have a password, which can be set using the `gpasswd` command. Users, who are not members of a group but know its password, can join it temporarily using the `newgrp` command. Remember that `gpasswd` is also used to add and remove users from a group and to set the list of administrators and ordinary members of the group.

## The `chage` Command

This command, which stands for “change age”, is used to change the password aging information of a user. The `chage` command is restricted to root, except for the `-l` option, which can be used by ordinary users to list password aging information of their own account.

The other options which apply to the `chage` command are:

**-d**

Set the last password change for a user account.

**-E**

Set the expiration date for a user account.

**-I**

Set the number of days of inactivity after a password expires during which the user should update the password (otherwise the account will be disabled).

**-m**

Set the minimum password lifetime for a user account.

**-M**

Set the maximum password lifetime for a user account.

**-W**

Set the number of days of warning before the password expires during which the user is warned that the password must be changed.

## Guided Exercises

1. For each of the following commands, identify the corresponding purpose:

<code>usermod -L</code>	
<code>passwd -u</code>	
<code>chage -E</code>	
<code>groupdel</code>	
<code>useradd -s</code>	
<code>groupadd -g</code>	
<code>userdel -r</code>	
<code>usermod -l</code>	
<code>groupmod -n</code>	
<code>useradd -m</code>	

2. For each of the following `passwd` commands, identify the corresponding `chage` command:

<code>passwd -n</code>	
<code>passwd -x</code>	
<code>passwd -w</code>	
<code>passwd -i</code>	
<code>passwd -S</code>	

3. Explain in detail the purpose of the commands in the previous question:

4. What commands can you use to lock a user account? And which commands to unlock it?

# Explorational Exercises

1. Using the `groupadd` command, create the `administrators` and `developers` groups. Assume you are working as root.

2. Now that you have created these groups, run the following command: `useradd -G administrators,developers kevin`. What operations does this command perform? Assume that `CREATE_HOME` and `USERGROUPS_ENAB` in `/etc/login.defs` are set to `yes`.

3. Create a new group named `designers`, rename it to `web-designers` and add this new group to the secondary groups of the `kevin` user account. Identify all the groups `kevin` belongs to and their IDs.

4. Remove only the `developers` group from the secondary groups of `kevin`.

5. Set the password for the `kevin` user account.

6. Using the `chage` command, first check the expiry date of the `kevin` user account and then change it to December 31st 2022. What other command can you use to change the expiration date of a user account?

7. Add a new user account named `emma` with UID 1050 and set `administrators` as its primary group and `developers` and `web-designers` as its secondary groups.

8. Change the login shell of `emma` to `/bin/sh`.

9. Delete the `emma` and `kevin` user accounts and the `administrators`, `developers` and `web-designers` groups.

# Summary

In this lesson you learned:

- The fundamentals of user and group management in Linux.
- How to add, modify and remove user accounts.
- How to add, modify and remove group accounts.
- Maintain the skeleton directory.
- Edit the file that controls the creation of users and groups.
- Change the passwords of user accounts.
- Change the password aging information of user accounts.

The following files and commands were discussed in this lesson:

## **useradd**

Create a new user account.

## **usermod**

Modify a user account.

## **userdel**

Delete a user account.

## **groupadd**

Create a new group account.

## **groupmod**

Modify a group account.

## **groupdel**

Delete a group account.

## **passwd**

Change the password of user accounts and control all aspects of password aging.

## **chage**

Change user password expiry information.

### **/etc/skel**

The default location of the skeleton directory.

### **/etc/login.defs**

The file that controls the creation of users and groups and provides default values for several user account parameters.

## Answers to Guided Exercises

1. For each of the following commands, identify the corresponding purpose:

<code>usermod -L</code>	Lock the user account
<code>passwd -u</code>	Unlock the user account
<code>chage -E</code>	Set the expiration date for the user account
<code>groupdel</code>	Delete the group
<code>useradd -s</code>	Create a new user account with a specific login shell
<code>groupadd -g</code>	Create a new group with a specific GID
<code>userdel -r</code>	Remove the user account and all files in its home directory, the home directory itself and the user's mail spool
<code>usermod -l</code>	Change the login name of the user account
<code>groupmod -n</code>	Change the name of the group
<code>useradd -m</code>	Create a new user account and its home directory

2. For each of the following `passwd` commands, identify the corresponding `chage` command:

<code>passwd -n</code>	<code>chage -m</code>
<code>passwd -x</code>	<code>chage -M</code>
<code>passwd -w</code>	<code>chage -W</code>
<code>passwd -i</code>	<code>chage -I</code>
<code>passwd -S</code>	<code>chage -l</code>

3. Explain in detail the purpose of the commands in the previous question:

In Linux, you can use the `passwd -n` command (or `chage -m`) to set the minimum number of days between password changes, the `passwd -x` command (or `chage -M`) to set the maximum number of days during which a password is valid, the `passwd -w` command (or `chage -W`) to set the number of days of warning before the password expires, the `passwd -i` command (or `chage -I`) to set the number of days of inactivity during which the user should change the password and the `passwd -S` command (or `chage -l`) to show brief information about the

password of the user account.

4. What commands can you use to lock a user account? And which commands to unlock it?

If you want to lock an user account, you can use one of these commands: `usermod -L`, `usermod --lock` and `passwd -l`. Instead, if you want to unlock it, you can use `usermod -U`, `usermod --unlock` and `passwd -u`.



## Answers to Explorational Exercises

- Using the `groupadd` command, create the `administrators` and `developers` groups. Assume you are working as root.

```
# groupadd administrators
# groupadd developers
```

- Now that you have created these groups, run the following command: `useradd -G administrators,developers kevin`. What operations does this command perform? Assume that `CREATE_HOME` and `USERGROUPS_ENAB` in `/etc/login.defs` are set to `yes`.

The command adds a new user, named `kevin`, to the list of users in the system, creates its home directory (`CREATE_HOME` is set to `yes` and therefore you can omit the `-m` option) and creates a new group, named `kevin`, as the primary group of this user account (`USERGROUPS_ENAB` is set to `yes`). Finally, the files and folders contained in the skeleton directory are copied to the home dir of `kevin`.

- Create a new group named `designers`, rename it to `web-designers` and add this new group to the secondary groups of the `kevin` user account. Identify all the groups `kevin` belongs to and their IDs.

```
# groupadd designers
# groupmod -n web-designers designers
# usermod -a -G web-designers kevin
# id kevin
uid=1010(kevin) gid=1030(kevin)
groups=1030(kevin),1028(administrators),1029(developers),1031(web-designers)
```

- Remove only the `developers` group from the secondary groups of `kevin`.

```
# usermod -G administrators,web-designers kevin
# id kevin
uid=1010(kevin) gid=1030(kevin) groups=1030(kevin),1028(administrators),1031(web-designers)
```

The `usermod` command does not have an option to remove only one group; therefore, you need to specify all the secondary groups to which the user belongs.

- Set the password for the `kevin` user account.

```
# passwd kevin
Changing password for user kevin.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

6. Using the `chage` command, first check the expiry date of the `kevin` user account and then change it to December 31st 2022. What other command can you use to change the expiration date of a user account?

```
# chage -l kevin | grep "Account expires"
Account expires      : never
# chage -E 2022-12-31 kevin
# chage -l kevin | grep "Account expires"
Account expires      : dec 31, 2022
```

The `usermod` command with the `-e` option is equivalent to `chage -E`.

7. Add a new user account named `emma` with UID 1050 and set `administrators` as its primary group and `developers` and `web-designers` as its secondary groups.

```
# useradd -u 1050 -g administrators -G developers,web-designers emma
# id emma
uid=1050(emma) gid=1028(administrators)
groups=1028(administrators),1029(developers),1031(web-designers)
```

8. Change the login shell of `emma` to `/bin/sh`.

```
# usermod -s /bin/sh emma
```

9. Delete the `emma` and `kevin` user accounts and the `administrators`, `developers` and `web-designers` groups.

```
# userdel -r emma
# userdel -r kevin
# groupdel administrators
# groupdel developers
# groupdel web-designers
```



## 107.1 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	107 Administrative Tasks
<b>Objective:</b>	107.1 Manage user and group accounts and related system files
<b>Lesson:</b>	2 of 2

### Introduction

The command line tools discussed in the previous lesson and the graphical applications provided by each distribution that perform the same tasks update a series of files that store information about users and groups.

These files are located under the `/etc/` directory and are:

#### `/etc/passwd`

A file of seven colon-delimited fields containing basic information about users.

#### `/etc/group`

A file of four colon-delimited fields containing basic information about groups.

#### `/etc/shadow`

A file of nine colon-delimited fields containing encrypted user passwords.

## **/etc/gshadow**

A file of four colon-delimited fields file containing encrypted group passwords.

Although these four files are in plain text, they should not be edited directly, but always through the tools provided by the distribution you are using.

## **/etc/passwd**

This is a world-readable file that contains a list of users, each on a separate line. Each line consists of seven colon-delimited fields:

### **Username**

The name used when the user logs into the system.

### **Password**

The encrypted password (or an `x` if shadow passwords are used).

### **User ID (UID)**

The ID number assigned to the user in the system.

### **Group ID (GID)**

The primary group number of the user in the system.

### **GECOS**

An optional comment field, which is used to add extra information about the user (such as the full name). The field can contain multiple comma-separated entries.

### **Home Directory**

The absolute path of the user's home directory.

### **Shell**

The absolute path of the program that is automatically launched when the user logs into the system (usually an interactive shell such as `/bin/bash`).

## **/etc/group**

This is a world-readable file that contains a list of groups, each on a separate line. Each line consists of four colon-delimited fields:

**Group Name**

The name of the group.

**Group Password**

The encrypted password of the group (or an `x` if shadow passwords are used).

**Group ID (GID)**

The ID number assigned to the group in the system.

**Member List**

A comma-delimited list of users belonging to the group, except those for whom this is the primary group.

**/etc/shadow**

This is a file readable only by root and by users with root privileges that contains encrypted user passwords, each on a separate line. Each line consists of nine colon-delimited fields:

**Username**

The name used when the user logs into the system.

**Encrypted Password**

The encrypted password of the user (if the value starts with `!`, the account is locked).

**Date of Last Password Change**

The date of the last password change, as number of days since 01/01/1970 (a value of 0 means that the user must change the password when they next login).

**Minimum Password Age**

The minimum number of days, after a password change, which must pass before the user will be allowed to change the password again.

**Maximum Password Age**

The maximum number of days that must pass before a password change is required.

**Password Warning Period**

The number of days, before the password expires, during which the user is warned that the password must be changed.

## Password Inactivity Period

The number of days after a password expires during which the user should update the password. After this period, if the user does not change the password, the account will be disabled.

## Account Expiration Date

The date, expressed as the number of days since 01/01/1970, in which the user account will be disabled (an empty field means that the user account will never expire).

## A reserved field

A field that is reserved for future use.

## /etc/gshadow

This is a file readable only by root and by users with root privileges that contains encrypted group passwords, each on a separate line. Each line consists of four colon-delimited fields:

### Group Name

The name of the group.

### Encrypted Password

The encrypted password for the group (it is used when a user, who is not a member of the group, wants to join the group using the `newgrp` command — if the password starts with `!`, no one is allowed to access the group with `newgrp`).

### Group Administrators

A comma-delimited list of the administrators of the group (they can change the password of the group and can add or remove group members with the `gpasswd` command).

### Group Members

A comma-delimited list of the members of the group.

## Filter the Password and Group Databases

Very often it may be necessary to review information on users and groups stored in these four files and search for specific records. To perform this task, you can use the `grep` command or alternatively concatenate `cat` and `grep`.

```
# grep emma /etc/passwd
emma:x:1020:1020:User Emma:/home/emma:/bin/bash
```

```
# cat /etc/group | grep db-admin
db-admin:x:1050:grace,frank
```

Another way to access these databases is to use the `getent` command. In general, this command displays entries from databases supported by the *Name Service Switch* (NSS) libraries and requires the name of the database and a lookup key. If no key argument is provided, all entries in the specified database are displayed (unless the database does not support enumeration). Otherwise, if one or more key arguments are provided, the database is filtered accordingly.

```
# getent passwd emma
emma:x:1020:1020:User Emma:/home/emma:/bin/bash
# getent group db-admin
db-admin:x:1050:grace,frank
```

The `getent` command does not require root authority; you just need to be able to read the database from which you want to retrieve records.

**NOTE** Remember that `getent` can only access databases configured in the `/etc/nsswitch.conf` file.

# Guided Exercises

1. Observe the following output and answer the following questions:

```
# cat /etc/passwd | grep '\(root\|mail\|catherine\|kevin\)'
root:x:0:0:root:/root:/bin/bash
mail:x:8:8:mail:/var/spool/mail:/sbin/nologin
catherine:x:1030:1025:User Chaterine:/home/catherine:/bin/bash
kevin:x:1040:1015:User Kevin:/home/kevin:/bin/bash
# cat /etc/group | grep '\(root\|mail\|db-admin\|app-developer\)'
root:x:0:
mail:x:8:
db-admin:x:1015:emma,grace
app-developer:x:1016:catherine,dave,christian
# cat /etc/shadow | grep '\(root\|mail\|catherine\|kevin\)'
root:$6$1u36Ipok$1jt8ooPMLewAhkQPf.1YgGopAB.jClT06ljsdczvxkLPkpi/amgp.zyfAN680zrLLp2avvpd
KA0llpssdfcPppOp:18015:0:99999:7:::
mail:*:18015:0:99999:7:::
catherine:$6$ABCD25jlld14hpPthEFGnnsEwW1234yioMpliABCdef1f3478kAfhhAfgbAMjY1/BAeeAsl/FeE
dddKd12345g6kPACcik:18015:20:90:5:::
kevin:$6$DEFGabc123WrLp223fsvp0ddx3dbA7pPPc4LMaa123u6Lp02Lpvm123456pyphhh5ps012vbArL245.P
R1345kkA3Gas12P:18015:0:60:7:2:::
# cat /etc/gshadow | grep '\(root\|mail\|db-admin\|app-developer\)'
root*::
mail*::
db-admin!:emma:emma,grace
app-developer!:::catherine,dave,christian
```

- What is the User ID (UID) and the Group ID (GID) of root and catherine?

- What is the name of the primary group of kevin? Are there other members in this group?

- Which shell is set for mail? What does it mean?

- Who are the members of the app-developer group? Which of these members are group administrators and which are ordinary members?



- What is the minimum password lifetime for `catherine`? And what is the maximum password lifetime?

- What is the password inactivity period for `kevin`?

2. By convention, which IDs are assigned to system accounts and which to ordinary users?

3. How do you find out if a user account, which was previously able to access the system, is now locked? Assume your system uses shadow passwords.

## Explorational Exercises

1. Create a user account named `christian` using the `useradd -m` command and identify its User ID (UID), Group ID (GID) and shell.

2. Identify the name of the primary group of `christian`. What can you deduce?

3. Using the `getent` command, review password aging information for the `christian` user account.

4. Add the `editor` group to the secondary groups of `christian`. Assume that this group already contains `emma`, `dave` and `frank` as ordinary members. How can you verify that there are no administrators for this group?

5. Run the `ls -l /etc/passwd /etc/group /etc/shadow /etc/gshadow` command and describe the output that it gives you in terms of file permissions. Which of these four files are shadowed for security reasons? Assume your system uses shadow passwords.

# Summary

In this lesson you learned:

- The location of files that store information about users and groups.
- Manage user and group information stored in password and group databases.
- Retrieve information from password and group databases.

The following files and commands were discussed in this lesson:

## **/etc/passwd**

The file containing basic information about users.

## **/etc/group**

The file containing basic information about groups.

## **/etc/shadow**

The file containing encrypted user passwords.

## **/etc/gshadow**

The file containing encrypted group passwords.

## **getent**

Filter the password and group databases.

# Answers to Guided Exercises

1. Observe the following output and answer the following questions:

```
# cat /etc/passwd | grep '\(root\|mail\|catherine\|kevin\)'
root:x:0:0:root:/root:/bin/bash
mail:x:8:8:mail:/var/spool/mail:/sbin/nologin
catherine:x:1030:1025:User Chaterine:/home/catherine:/bin/bash
kevin:x:1040:1015:User Kevin:/home/kevin:/bin/bash
# cat /etc/group | grep '\(root\|mail\|db-admin\|app-developer\)'
root:x:0:
mail:x:8:
db-admin:x:1015:emma,grace
app-developer:x:1016:catherine,dave,christian
# cat /etc/shadow | grep '\(root\|mail\|catherine\|kevin\)'
root:$6$1u36Ipok$1jt8ooPMLewAhkQPf.1YgGopAB.jClT06ljsdczvkLPkpi/amgp.zyfAN680zrLLp2avvpd
KA0llpssdfcPppOp:18015:0:99999:7:::
mail:*:18015:0:99999:7:::
catherine:$6$ABCD25jlld14hpPthEFGnnsEwW1234yioMpliABCdef1f3478kAfhhAfgbAMjY1/BAeeAsl/FeE
dddKd12345g6kPACcik:18015:20:90:5:::
kevin:$6$DEFGabc123WrLp223fsvp0ddx3dbA7pPPc4LMaa123u6Lp02Lpvm123456pyphhh5ps012vbArL245.P
R1345kkA3Gas12P:18015:0:60:7:2:::
# cat /etc/gshadow | grep '\(root\|mail\|db-admin\|app-developer\)'
root*::
mail*::
db-admin!:emma:emma,grace
app-developer!:catherine,dave,christian
```

- What is the User ID (UID) and the Group ID (GID) of `root` and `catherine`?

The UID and GID of `root` are 0 and 0, while the UID and GID of `catherine` are 1030 and 1025.

- What is the name of the primary group of `kevin`? Are there other members in this group?

The group name is `db-admin`. Also `emma` and `grace` are in this group.

- Which shell is set for `mail`? What does it mean?

`mail` is a system user account and its shell is `/sbin/nologin`. In fact, system user accounts such as `mail`, `ftp`, `news` and `daemon` are used to perform administrative tasks and therefore normal login should be prevented for these accounts. This is why the shell is usually set to

`/sbin/nologin` or `/bin/false`.

- What are the members of the `app-developer` group? Which of these are group administrators and which are ordinary members?

The members are `catherine`, `dave` and `christian` and they are all ordinary members.

- What is the minimum password lifetime for `catherine`? And what is the maximum password lifetime?

The minimum password lifetime is 20 days, while the maximum password lifetime is 90 days.

- What is the password inactivity period for `kevin`?

The password inactivity period is 2 days. During this period `kevin` should update the password, otherwise the account will be disabled.

2. By convention, which IDs are assigned to system accounts and which to ordinary users?

System accounts usually have UIDs less than 100 or between 500 and 1000, while ordinary users have UIDs starting at 1000 although some legacy systems may start numbering at 500. The `root` user has UID 0. Remember that the `UID_MIN` and `UID_MAX` values in `/etc/login.defs` define the range of UIDs used for the creation of ordinary users. From the standpoint of LPI Linux Essentials and LPIC-1, system accounts have UIDs less than 1000 and ordinary users have UIDs greater than 1000.

3. How do you find out if a user account, which was previously able to access the system, is now locked? Assume your system uses shadow passwords.

When shadow passwords are used, the second field in `/etc/passwd` contains the `x` character for each user account, because the encrypted user passwords are stored in `/etc/shadow`. In particular, the encrypted password of a user account is stored in the second field of this file and, if it starts with an exclamation mark, the account is locked.

## Answers to Explorational Exercises

1. Create a user account named `christian` using the `useradd -m` command and identify its User ID (UID), Group ID (GID) and shell.

```
# useradd -m christian
# cat /etc/passwd | grep christian
christian:x:1050:1060:./home/christian:/bin/bash
```

The UID and GID of `christian` are 1050 and 1060 respectively (the third and fourth fields in `/etc/passwd`). `/bin/bash` is the shell set for this user account (the seventh field in `/etc/passwd`).

2. Identify the name of the primary group of `christian`. What can you deduce?

```
# cat /etc/group | grep 1060
christian:x:1060:
```

The name of the primary group of `christian` is `christian` (the first field in `/etc/group`). Therefore, `USERGROUPS_ENAB` in `/etc/login.defs` is set to `yes` so that `useradd` creates by default a group with the same name of the user account.

3. Using the `getent` command, review password aging information for the `christian` user account.

```
# getent shadow christian
christian:!:18015:0:99999:7:::
```

The `christian` user account does not have the password set and is now locked (the second field in `/etc/shadow` contains an exclamation mark). There is no minimum and maximum password age for this user account (the fourth and fifth fields in `/etc/shadow` are set to 0 and 99999 days), while the password warning period is set to 7 days (the sixth field in `/etc/shadow`). Finally, there is no inactivity period (the seventh field in `/etc/shadow`) and the account never expires (the eighth field in `/etc/shadow`).

4. Add the `editor` group to the secondary groups of `christian`. Assume that this group already contains `emma`, `dave` and `frank` as ordinary members. How can you verify that there are no administrators for this group?

```
# cat /etc/group | grep editor
editor:x:1100:emma,dave,frank
# usermod -a -G editor christian
# cat /etc/group | grep editor
editor:x:1100:emma,dave,frank,christian
# cat /etc/gshadow | grep editor
editor!:::emma,dave,frank,christian
```

The third and fourth fields in `/etc/gshadow` contain administrators and ordinary members for the specified group. Therefore, since the third field is empty for `editor`, there are no administrators for this group (`emma`, `dave`, `frank` and `christian` are all ordinary members).

5. Run the `ls -l /etc/passwd /etc/group /etc/shadow /etc/gshadow` command and describe the output that it gives you in terms of file permissions. Which of these four files are shadowed for security reasons? Assume your system uses shadow passwords.

```
# ls -l /etc/passwd /etc/group /etc/shadow /etc/gshadow
-rw-r--r-- 1 root root 853 mag 1 08:00 /etc/group
-rw-r----- 1 root shadow 1203 mag 1 08:00 /etc/gshadow
-rw-r--r-- 1 root root 1354 mag 1 08:00 /etc/passwd
-rw-r----- 1 root shadow 1563 mag 1 08:00 /etc/shadow
```

The `/etc/passwd` and `/etc/group` files are world readable and are shadowed for security reasons. When shadow passwords are used, you can see an `x` in the second field of these files, because the encrypted passwords for users and groups are stored in `/etc/shadow` and `/etc/gshadow`, which are readable only by root and, in my system, even by members belonging to the `shadow` group.



## 107.2 Automate system administration tasks by scheduling jobs

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 107.2](#)

### Weight

4

### Key knowledge areas

- Manage cron and at jobs.
- Configure user access to cron and at services.
- Understand systemd timer units.

### Partial list of the used files, terms and utilities

- `/etc/cron.{d,daily,hourly,monthly,weekly}/`
- `/etc/at.deny`
- `/etc/at.allow`
- `/etc/crontab`
- `/etc/cron.allow`
- `/etc/cron.deny`
- `/var/spool/cron/`
- `crontab`
- `at`
- `atq`
- `atrm`



- `systemctl`
- `systemd-run`



## 107.2 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	107 Administrative Tasks
<b>Objective:</b>	107.2 Automate system administration tasks by scheduling jobs
<b>Lesson:</b>	1 of 2

### Introduction

One of the most important tasks of a good system administrator is to schedule jobs that must be executed on a regular basis. For example, an administrator can create and automate jobs for backups, system upgrades and to perform many other repetitive activities. To do this you can use the `cron` facility, which is useful to automate periodic job scheduling.

### Schedule Jobs with Cron

In Linux, `cron` is a daemon that runs continuously and wakes up every minute to check a set of tables to find tasks to execute. These tables are known as *crontabs* and contain the so-called *cron jobs*. Cron is suitable for servers and systems that are constantly powered on, because each cron job is executed only if the system is running at the scheduled time. It can be used by ordinary users, each of whom has their own *crontab*, as well as the root user who manages the system *crontabs*.

#### NOTE

In Linux there is also the `anacron` facility, which is suitable for systems that can be powered off (such as desktops or laptops). It can only be used by root. If the

machine is off when the `anacron` jobs must be executed, they will run the next time the machine is powered on. `anacron` is out of scope for the LPIC-1 certification.

## User Crontabs

*User crontabs* are text files that manage the scheduling of user-defined cron jobs. They are always named after the user account that created them, but the location of these files depends on the distribution used (generally a subdirectory of `/var/spool/cron`).

Each line in a user crontab contains six fields separated by a space:

- The minute of the hour (0-59).
- The hour of the day (0-23).
- The day of the month (1-31).
- The month of the year (1-12).
- The day of the week (0-7 with Sunday=0 or Sunday=7).
- The command to run.

For the month of the year and the day of the week you can use the first three letters of the name instead of the corresponding number.

The first five fields indicate when to execute the command that is specified in the sixth field, and they can contain one or more values. In particular, you can specify multiple values using:

### \* (asterisk)

Refers to any value.

### , (comma)

Specifies a list of possible values.

### - (dash)

Specifies a range of possible values.

### / (slash)

Specifies stepped values.

Many distributions include the `/etc/crontab` file which can be used as a reference for the layout of a `cron` file. Here is an example `/etc/crontab` file from a Debian installation:

```

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed

```

## System Crontabs

*System crontabs* are text files that manage the scheduling of system cron jobs and can only be edited by the root user. `/etc/crontab` and all files in the `/etc/cron.d` directory are system crontabs.

Most distributions also include the `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly` and `/etc/cron.monthly` directories that contain scripts to be run with the appropriate frequency. For example, if you want to run a script daily you can place it in `/etc/cron.daily`.

### WARNING

Some distributions use `/etc/cron.d/hourly`, `/etc/cron.d/daily`, `/etc/cron.d/weekly` and `/etc/cron.d/monthly`. Always remember to check for the correct directories in which to place scripts you would like cron to run.

The syntax of system crontabs is similar to that of user crontabs, however it also requires an additional mandatory field that specifies which user will run the cron job. Therefore, each line in a system crontab contains seven fields separated by a space:

- The minute of the hour (0-59).
- The hour of the day (0-23).
- The day of the month (1-31).
- The month of the year (1-12).
- The day of the week (0-7 with Sunday=0 or Sunday=7).
- The name of the user account to be used when executing the command.
- The command to run.

As for user crontabs, you can specify multiple values for the time fields using the `*`, `,`, `-` and `/` operators. You may also indicate the month of the year and the day of the week with the first three letters of the name instead of the corresponding number.

## Particular Time Specifications

When editing crontab files, you can also use special shortcuts in the first five columns instead of the time specifications:

### **@reboot**

Run the specified task once after reboot.

### **@hourly**

Run the specified task once an hour at the beginning of the hour.

### **@daily (or @midnight)**

Run the specified task once a day at midnight.

### **@weekly**

Run the specified task once a week at midnight on Sunday.

### **@monthly**

Run the specified task once a month at midnight on the first day of the month.

### **@yearly (or @annually)**

Run the specified task once a year at midnight on the 1st of January.

## Crontab Variables

Within a crontab file, there are sometimes variable assignments defined before the scheduled tasks are declared. The environment variables commonly set are:

### **HOME**

The directory where `cron` invokes the commands (by default the user's home directory).

### **MAILTO**

The name of the user or the address to which the standard output and error is mailed (by default the crontab owner). Multiple comma-separated values are also allowed and an empty value indicates that no mail should be sent.

## PATH

The path where commands can be found.

## SHELL

The shell to use (by default `/bin/sh`).

## Creating User Cron Jobs

The `crontab` command is used to maintain crontab files for individual users. In particular, you can run the `crontab -e` command to edit your own crontab file or to create one if it doesn't already exist.

```
$ crontab -e
no crontab for frank - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/ed
 2. /bin/nano      < ---- easiest
 3. /usr/bin/emacs24
 4. /usr/bin/vim.tiny

Choose 1-4 [2]:
```

By default, the `crontab` command opens the editor specified by the `VISUAL` or `EDITOR` environment variables so you can start editing your crontab file with the preferred editor. Some distributions, as shown in the example above, allow you to choose the editor from a list when `crontab` is run for the first time.

If you want to run the `foo.sh` script located in your home directory every day at 10:00 am, you can add the following line to your crontab file:

```
0 10 * * * /home/frank/foo.sh
```

Consider the following sample crontab entries:

```
0,15,30,45 08 * * 2 /home/frank/bar.sh
30 20 1-15 1,6 1-5 /home/frank/foobar.sh
```

In the first line the `bar.sh` script is executed every Tuesday at 08:00 am, at 08:15 am, at 08:30 am

and at 08:45 am. On the second line the `foobar.sh` script is executed at 08:30 pm from Monday to Friday for the first fifteen days of January and June.

**WARNING**

Although crontab files can be edited manually, it is always recommended to use the `crontab` command. The permissions on the crontab files usually only allow them to be edited via the `crontab` command.

In addition to the `-e` option mentioned above, the `crontab` command has other useful options:

**-l**

Display the current crontab on standard output.

**-r**

Remove the current crontab.

**-u**

Specify the name of the user whose crontab needs to be modified. This option requires root privileges and allows the root user to edit user crontab files.

## Creating System Cron Jobs

Unlike user crontabs, system crontabs are updated using an editor: therefore, you do not need to run the `crontab` command to edit `/etc/crontab` and the files in `/etc/cron.d`. Remember that when editing system crontabs, you must specify the account that will be used to run the cron job (usually the root user).

For example, if you want to run the `barfoo.sh` script located in the `/root` directory every day at 01:30 am, you can open `/etc/crontab` with your preferred editor and add the following line:

```
30 01 * * * root /root/barfoo.sh >>/root/output.log 2>>/root/error.log
```

In the above example, output of the job is appended to `/root/output.log`, while errors are appended to `/root/error.log`.

**WARNING**

Unless output is redirected to a file as in the example above (or the `MAILTO` variable is set to a blank value), all output from a cron job will be sent to the user via e-mail. A common practice is to redirect standard output to `/dev/null` (or to a file for later review if necessary) and to not redirect standard error. This way the user will be notified immediately by e-mail of any errors.

## Configure Access to Job Scheduling

In Linux the `/etc/cron.allow` and `/etc/cron.deny` files are used to set `crontab` restrictions. In particular, they are used to allow or disallow the scheduling of cron jobs for different users. If `/etc/cron.allow` exists, only non-root users listed within it can schedule cron jobs using the `crontab` command. If `/etc/cron.allow` does not exist but `/etc/cron.deny` exists, only non-root users listed within this file cannot schedule cron jobs using the `crontab` command (in this case an empty `/etc/cron.deny` means that each user is allowed to schedule cron jobs with `crontab`). If neither of these files exist, the user's access to cron job scheduling depends on the distribution used.

**NOTE** The `/etc/cron.allow` and `/etc/cron.deny` files contain of a list of usernames, each on a separate line.

## An Alternative to Cron

Using `systemd` as the system and service manager, you can set *timers* as an alternative to `cron` to schedule your tasks. Timers are `systemd` unit files identified by the `.timer` suffix, and for each of these there must be a corresponding unit file which describes the unit to be activated when the timer elapses. By default, a `timer` activates a service with the same name, except for the suffix.

A timer includes a `[Timer]` section that specifies when scheduled jobs should run. Specifically, you can use the `OnCalendar=` option to define *real-time timers* which work in the same way as cron jobs (they are based on calendar event expressions). The `OnCalendar=` option requires the following syntax:

```
DayOfWeek Year-Month-Day Hour:Minute:Second
```

with `DayOfWeek` being optional. The `*`, `/` and `,` operators have the same meaning as those used for cron jobs, while you can use `..` between two values to indicate a contiguous range. For `DayOfWeek` specification, you can use the first three letters of the name or the full name.

**NOTE** You can also define *monotonic timers* that activate after some time has elapsed from a specific start point (for example, when the machine was booted up or when the timer itself is activated).

For example, if you want to run the service named `/etc/systemd/system/foobar.service` at 05:30 on the first Monday of each month, you can add the following lines in the corresponding `/etc/systemd/system/foobar.timer` unit file.



```
[Unit]
Description=Run the foobar service

[Timer]
OnCalendar=Mon *-*..7 05:30:00
Persistent=true

[Install]
WantedBy=timers.target
```

Once you have created the new timer, you can enable it and start it by running the following commands as root:

```
# systemctl enable foobar.timer
# systemctl start foobar.timer
```

You can change the frequency of your scheduled job, modifying the `OnCalendar` value and then typing the `systemctl daemon-reload` command.

Finally, if you want to view the list of active timers sorted by the time they elapse next, you can use the `systemctl list-timers` command. You can add the `--all` option to see the inactive timer units as well.

#### NOTE

Remember that timers are logged to the `systemd` journal and you can review the logs of the different units using the `journalctl` command. Also remember that if you are acting as an ordinary user, you need to use the `--user` option of the `systemctl` and `journalctl` commands.

Instead of the longer normalized form mentioned above, you can use some special expressions which describe particular frequencies for job execution:

#### hourly

Run the specified task once an hour at the beginning of the hour.

#### daily

Run the specified task once a day at midnight.

#### weekly

Run the specified task once a week at midnight on Monday.

### **monthly**

Run the specified task once a month at midnight on the first day of the month.

### **yearly**

Run the specified task once a year at midnight on the first day of January.

You can see the manual pages for the full list of time and date specifications at `systemd.timer(5)`.

## Guided Exercises

1. For each of the following `crontab` shortcuts, indicate the corresponding time specification (i.e. the first five columns in a user `crontab` file):

@hourly	
@daily	
@weekly	
@monthly	
@annually	

2. For each of the following `OnCalendar` shortcuts, indicate the corresponding time specification (the longer normalized form):

hourly	
daily	
weekly	
monthly	
yearly	

3. Explain the meaning of the following time specifications found in a `crontab` file:

30 13 * * 1-5	
00 09-18 * * *	
30 08 1 1 *	
0,20,40 11 * * Sun	
00 09 10-20 1-3 *	
*/20 * * * *	

4. Explain the meaning of the following time specifications used in the `OnCalendar` option of a timer file:

*-*-* 08:30:00	
Sat,Sun *-*-* 05:00:00	

*-*01 13:15,30,45:00	
Fri *-09..12-* 16:20:00	
Mon,Tue *-*-1,15 08:30:00	
*-*-* *:00/05:00	

## Explorational Exercises

1. Assuming that you are authorized to schedule jobs with `crontab` as an ordinary user, what command would you use to create your own crontab file?

2. Create a simple scheduled job that executes the `date` command every Friday at 01:00 pm. Where can you see the output of this job?

3. Create another scheduled job that executes the `foobar.sh` script every minute, redirecting the output to the `output.log` file in your home directory so that only standard error is sent to you by e-mail.

4. Look at the `crontab` entry of the newly created scheduled job. Why is it not necessary to specify the absolute path of the file in which the standard output is saved? And why can you use the `./foobar.sh` command to execute the script?

5. Edit the previous `crontab` entry by removing the output redirection and disable the first cron job you have created.

6. How can you send the output and errors of your scheduled job to the `emma` user account via e-mail? And how can you avoid sending the standard output and error via e-mail?

7. Execute the command `ls -l /usr/bin/crontab`. Which special bit is set and what is its meaning?

# Summary

In this lesson, you learned:

- Use `cron` to run jobs at regular intervals.
- Manage cron jobs.
- Configure user access to cron job scheduling.
- Understand the role of systemd timer units as an alternative to `cron`.

The following commands and files were discussed in this lesson:

## `crontab`

Maintain `crontab` files for individual users.

## `/etc/cron.allow` and `/etc/cron.deny`

Particular files used to set `crontab` restrictions.

## `/etc/crontab`

System `crontab` file.

## `/etc/cron.d`

The directory that contains system `crontab` files.

## `systemctl`

Control the systemd system and service manager. In relation to timers, it can be used to enable and start them.

## Answers to Guided Exercises

1. For each of the following `crontab` shortcuts, indicate the corresponding time specification (the first five columns in a `crontab` file):

@hourly	0 * * * *
@daily	0 0 * * *
@weekly	0 0 * * 0
@monthly	0 0 1 * *
@annually	0 0 1 1 *

2. For each of the following `OnCalendar` shortcuts, indicate the corresponding time specification (the longer normalized form):

hourly	*-*-* *:00:00
daily	*-*-* 00:00:00
weekly	Mon *-*-* 00:00:00
monthly	*-*-01 00:00:00
yearly	*-01-01 00:00:00

3. Explain the meaning of the following time specifications for a `crontab` file:

30 13 * * 1-5	At 01:30 pm every day of the week from Monday to Friday
00 09-18 * * *	Every day and every hour from 09 am to 06 pm
30 08 1 1 *	At 08:30 am on the first day of January
0,20,40 11 * * Sun	Every Sunday at 11:00 am, 11:20 am and 11:40 am
00 09 10-20 1-3 *	At 09:00 am from the 10th to the 20th of January, February and March
*/20 * * * *	Every twenty minutes

4. Explain the meaning of the following time specifications used in the `OnCalendar` option of a timer file:

*-*-* 08:30:00	Every day at 08:30 am
Sat,Sun *--* 05:00:00	At 05:00 am on Saturday and Sunday
*--01 13:15,30,45:00	At 01:15 pm, 01:30 pm and 01:45 pm on the first day of the month
Fri *-09..12-* 16:20:00	At 04:20 pm every Friday in September, October, November and December
Mon,Tue *-*-1,15 08:30:00	At 08.30 am on the first or fifteenth day of each month only if the day is a Monday or Tuesday
*-*-* *:00/05:00	Every five minutes



## Answers to Explorational Exercises

1. Assuming that you are authorized to schedule jobs with `cron` as an ordinary user, what command would you use to create your own crontab file?

```
dave@hostname ~ $ crontab -e
no crontab for dave - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/ed
 2. /bin/nano      < ---- easiest
 3. /usr/bin/emacs24
 4. /usr/bin/vim.tiny

Choose 1-4 [2]:
```

2. Create a simple scheduled job that executes the `date` command every Friday at 01:00 pm. Where can you see the output of this job?

```
00 13 * * 5 date
```

The output is mailed to the user; to view it, you can use the `mail` command.

3. Create another scheduled job that executes the `foobar.sh` script every minute, redirecting the output to the `output.log` file in your home directory so that only standard error is sent to you by e-mail.

```
*/* * * * * ./foobar.sh >> output.log
```

4. Look at the `crontab` entry of the newly created scheduled job. Why is it not necessary to specify the absolute path of the file in which the standard output is saved? And why can you use the `./foobar.sh` command to execute the script?

`cron` invokes the commands from the user's home directory, unless another location is specified by the `HOME` environment variable within the `crontab` file. For this reason, you can use the relative path of the output file and run the script with `./foobar.sh`.

5. Edit the previous `crontab` entry by removing the output redirection and disable the first cron job you have created.

```
#00 13 * * 5 date
*/1 * * * * ./foobar.sh
```

To disable a cron job, you can simply comment the corresponding line within the `crontab` file.

6. How can you send the output and errors of your scheduled job to the `emma` user account via e-mail? And how can you avoid sending the standard output and error via e-mail?

To send the standard output and error to `emma`, you must set the `MAILTO` environment variable in your `crontab` file as follows:

```
MAILTO="emma"
```

To tell `cron` that no mail should be sent, you can assign an empty value to the `MAILTO` environment variable.

```
MAILTO=""
```

7. Execute the command `ls -l /usr/bin/crontab`. Which special bit is set and what is its meaning?

```
$ ls -l /usr/bin/crontab
-rwxr-sr-x 1 root crontab 25104 feb 10 2015 /usr/bin/crontab
```

The `crontab` command has the SGID bit set (the `s` character instead of the executable flag for the group), which means that it is executed with the privileges of the group (thus `crontab`). This is why ordinary users can edit their `crontab` file using the `crontab` command. Note that many distributions have file permissions set such that `crontab` files can only be edited via the `crontab` command.



## 107.2 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	107 Administrative Tasks
<b>Objective:</b>	107.2 Automate system administration tasks by scheduling jobs
<b>Lesson:</b>	2 of 2

### Introduction

As you learned in the previous lesson, you can schedule regular jobs using cron or systemd timers, but sometimes you may need to run a job at a specific time in the future only once. To do this, you can use another powerful utility: the `at` command.

### Schedule Jobs with `at`

The `at` command is used for one-time task scheduling and only requires that you specify when the job should be run in the future. After entering `at` on the command line followed by the time specification, you will enter the `at` prompt where you can define the commands to be executed. You can exit the prompt with the `Ctrl + D` key-sequence.

```
$ at now +5 minutes
warning: commands will be executed using /bin/sh
at> date
at> Ctrl+D
```

```
job 12 at Sat Sep 14 09:15:00 2019
```

The `at` job in the above example simply executes the `date` command after five minutes. Similar to `cron`, the standard output and error is sent to you via e-mail. Note that the `atd` daemon will need to be running on the system in order for you to use `at` job scheduling.

**NOTE**

In Linux, the `batch` command is similar to `at`, however `batch` jobs are executed only when the system load is low enough to allow it.

The most important options which apply to the `at` command are:

**-c**

Print the commands of a specific job ID to the standard output.

**-d**

Delete jobs based on their job ID. It is an alias for `atrm`.

**-f**

Read the job from a file instead of the standard input.

**-l**

List the pending jobs of the user. If the user is root, all jobs of all users are listed. It is an alias for `atq`.

**-m**

Send mail to the user at the end of the job even if there was no output.

**-q**

Specify a queue in the form of a single letter from `a` to `z` and from `A` to `Z` (by default `a` for `at` and `b` for `batch`). Jobs in the queues with the highest letters are executed with increased niceness. Jobs submitted to a queue with a capital letter are treated as `batch` jobs.

**-v**

Show the time at which the job will run before reading the job.

## List Scheduled Jobs with `atq`

Now let us schedule two more `at` jobs: the first executes the `foo.sh` script at 09:30 am, while the second executes the `bar.sh` script after one hour.

```
$ at 09:30 AM
warning: commands will be executed using /bin/sh
at> ./foo.sh
at> Ctrl+D
job 13 at Sat Sep 14 09:30:00 2019
$ at now +2 hours
warning: commands will be executed using /bin/sh
at> ./bar.sh
at> Ctrl+D
job 14 at Sat Sep 14 11:10:00 2019
```

To list your pending jobs, you can use the `atq` command which shows the following information for each job: job ID, job execution date, job execution time, queue, and username.

```
$ atq
14      Sat Sep 14 11:10:00 2019 a frank
13      Sat Sep 14 09:30:00 2019 a frank
12      Sat Sep 14 09:15:00 2019 a frank
```

Remember that the `at -l` command is an alias for `atq`.

**NOTE** If you run `atq` as root, it will display the queued jobs for all users.

## Delete Jobs with `atrm`

If you want to delete an `at` job, you can use the `atrm` command followed by the job ID. For example, to delete the job with ID 14, you can run the following:

```
$ atrm 14
```

You can delete multiple jobs with `atrm` by specifying multiple IDs separated by spaces. Remember that the `at -d` command is an alias for `atrm`.

**NOTE** If you run `atrm` as root you can delete the jobs of all users.

## Configure Access to Job Scheduling

Authorization for ordinary users to schedule `at` jobs is determined by the `/etc/at.allow` and `/etc/at.deny` files. If `/etc/at.allow` exists, only non-root users listed within it can schedule `at` jobs. If `/etc/at.allow` does not exist but `/etc/at.deny` exists, only non-root users listed within

it cannot schedule at jobs (in this case an empty `/etc/at.deny` file means that each user is allowed to schedule at jobs). If neither of these files exist, the user's access to at job scheduling depends on the distribution used.

## Time Specifications

You can specify when to execute a particular at job using the form `HH:MM`, optionally followed by `AM` or `PM` in case of 12-hour format. If the specified time has already passed, the next day is assumed. If you want to schedule a particular date on which the job will run, you must add the date information after the time using one of the following forms: `month-name day-of-month`, `month-name day-of-month year`, `MMDDYY`, `MM/DD/YY`, `DD.MM.YY` and `YYYY-MM-DD`.

The following keywords are also accepted: `midnight`, `noon`, `teatime` (4 pm) and `now` followed by a plus sign (+) and a time period (minutes, hours, days and weeks). Finally, you can tell at to run the job today or tomorrow by suffixing the time with the words `today` or `tomorrow`. For example, you can use `at 07:15 AM Jan 01` to execute a job at 07:15 am on 01 January and `at now +5 minutes` to execute a job five minutes from now. You can read the `timespec` file under the `/usr/share` tree for more information about the exact definition of time specifications.

## An Alternative to at

Using `systemd` as the system and service manager, you can also schedule one-time tasks with the `systemd-run` command. It is typically used to create a transient timer unit so that a command will be executed at a specific time without the need to create a service file. For example, acting as root, you can run the `date` command at 11:30 AM on 2019/10/06 using the following:

```
# systemd-run --on-calendar='2019-10-06 11:30' date
```

If you want to run the `foo.sh` script, located in your current working directory, after two minutes you can use:

```
# systemd-run --on-active="2m" ./foo.sh
```

Consult the manual pages to learn all possible uses of `systemd-run` with `systemd-run(1)`.

### NOTE

Remember that timers are logged to the `systemd` journal and you can review the logs of the different units using the `journalctl` command. Also remember that if you are acting as an ordinary user, you need to use the `--user` option of the `systemd-run` and `journalctl` commands.

## Guided Exercises

1. For each of the following time specifications, indicate which is valid and which is invalid for `at`:

<code>at 08:30 AM next week</code>	
<code>at midday</code>	
<code>at 01-01-2020 07:30 PM</code>	
<code>at 21:50 01.01.20</code>	
<code>at now +4 days</code>	
<code>at 10:15 PM 31/03/2021</code>	
<code>at tomorrow 08:30 AM</code>	

2. Once you have scheduled a job with `at`, how can you review its commands?
3. Which commands can you use to review your pending `at` jobs? Which commands would you use to delete them?
4. With `systemd`, which command is used as an alternative to `at`?

## Explorational Exercises

1. Create an `at` job that runs the `foo.sh` script, located in your home directory, at 10:30 am on coming October 31st. Assume you are acting as an ordinary user.

2. Login to the system as another ordinary user and create another `at` job that runs the `bar.sh` script tomorrow at 10:00 am. Assume the script is located in the user's home directory.

3. Login to the system as another ordinary user and create another `at` job that runs the `foobar.sh` script just after 30 minutes. Assume the script is located in the user's home directory.

4. Now as root, run the `atq` command to review the scheduled `at` jobs of all users. What happens if an ordinary user executes this command?

5. As root, delete all these pending `at` jobs using a single command.

6. Run the `ls -l /usr/bin/at` command and examine its permissions.



# Summary

In this lesson, you learned:

- Use `at` to run one-time jobs at a specific time.
- Manage `at` jobs.
- Configure user access to `at` job scheduling.
- Use `systemd-run` as an alternative to `at`.

The following commands and files were discussed in this lesson:

## `at`

Execute commands at a specified time.

## `atq`

List the user's pending `at` jobs, unless the user is the superuser.

## `atrm`

Delete `at` jobs, identified by their job number.

## `/etc/at.allow` and `/etc/at.deny`

Particular files used to set `at` restrictions.

## `systemd-run`

Create and start a transient `timer` unit as an alternative to `at` for one-time scheduling.

## Answers to Guided Exercises

1. For each of the following time specifications, indicate which is valid and which is invalid for `at`:

<code>at 08:30 AM next week</code>	Valid
<code>at midday</code>	Invalid
<code>at 01-01-2020 07:30 PM</code>	Invalid
<code>at 21:50 01.01.20</code>	Valid
<code>at now +4 days</code>	Valid
<code>at 10:15 PM 31/03/2021</code>	Invalid
<code>at tomorrow 08:30 AM monotonic</code>	Invalid

2. Once you have scheduled a job with `at`, how can you review its commands?

You can use the `at -c` command followed by the ID of the job whose commands you want to review. Note that the output also contains most of the environment that was active at the time the job was scheduled. Remember that root can review the jobs of all users.

3. Which commands can you use to review your pending `at` jobs? Which commands would you use to delete them?

You can use the `at -l` command to review your pending jobs, and you can use the `at -d` command to delete your jobs. `at -l` is an alias for `atq` and `at -d` is an alias for `atrm`. Remember that root can list and delete the jobs of all users.

4. With `systemd`, which command is used as an alternative to `at`?

The `systemd-run` command can be used as an alternative to `at` to schedule one-time jobs. For example, you can use it to run commands at a specific time, defining a *calendar timer* or a *monotonic timer* relative to different starting points.

## Answers to Explorational Exercises

1. Create an `at` job that runs the `foo.sh` script, located in your home directory, at 10:30 am on coming October 31st. Assume you are acting as an ordinary user.

```
$ at 10:30 AM October 31
warning: commands will be executed using /bin/sh
at> ./foo.sh
at> Ctrl+D
job 50 at Thu Oct 31 10:30:00 2019
```

2. Login to the system as another ordinary user and create another `at` job that runs the `bar.sh` script tomorrow at 10:00 am. Assume the script is located in the user's home directory.

```
$ at 10:00 AM tomorrow
warning: commands will be executed using /bin/sh
at> ./bar.sh
at> Ctrl+D
job 51 at Sun Oct 6 10:00:00 2019
```

3. Login to the system as another ordinary user and create another `at` job that runs the `foobar.sh` script just after 30 minutes. Assume the script is located in the user's home directory.

```
$ at now +30 minutes
warning: commands will be executed using /bin/sh
at> ./foobar.sh
at> Ctrl+D
job 52 at Sat Oct 5 10:19:00 2019
```

4. Now as root, run the `atq` command to review the scheduled `at` jobs of all users. What happens if an ordinary user executes this command?

```
# atq
52      Sat Oct  5 10:19:00 2019 a dave
50      Thu Oct 31 10:30:00 2019 a frank
51      Sun Oct  6 10:00:00 2019 a emma
```

If you run the `atq` command as root, all pending `at` jobs of all users are listed. If you run it as

an ordinary user, only your own pending at jobs are listed.

5. As root, delete all of these pending at jobs using a single command.

```
# atrm 50 51 52
```

6. As root, run the `ls -l /usr/bin/at` command and examine its permissions.

```
# ls -l /usr/bin/at
-rwsr-sr-x 1 daemon daemon 43762 Dec  1 2015 /usr/bin/at
```

In this distribution, the `at` command has both the SUID (the `s` character instead of the executable flag for the owner) and SGID (the `s` character instead of the executable flag for the group) bits set, which means that it is executed with the privileges of the owner and group of the file (`daemon` for both). This is why ordinary users are able to schedule jobs with `at`.



## 107.3 Localisation and internationalisation

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 107.3](#)

### Weight

3

### Key knowledge areas

- Configure locale settings and environment variables.
- Configure timezone settings and environment variables.

### Partial list of the used files, terms and utilities

- `/etc/timezone`
- `/etc/localtime`
- `/usr/share/zoneinfo/`
- `LC_*`
- `LC_ALL`
- `LANG`
- `TZ`
- `/usr/bin/locale`
- `tzselect`
- `timedatectl`
- `date`
- `iconv`

- UTF-8
- ISO-8859
- ASCII
- Unicode



## 107.3 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	107 Administrative Tasks
<b>Objective:</b>	107.3 Localisation and internationalisation
<b>Lesson:</b>	1 of 1

### Introduction

All major Linux distributions can be configured to use custom localisation settings. These settings include region and language related definitions such as the time zone, interface language as well as character encoding can be modified during the installation of the operating system or anytime after that.

Applications rely on environment variables, system configuration files and commands to decide the proper time and language to use; hence most Linux distributions share a standardized way to adjust time and localisation settings. These adjustments are important not only to improve user experience, but also to ensure that the timing of system events—important, for example, for reporting security related issues—are correctly calculated.

To be able to represent any written text, regardless of the spoken language, modern operating systems need a reference *character encoding standard*, and Linux systems are no different. As computers are only able to deal with numbers, a text character is nothing more than a number associated with a graphic symbol. Distinct computer platforms may associate distinct number values to the same character, so a common character encoding standard is necessary to make them compatible. A text document created in one system will be readable in another system only

if both agree on the encoding format and on what number is associated to what character, or at least if they know how to convert between the two standards.

The heterogeneous nature of the localisation settings in Linux-based systems result in subtle differences between the distributions. Despite these differences, all distributions share the same basic tools and concepts to setup the internationalisation aspects of a system.

## Time Zones

Time zones are roughly proportional discrete bands of Earth's surface spanning the equivalent to one hour, that is, regions of the world experiencing the same hour of the day at any given time. Since there is not a single longitude that can be considered as the beginning of the day for the whole world, time zones are relative to the *prime meridian*, where the Earth's longitude angle is defined to be 0. The time at the prime meridian is called the *Coordinated Universal Time*, by convention abbreviated to UTC. Due to practical reasons, time zones do not follow the exact longitudinal distance from the reference point (the prime meridian). Instead, time zones are artificially adapted to follow the borders of countries or other significant subdivisions.

The political subdivisions are so relevant that time zones are named after some major geographical agent in that particular area, usually based on the name of a large country or city inside the zone. Yet, time zones are divided according to their time offset relative to UTC and this offset can also be used to indicate the zone in question. The time zone *GMT-5*, for example, indicates a region for which UTC time is five hours ahead, i.e. that region is 5 hours behind UTC. Likewise, the time zone *GMT+3* indicates a region for which UTC time is three hours behind. The term GMT — from *Greenwich Mean Time* — is used as a synonym for UTC in the offset based zone names.

A connected machine can be accessed from different parts of the world, so it is a good practice to set the hardware clock to UTC (the GMT+0 time zone) and leave the choice of the time zone to each particular case. Cloud services, for example, are commonly configured to use UTC, as it can help to mitigate occasional inconsistencies between local time and time at clients or at other servers. In contrast, users who open a remote session on the server may want to use their local time zone. Thus, it will be up to the operating system to set up the correct time zone according to each case.

In addition to the current date and time, command `date` will also print the currently configured time zone:

```
$ date
Mon Oct 21 10:45:21 -03 2019
```



The offset relative to UTC is given by the `-03` value, meaning that the displayed time is three hours less than UTC. Therefore, UTC time is three hours ahead, making *GMT-3* the corresponding time zone for the given time. Command `timedatectl`, which is available in distributions using `systemd`, shows more details about the system time and date:

```
$ timedatectl
           Local time: Sat 2019-10-19 17:53:18 -03
           Universal time: Sat 2019-10-19 20:53:18 UTC
           RTC time: Sat 2019-10-19 20:53:18
           Time zone: America/Sao_Paulo (-03, -0300)
System clock synchronized: yes
systemd-timesyncd.service active: yes
           RTC in local TZ: no
```

As shown in the `Time zone` entry, time zone names based on localities—like `America/Sao_Paulo`—are also accepted. The default time zone for the system is kept in the file `/etc/timezone`, either by the zone’s full descriptive name or offset. Generic time zone names given by the UTC offset must include `Etc` as the first part of the name. So, to set the default time zone to `GMT+3`, the name of the time zone must be `Etc/GMT+3`:

```
$ cat /etc/timezone
Etc/GMT+3
```

Although time zone names based on localities do not require the time offset to work, they are not so straight forward to choose from. The same zone can have more than one name, which can make it difficult to remember. In order to ease this issue, command `tzselect` offers an interactive method which will guide the user towards the correct time zone definition. Command `tzselect` should be available by default in all Linux distributions, as it is provided by the package that contains necessary utility programs related to the GNU C Library.

Command `tzselect` will be useful, for example, for a user who wants to identify the time zone for “São Paulo City” in “Brazil”. `tzselect` starts by asking the macro region of the desired location:

```
$ tzselect
Please identify a location so that time zone rules can be set correctly.
Please select a continent, ocean, "coord", or "TZ".
 1) Africa
 2) Americas
 3) Antarctica
```

```

4) Asia
5) Atlantic Ocean
6) Australia
7) Europe
8) Indian Ocean
9) Pacific Ocean
10) coord - I want to use geographical coordinates.
11) TZ - I want to specify the time zone using the Posix TZ format.
#? 2

```

Option 2 is for (North and South) American locations, not necessarily in the same time zone. It is also possible to specify the time zone with geographical coordinates or with the offset notation, also known as the *Posix TZ format*. The next step is to choose the country:

```

Please select a country whose clocks agree with yours.
1) Anguilla           19) Dominican Republic  37) Peru
2) Antigua & Barbuda 20) Ecuador             38) Puerto Rico
3) Argentina         21) El Salvador        39) St Barthelemy
4) Aruba             22) French Guiana      40) St Kitts & Nevis
5) Bahamas          23) Greenland          41) St Lucia
6) Barbados         24) Grenada            42) St Maarten (Dutch)
7) Belize           25) Guadeloupe        43) St Martin (French)
8) Bolivia          26) Guatemala         44) St Pierre & Miquelon
9) Brazil           27) Guyana             45) St Vincent
10) Canada          28) Haiti              46) Suriname
11) Caribbean NL    29) Honduras           47) Trinidad & Tobago
12) Cayman Islands  30) Jamaica            48) Turks & Caicos Is
13) Chile           31) Martinique        49) United States
14) Colombia        32) Mexico             50) Uruguay
15) Costa Rica      33) Montserrat        51) Venezuela
16) Cuba            34) Nicaragua         52) Virgin Islands (UK)
17) Curaçao         35) Panama             53) Virgin Islands (US)
18) Dominica        36) Paraguay
#? 9

```

Brazil's territory spans four time zones, so the country information alone is not enough to set the time zone. In the next step `tzselect` will require the user to specify the local region:

```

Please select one of the following time zone regions.
1) Atlantic islands
2) Pará (east); Amapá
3) Brazil (northeast: MA, PI, CE, RN, PB)

```

```

4) Pernambuco
5) Tocantins
6) Alagoas, Sergipe
7) Bahia
8) Brazil (southeast: GO, DF, MG, ES, RJ, SP, PR, SC, RS)
9) Mato Grosso do Sul
10) Mato Grosso
11) Pará (west)
12) Rondônia
13) Roraima
14) Amazonas (east)
15) Amazonas (west)
16) Acre
#? 8

```

Not all locality names are available, but choosing the closest region will be good enough. The given information will then be used by `tzselect` to display the corresponding time zone:

The following information has been given:

```

Brazil
Brazil (southeast: GO, DF, MG, ES, RJ, SP, PR, SC, RS)

```

```

Therefore TZ='America/Sao_Paulo' will be used.
Selected time is now:   sex out 18 18:47:07 -03 2019.
Universal Time is now: sex out 18 21:47:07 UTC 2019.
Is the above information OK?
1) Yes
2) No
#? 1

```

You can make this change permanent for yourself by appending the line

```
TZ='America/Sao_Paulo'; export TZ
```

to the file `/.profile` in your home directory; then log out and log in again.

Here is that TZ value again, this time on standard output so that you can use the `/usr/bin/tzselect` command in shell scripts:

```
America/Sao_Paulo
```

The resulting time zone name, `America/Sao_Paulo`, can also be used as the content of the `/etc/timezone` to inform the default time zone for the system:

```
$ cat /etc/timezone
America/Sao_Paulo
```

As stated by the output of `tzselect`, the environment variable `TZ` defines the time zone for the shell session, whatever the system's default time zone is. Adding the line `TZ='America/Sao_Paulo'; export TZ` to the file `~/.profile` will make `America/Sao_Paulo` the time zone for the user's future sessions. The `TZ` variable can also be temporarily modified during the current session, in order to display the time in a different time zone:

```
$ env TZ='Africa/Cairo' date
Mon Oct 21 15:45:21 EET 2019
```

In the example, command `env` will run the given command in a new sub-shell session with the same environment variables of the current session, except for the variable `TZ`, modified by the argument `TZ='Africa/Cairo'`.

## Daylight Savings Time

Many regions adopt daylight savings time for part of the year—when clocks are typically adjusted by an hour—that could lead a misconfigured system to report the wrong time during that season of the year.

The `/etc/localtime` file contains the data used by the operating system to adjust its clock accordingly. Standard Linux systems have files for all time zones in the directory `/usr/share/zoneinfo/`, so `/etc/localtime` is just a symbolic link to the actual data file inside that directory. The files in `/usr/share/zoneinfo/` are organized by the name of the corresponding time zone, so the data file for the `America/Sao_Paulo` time zone will be `/usr/share/zoneinfo/America/Sao_Paulo`

As the definitions for the daylight savings time may change, it is important to keep the files in `/usr/share/zoneinfo/` up to date. The upgrade command of the package management tool provided by the distribution should update them every time a new version is available.

## Language and Character Encoding

Linux systems can work with a wide variety of languages and non-western character encodings, definitions known as *locales*. The most basic locale configuration is the definition of the environment variable `LANG`, from which most shell programs identify the language to use.

The content of the `LANG` variable follows the format `ab_CD`, where `ab` is the language code and `CD` is the region code. The language code should follow the ISO-639 standard and the region code should follow the ISO-3166 standard. A system configured to use Brazilian Portuguese, for example, should have the `LANG` variable defined to `pt_BR.UTF-8`:

```
$ echo $LANG
pt_BR.UTF-8
```

As seen in the sample output, the `LANG` variable also holds the character encoding intended for the system. ASCII, short for *American Standard Code for Information Interchange*, was the first widely used character encoding standard for electronic communication. However, since ASCII has a very limited range of available numerical values and it was based on the English alphabet, it does not contain characters used by other languages or an extended set of non-alphabetical symbols. The UTF-8 encoding is a *Unicode Standard* for the ordinary western characters, plus many other non-conventional symbols. As stated by the *Unicode Consortium*, the maintainer of the *Unicode Standard*, it should be adopted by default to ensure compatibility between computer platforms:

The Unicode Standard provides a unique number for every character, no matter what platform, device, application or language. It has been adopted by all modern software providers and now allows data to be transported through many different platforms, devices and applications without corruption. Support of Unicode forms the foundation for the representation of languages and symbols in all major operating systems, search engines, browsers, laptops, and smart phones — plus the Internet and World Wide Web (URLs, HTML, XML, CSS, JSON, etc.). (...) the Unicode Standard and the availability of tools supporting it are among the most significant recent global software technology trends.

— The Unicode Consortium, What is Unicode?

Some systems may still use ISO defined standards—like the ISO-8859-1 standard—for the encoding of non-ASCII characters. However, such character encoding standards should be deprecated in favor of Unicode encodings standards. Nevertheless, all major operating systems tend to adopt the Unicode standard by default.

System wide locale settings are configured in the file `/etc/locale.conf`. Variable `LANG` and other locale related variables are assigned in this file like an ordinary shell variable, for example:

```
$ cat /etc/locale.conf
LANG=pt_BR.UTF-8
```

Users can use a custom locale configuration by redefining the `LANG` environment variable. It can

be done for the current session only or for future sessions, by adding the new definition to the user's Bash profile in `~/.bash_profile` or `~/.profile`. Until the user logs in, however, the default system locale will still be used by user independent programs, like the display manager's login screen.

**TIP** The command `localectl`, available on systems employing *systemd* as the system manager, can also be used to query and change the system locale. For example:  
`localectl set-locale LANG=en_US.UTF-8.`

In addition to the `LANG` variable, other environment variables have an affect on specific locale aspects, like which currency symbol to use or the correct thousand separator for numbers:

### **LC\_COLLATE**

Sets the alphabetical ordering. One of its purposes is to define the order files and directories are listed.

### **LC\_CTYPE**

Sets how the system will treat certain sets of characters. It defines, for example, which characters to consider as *uppercase* or *lowercase*.

### **LC\_MESSAGES**

Sets the language to display program messages (mostly GNU programs).

### **LC\_MONETARY**

Sets the money unit and currency format.

### **LC\_NUMERIC**

Sets the numerical format for non-monetary values. Its main purpose is to define the thousand and decimal separators.

### **LC\_TIME**

Sets the time and date format.

### **LC\_PAPER**

Sets the standard paper size.

### **LC\_ALL**

Overrides all other variables, including `LANG`.

The `locale` command will show all defined variables in the current locale configuration:

**\$ locale**

```
LANG=pt_BR.UTF-8
LC_CTYPE="pt_BR.UTF-8"
LC_NUMERIC=pt_BR.UTF-8
LC_TIME=pt_BR.UTF-8
LC_COLLATE="pt_BR.UTF-8"
LC_MONETARY=pt_BR.UTF-8
LC_MESSAGES="pt_BR.UTF-8"
LC_PAPER=pt_BR.UTF-8
LC_NAME=pt_BR.UTF-8
LC_ADDRESS=pt_BR.UTF-8
LC_TELEPHONE=pt_BR.UTF-8
LC_MEASUREMENT=pt_BR.UTF-8
LC_IDENTIFICATION=pt_BR.UTF-8
LC_ALL=
```

The only undefined variable is `LC_ALL`, which can be used to temporarily override all the other locale settings. The following example shows how the `date` command—running in a system configured to `pt_BR.UTF-8` locale— will modify its output to comply with the new `LC_ALL` variable:

**\$ date**

```
seg out 21 10:45:21 -03 2019
$ env LC_ALL=en_US.UTF-8 date
Mon Oct 21 10:45:21 -03 2019
```

The modification of the `LC_ALL` variable made both abbreviations for the day of the week and month name to be shown in American English (`en_US`). It is not mandatory, however, to set the same locale for all variables. It is possible, for example, to have the language defined to `pt_BR` and the numerical format (`LC_NUMERIC`) set to the American standard.

Some localisation settings change how programs deal with alphabetical ordering and number formats. Whilst conventional programs are usually prepared to correctly choose a common locale for such situations, scripts can behave unexpectedly when trying to correctly alphabetically order a list of items, for example. For this reason, it is recommended to set the environment variable `LANG` to the common `C` locale, as in `LANG=C`, so the script produces unambiguous results, regardless the localisation definitions used in the system where it is executed. The `C` locale only conducts a simple bitwise comparison, so it will also perform better than the others.

## Encoding Conversion

Text may appear with unintelligible characters when displayed on a system with a character encoding configuration other than the system where the text was created. The command `iconv` can be used to solve this issue, by converting the file from its original character encoding to the desired one. For example, to convert a file named `original.txt` from the ISO-8859-1 encoding to the file named `converted.txt` using UTF-8 encoding, the following command can be used:

```
$ iconv -f ISO-8859-1 -t UTF-8 original.txt > converted.txt
```

The option `-f ISO-8859-1` (or `--from-code=ISO-8859-1`) sets the encoding of the original file and option `-t UTF-8` (or `--to-code=UTF-8`) sets the encoding for the converted file. All encoding supported by command `iconv` are listed with command `iconv -l` or `iconv --list`. Instead of using the output redirection, like in the example, option `-o converted.txt` or `--output converted.txt` could also be used.



## Guided Exercises

1. Based on the following output of the command `date`, what is the time zone of the system in GMT notation?

```
$ date
Mon Oct 21 18:45:21 +05 2019
```

2. To what file should the symbolic link `/etc/localtime` point to in order to make Europe/Brussels the system's default local time?

3. Characters in text files may not be rendered correctly in a system with a character encoding different from that used in the text document. How could `iconv` be used to convert the WINDOWS-1252 encoded file `old.txt` to the file `new.txt` using UTF-8 encoding?

## Explorational Exercises

1. What command will make `Pacific/Auckland` the default time zone for the current shell session?

2. Command `uptime` shows, among other things, the *load average* of the system in fractional numbers. It uses the current locale settings to decide if the decimal place separator should be a dot or a comma. If, for example, the current locale is set to `de_DE.UTF-8` (the standard locale of Germany), `uptime` will use a comma as the separator. Knowing that in the American English language the dot is used as the separator, what command will make `uptime` display the fractions using a dot instead of a comma for the rest of the current session?

3. Command `iconv` will replace all characters outside the target character set with a question mark. If `//TRANSLIT` is appended to the target encoding, characters not represented in the target character set will be replaced (transliterated) by one or more similar looking characters. How could this method be used to convert a UTF-8 text file named `readme.txt` to a plain ASCII file named `ascii.txt`?

## Summary

This lesson covers how to set up a Linux system to work with custom languages and time settings. Character encoding concepts and settings are also covered, as they are very important to correctly render text content. The lesson goes through the following topics:

- How Linux systems select the language to display shell messages.
- Understanding how time zones affect the local time.
- How to identify the appropriate time zone and modify system settings accordingly.
- What character encodings are and how to convert between them.

The commands and procedures addressed were:

- Locale and time related environment variables, such as `LC_ALL`, `LANG` and `TZ`.
- `/etc/timezone`
- `/etc/localtime`
- `/usr/share/zoneinfo/`
- `locale`
- `tzselect`
- `timedatectl`
- `date`
- `iconv`

## Answers to Guided Exercises

1. Based on the following output of the command `date`, what is the time zone of the system in GMT notation?

```
$ date
Mon Oct 21 18:45:21 +05 2019
```

It's the `Etc/GMT+5` time zone.

2. To what file should the symbolic link `/etc/localtime` point to in order to make `Europe/Brussels` the system's default local time?

The link `/etc/localtime` should point to `/usr/share/zoneinfo/Europe/Brussels`.

3. Characters in text files may not be rendered correctly in a system with a character encoding different from that used in the text document. How could `iconv` be used to convert the `WINDOWS-1252` encoded file `old.txt` to the file `new.txt` using `UTF-8` encoding?

Command `iconv -f WINDOWS-1252 -t UTF-8 -o new.txt old.txt` will perform the desired conversion.

## Answers to Explorational Exercises

1. What command will make Pacific/Auckland the default time zone for the current shell session?

```
export TZ=Pacific/Auckland
```

2. Command `uptime` shows, among other things, the *load average* of the system in fractional numbers. It uses the current locale settings to decide if the decimal place separator should be a dot or a comma. If, for example, the current locale is set to `de_DE.UTF-8` (the standard locale of Germany), `uptime` will use a comma as the separator. Knowing that in the American English language the dot is used as the separator, what command will make `uptime` display the fractions using a dot instead of a comma for the rest of the current session?

The command `export LC_NUMERIC=en_US.UTF-8` or `export LC_ALL=en_US.UTF-8`.

3. Command `iconv` will replace all characters outside the target character set with a question mark. If `//TRANSLIT` is appended to the target encoding, characters not represented in the target character set will be replaced (transliterated) by one or more similar looking characters. How this method can be used to convert a UTF-8 text file named `readme.txt` to a plain ASCII file named `ascii.txt`?

Command `iconv -f UTF-8 -t ASCII//TRANSLIT -o ascii.txt readme.txt` will perform the desired conversion.



## **Topic 108: Essential System Services**



## 108.1 Maintain system time

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 108.1](#)

### Weight

3

### Key knowledge areas

- Set the system date and time.
- Set the hardware clock to the correct time in UTC.
- Configure the correct timezone.
- Basic NTP configuration using `ntpd` and `chrony`.
- Knowledge of using the `pool.ntp.org` service.
- Awareness of the `ntpq` command.

### Partial list of the used files, terms and utilities

- `/usr/share/zoneinfo/`
- `/etc/timezone`
- `/etc/localtime`
- `/etc/ntp.conf`
- `/etc/chrony.conf`
- `date`
- `hwclock`
- `timedatectl`

- ntpd
- ntpdate
- chronyc
- pool.ntp.org





# 108.1 Lesson 1

<b>Certificate:</b>	LPIC-1 (102)
<b>Version:</b>	5.0
<b>Topic:</b>	108 Essential System Services
<b>Objective:</b>	108.1 Maintain system time
<b>Lesson:</b>	1 of 2

## Introduction

Accurate timekeeping is absolutely crucial for modern computing. The implementation of keeping time, however, is surprisingly complex. The practice of keeping time seems trivial to an end-user, but the system needs to be able to handle many idiosyncrasies and edge cases intelligently. Consider that time zones are not static, but may be changed by an administrative or political decision. A country may choose to stop observing Daylight Savings Time. Any program must be able to handle those changes logically. Fortunately for system administrators, the solutions for timekeeping on the Linux operating system are mature, robust and usually work without much interference.

When a Linux computer boots up, it starts keeping time. We refer to this as a *system clock*, since it is updated by the operating system. In addition, modern computers will also have a *hardware* or *real time clock*. This hardware clock is often a feature of the motherboard and keeps time regardless if the computer is running or not. During boot, the system time is set from the hardware clock, but for the most part these two clocks run independently of each other. In this lesson we will be discussing methods of interacting with both the system and hardware clocks.

On most modern Linux systems, system time and hardware time are synchronised to *network*

*time*, which is implemented by the *Network Time Protocol* (NTP). In the vast majority of cases, the only configuration a normal user will be required to do is to set their time zone and NTP will take care of the rest. However, we will cover some ways of working with time manually and the specifics of configuring network time will be covered in the next lesson.

## Local Versus Universal Time

The system clock is set to Coordinated Universal Time (UTC), which is the local time at Greenwich, United Kingdom. Usually a user wants to know their *local time*. Local time is calculated by taking UTC time and applying an *offset* based on time zone and Daylight Savings. In this way, a lot of complexity can be avoided.

The system clock can be set to either UTC time or local time, but it is recommended that it also be set to UTC time.

## Date

`date` is a core utility which simply prints local time:

```
$ date
Sun Nov 17 12:55:06 EST 2019
```

Modifying the options of the `date` command will change the format of the output.

For example, a user can use `date -u` to view the current UTC time.

```
$ date -u
Sun Nov 17 18:02:51 UTC 2019
```

Some other commonly-used options will return the local time in a format which adheres to an accepted RFC format:

### -I

Date/time in ISO 8601 format. Appending `date (-Idate)` will limit the output to date only. Other formats are `hours`, `minutes`, `seconds` and `ns` for nanoseconds.

### -R

Returns date and time in RFC 5322 format.

**--rfc-3339**

Returns date and time in RFC 3339 format.

The format of `date` can be customized by the user with sequences specified in the man page. For example, the current time can be formatted as Unix time thusly:

```
$ date +%s
1574014515
```

From `date`'s man page we can see that `%s` refers to Unix time.

Unix time is used internally on most Unix-like systems. It stores UTC time as the number of seconds since *Epoch*, which has been defined as January 1st, 1970.

**NOTE**

The number of bits required to store Unix time at the present moment is 32 bits. There is a future issue when 32 bits will become insufficient to contain the current time in Unix format. This will cause serious issues for any 32-bit Linux systems. Fortunately, this will not occur until January 19, 2038.

Using these sequences, we are able to format date and time in almost any format required by any application. Of course, in most cases it is far preferable to stick with an accepted standard.

Additionally, `date --date` can be used to format a time that is not the current time. In this scenario, a user can specify the date to be applied to the system using Unix time for example:

```
$ date --date='@1564013011'
Wed Jul 24 20:03:31 EDT 2019
```

Using the `--debug` option can be very useful for ensuring that a date can be successfully parsed. Observe what happens when passing a valid date to the command:

```
$ date --debug --date="Fri, 03 Jan 2020 14:00:17 -0500"
date: parsed day part: Fri (day ordinal=0 number=5)
date: parsed date part: (Y-M-D) 2020-01-03
date: parsed time part: 14:00:17 UTC-05
date: input timezone: parsed date/time string (-05)
date: using specified time as starting value: '14:00:17'
date: warning: day (Fri) ignored when explicit dates are given
date: starting date/time: '(Y-M-D) 2020-01-03 14:00:17 TZ=-05'
date: '(Y-M-D) 2020-01-03 14:00:17 TZ=-05' = 1578078017 epoch-seconds
date: timezone: system default
```

```
date: final: 1578078017.000000000 (epoch-seconds)
date: final: (Y-M-D) 2020-01-03 19:00:17 (UTC)
date: final: (Y-M-D) 2020-01-03 14:00:17 (UTC-05)
```

This can be a handy tool when troubleshooting an application that generates a date.

## Hardware Clock

A user may run the `hwclock` command to view the time as maintained on the real-time clock. This command will require elevated privileges, so we will use `sudo` to call the command in this case:

```
$ sudo hwclock
2019-11-20 11:31:29.217627-05:00
```

Using the option `--verbose` will return more output which might be useful for troubleshooting:

```
$ sudo hwclock --verbose
hwclock from util-linux 2.34
System Time: 1578079387.976029
Trying to open: /dev/rtc0
Using the rtc interface to the clock.
Assuming hardware clock is kept in UTC time.
Waiting for clock tick...
...got clock tick
Time read from Hardware Clock: 2020/01/03 19:23:08
Hw clock time : 2020/01/03 19:23:08 = 1578079388 seconds since 1969
Time since last adjustment is 1578079388 seconds
Calculated Hardware Clock drift is 0.000000 seconds
2020-01-03 14:23:07.948436-05:00
```

Note the `Calculated Hardware Clock drift`. This output can tell you if system time and hardware time are deviating from one another.

## timedatectl

`timedatectl` is a command that can be used to check the general status of time and date, including whether or not network time has been synchronised (Network Time Protocol will be covered in the next lesson).

By default `timedatectl` returns information similar to `date`, but with the addition of the RTC

(hardware) time as well as the status of the NTP service:

```
$ timedatectl
    Local time: Thu 2019-12-05 11:08:05 EST
    Universal time: Thu 2019-12-05 16:08:05 UTC
    RTC time: Thu 2019-12-05 16:08:05
    Time zone: America/Toronto (EST, -0500)
System clock synchronized: yes
    NTP service: active
    RTC in local TZ: no
```

## Setting Time Using `timedatectl`

If NTP is unavailable, it is recommended to use `timedatectl` rather than `date` or `hwclock` to set time:

```
# timedatectl set-time '2011-11-25 14:00:00'
```

The process is similar to that of `date`. The user can also set time independent of date using the format `HH:MM:SS`.

## Setting Timezone Using `timedatectl`

`timedatectl` is the preferred way of setting the local time zone on `systemd` based Linux systems when no GUI exists. `timedatectl` will list possible time zones and then the time zone can be set using one of these as an argument.

First we will list possible timezones:

```
$ timedatectl list-timezones
Africa/Abidjan
Africa/Accra
Africa/Algiers
Africa/Bissau
Africa/Cairo
...
```

The list of possible time zones is long, so the use of the `grep` command is recommended in this case.

Next we can set the timezone using one of the elements of the list that was returned:

```
$ timedatectl set-timezone Africa/Cairo
$ timedatectl
      Local time: Thu 2019-12-05 18:18:10 EET
      Universal time: Thu 2019-12-05 16:18:10 UTC
      RTC time: Thu 2019-12-05 16:18:10
      Time zone: Africa/Cairo (EET, +0200)
System clock synchronized: yes
      NTP service: active
      RTC in local TZ: no
```

Keep in mind that the name of the time zone must be exact. `Africa/Cairo` for example will change the time zone, but `cairo` or `africa/cairo` will not.

## Disabling NTP Using `timedatectl`

In some cases it might be necessary to disable NTP. This could be done using `systemctl` but we will demonstrate this using `timedatectl`:

```
# timedatectl set-ntp no
$ timedatectl
      Local time: Thu 2019-12-05 18:19:04 EET Universal time: Thu 2019-12-05 16:19:04
UTC
      RTC time: Thu 2019-12-05 16:19:04
      Time zone: Africa/Cairo (EET, +0200)
      NTP enabled: no
      NTP synchronized: no
      RTC in local TZ: no
      DST active: n/a
```

## Setting Time Zone Without `timedatectl`

Setting time zone information is a standard step when installing Linux on a new machine. If there is a graphical installation process, this will most likely be handled without any further user input.

The `/usr/share/zoneinfo` directory contains information for the different time zones that are possible. In the `zoneinfo` directory, there are subdirectories that contain the names of continents as well as other symbolic links. It is recommended to find your region's `zoneinfo` starting from your continent.

zoneinfo files contain rules required to calculate the local time offset in relation to UTC, and they also are important if your region observes Daylight Savings Time. The contents of /etc/localtime will be read when Linux needs to determine the local time zone. In order to set the time zone without the use of a GUI, the user should create a symbolic link for their location from /usr/share/zoneinfo to /etc/localtime. For example:

```
$ ln -s /usr/share/zoneinfo/Canada/Eastern /etc/localtime
```

After setting the correct time zone, it is often recommended to run:

```
# hwclock --systohc
```

This will set the *hardware clock* from the *system clock* (that is, the real-time clock will be set to the same time as date). Please note that this command is run with root privileges, in this case by being logged in as root.

/etc/timezone is similar to /etc/localtime. It is a data representation of the local time zone, and as such it can be read using cat:

```
$ cat /etc/timezone  
America/Toronto
```

Note that this file is not used by all Linux distributions.

## Setting Date and Time Without timedatectl

### NOTE

Most modern Linux systems use systemd for its configuration and services and as such it is not recommended that you use date or hwclock for setting time. systemd uses timedatectl for this. Nonetheless it is important to know these legacy commands in the event that you must administer an older system.

### Using date

date has an option to set the system time. Use --set or -s to set the date and time. You may also choose to use --debug to verify the correct parsing of the command:

```
# date --set="11 Nov 2011 11:11:11"
```

Note that root privileges are required to set the date here. We may also choose to change to time or date independently:

```
# date +%Y%m%d -s "20111125"
```

Here we must specify the sequences so that our string is parsed properly. For example %Y refers to the year, and so the first four digits 2011 will be interpreted as the year 2011. Similarly, %T is the sequence for time, and it is demonstrated here by setting time:

```
# date +%T -s "13:11:00"
```

After changing system time, it is recommended to also set the hardware clock so that both system and hardware clocks are synchronised:

```
# hwclock --systohc
```

systohc means “system clock to hardware clock”.

## Using hwclock

Rather than setting the system clock and updating the hardware clock, you may choose to reverse the process. We will start by setting the hardware clock:

```
# hwclock --set --date "4/12/2019 11:15:19"  
# hwclock  
Fri 12 Apr 2019 6:15:19 AM EST -0.562862 seconds
```

Notice that by default `hwclock` is expecting UTC time, but returns the local time by default.

After setting the hardware clock, we will need to update the system clock from it. `hctosys` can be understood to mean “hardware clock to system clock”.

```
# hwclock --hctosys
```



## Guided Exercise

1. Indicate whether the following commands are displaying or modifying *system time* or *hardware time*:

Command(s)	System	Hardware	Both
<code>date -u</code>			
<code>hwclock --set --date "12:00:00"</code>			
<code>timedatectl</code>			
<code>timedatectl   grep RTC</code>			
<code>hwclock --hctosys</code>			
<code>date +%T -s "08:00:00"</code>			
<code>timedatectl set- time 1980-01-10</code>			

2. Observe the following output, and then correct the format of the argument so that the command is successful:

```
$ date --debug --date "20/20/12 0:10 -3"

date: warning: value 20 has less than 4 digits. Assuming MM/DD/YY[YY]
date: parsed date part: (Y-M-D) 0002-20-20
date: parsed time part: 00:10:00 UTC-03
date: input timezone: parsed date/time string (-03)
date: using specified time as starting value: '00:10:00'
date: error: invalid date/time value:
date:   user provided time: '(Y-M-D) 0002-20-20 00:10:00 TZ=-03'
date:   normalized time: '(Y-M-D) 0003-08-20 00:10:00 TZ=-03'
date:
date:   possible reasons:
date:     numeric values overflow;
date:     incorrect timezone
date: invalid date '20/20/2 0:10 -3'
```

3. Use the `date` command and sequences so that the system's month is set to February. Leave the rest of the date and time unchanged.

4. Assuming that the command above was successful, use `hwclock` to set the hardware clock from the system clock.

5. There is a location called `eucla`. What continent is it part of? Use the `grep` command to find out.

6. Set your current timezone to that of `eucla`.

## Explorational Exercises

1. Which method of setting time is optimal? In what scenario might the preferred method be impossible?

2. Why do you think there are so many methods to accomplish the same thing, i.e. setting system time?

3. After January 19, 2038, Linux System Time will require a 64-bit number to store. However, it is possible that we could simply choose to set a “New Epoch”. For example, January 1st, 2038 at midnight could be set to a New Epoch Time of 0. Why do you think this has not become the preferred solution?

# Summary

In this lesson you learned:

- How to display the time in different formats from the command line.
- The difference between the system clock and the hardware clock in Linux.
- How to manually set the system clock.
- How to manually set the hardware clock.
- How to change the system's time zone.

Commands used in this lesson:

## **date**

Display or the change the system clock. Other options:

### **-u**

Display UTC time.

### **+%s**

Use a sequence to display Epoch time.

### **--date=**

Specify a specific time to display, as opposed to current time.

### **--debug**

Display debug messages when parsing a user-inputted date.

### **-s**

Set system clock manually.

## **hwclock**

Display or change the hardware clock.

### **--systohc**

Use system clock to set hardware clock.

### **--hctosys**

Use hardware clock to set system clock.

**--set --date**

Set hardware clock manually.

**timedatectl**

Display system and hardware clocks, as well as NTP configuration on systemd-based Linux systems.

**set-time**

Set the time manually.

**list-timezones**

List possible timezones.

**set-timezone**

Set timzone manually.

**set-ntp**

Enable/disable NTP.

## Answers to Guided Exercises

1. Indicate whether the following commands are displaying or modifying *system time* or *hardware time*:

Command(s)	System	Hardware	Both
<code>date -u</code>	X		
<code>hwclock --set --date "12:00:00"</code>		X	
<code>timedatectl</code>			X
<code>timedatectl   grep RTC</code>		X	
<code>hwclock --hctosys</code>	X		
<code>date +%T -s "08:00:00"</code>	X		
<code>timedatectl set- time 1980-01-10</code>			X

2. Observe the following output, and then correct the format of the argument so that the command is successful:

```
$ date --debug --date "20/20/12 0:10 -3"

date: warning: value 20 has less than 4 digits. Assuming MM/DD/YY[YY]
date: parsed date part: (Y-M-D) 0002-20-20
date: parsed time part: 00:10:00 UTC-03
date: input timezone: parsed date/time string (-03)
date: using specified time as starting value: '00:10:00'
date: error: invalid date/time value:
date:   user provided time: '(Y-M-D) 0002-20-20 00:10:00 TZ=-03'
date:   normalized time: '(Y-M-D) 0003-08-20 00:10:00 TZ=-03'
date:                                     -----
date:   possible reasons:
date:     numeric values overflow;
date:     incorrect timezone
date: invalid date '20/20/2 0:10 -3'
```

```
date --debug --set "12/20/20 0:10 -3"
```

3. Use the `date` command and sequences so that the system's month is set to February. Leave the rest of the date and time unchanged.

```
date +%m -s "2"
```

4. Assuming that the command above was successful, use `hwclock` to set the hardware clock from the system clock.

```
hwclock -systohc
```

5. There is a location called `eucla`. What continent is it part of? Use the `grep` command to find out. Enter the complete command below:

```
timedatectl list-timezones \| grep -i eucla
```

OR

```
grep -ri eucla /usr/share/zoneinfo
```

6. Set your current timezone to that of `eucla`.

```
timedatectl set-timezone 'Australia/Eucla'
```

or

```
ln -s /usr/share/zoneinfo/Australia/Eucla /etc/localtime
```

## Answers to Explorational Exercises

1. Which method of setting time is optimal? In what scenario might the preferred method be impossible?

In most Linux distributions, NTP is enabled by default and should be left to set system time without interference. However, if there is a Linux system that isn't connected to the internet, NTP will be inaccessible. For example, an embedded Linux system running on industrial equipment might not have network connectivity.

2. Why do you think there are so many methods to accomplish the same thing, i.e. setting system time?

Since setting time has been a requirement of all \*nix systems for decades, there are many legacy methods for setting time that are still maintained.

3. After January 19, 2038, Linux System Time will require a 64-bit number to store. However, it is possible that we could simply choose to set a "New Epoch". For example, January 1st, 2038 at midnight could be set to a New Epoch Time of 0. Why do you think this has not become the preferred solution?

By 2038 the vast majority of computers will already be running 64-bit CPUs, and using a 64-bit number won't degrade performance in any significant way. However, it would be impossible to estimate the risks of "resetting" Epoch time in such a way. There is a lot of legacy software that might be impacted. Banks and large businesses, for example, often have a large amount of older programs that they rely on for internal use. So this scenario, like so many others, is a study in trade-offs. Any 32-bit systems still running in 2038 would be impacted by an Epoch Time overflow, but legacy software would be impacted by changing the value of Epoch.





## 108.1 Lesson 2

<b>Certificate:</b>	LPIC-1 (102)
<b>Version:</b>	5.0
<b>Topic:</b>	108 Essential System Services
<b>Objective:</b>	108.1 Maintain system time
<b>Lesson:</b>	2 of 2

### Introduction

While personal computers are able to keep reasonably accurate time on their own, production computing and network environments requires that very precise time be kept. The most accurate time is measured by *reference clocks*, which are typically atomic clocks. The modern world has devised a system where all internet-connected computer systems can be synchronised to these reference clocks using what is known as the *Network Time Protocol* (NTP). A computer system with NTP will be able to synchronise their system clocks to the time provided by reference clocks. If system time and the time as measured on these servers are different, then the computer will speed up or slow down its internal system time incrementally until system time matches network time.

NTP uses a hierarchical structure to disseminate time. Reference clocks are connected to servers at the top of the hierarchy. These servers are *Stratum 1* machines and typically are not accessible to the public. Stratum 1 machines are however accessible to *Stratum 2* machines, which are accessible to *Stratum 3* machines and so on. Stratum 2 servers are accessible to the public, as are any machines lower in the hierarchy. When setting up NTP for a large network, it is good practice to have a small number of computers connect to Stratum 2+ servers, and then have those machines provide NTP to all other machines. In this way, demands on Stratum 2 machines can be

minimized.

There are some important terms that come up when discussing NTP. Some of these terms are used in the commands we will use to track the status of NTP on our machines:

### Offset

This refers to the absolute difference between system time and NTP time. For example, if the system clock reads 12:00:02 and NTP time reads 11:59:58, then the offset between the two clocks is four seconds.

### Step

If the time offset between the NTP provider and a consumer is greater than 128ms, then NTP will perform a single significant change to system time, as opposed to slowing or speeding the system time. This is called *stepping*.

### Slew

Slewing refers to the changes made to system time when the offset between system time and NTP is less than 128ms. If this is the case, then changes will be made gradually. This is referred to as *slewing*.

### Insane Time

If the offset between system time and NTP time is greater than 17 minutes, then the system time is considered *insane* and the NTP daemon will not introduce any changes to system time. Special steps will have to be taken to bring system time within 17 minutes of proper time.

### Drift

Drift refers to the phenomenon where two clocks become out of sync over time. Essentially if two clocks are initially synchronised but then become out of sync over time, then clock drift is occurring.

### Jitter

Jitter refers to the amount of drift since the last time a clock was queried. So if the last NTP sync occurred 17 minutes ago, and the offset between the NTP provider and consumer is 3 milliseconds, then 3 milliseconds is the jitter.

Now we will discuss some of the specific ways that Linux implements NTP.

## timedatectl

If your Linux distribution uses `timedatectl`, then by default it implements an *SNTP* client rather than a full NTP implementation. This is a less complex implementation of network time and

means that your machine will not serve NTP to other connected computers.

In this case, SNTP will not work unless the `timesyncd` service is running. As with all `systemd` services, we can verify that it is running with:

```
$ systemctl status systemd-timesyncd
● systemd-timesyncd.service - Network Time Synchronization
   Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; enabled; vendor preset:
   enabled)
   Drop-In: /lib/systemd/system/systemd-timesyncd.service.d
            └─disable-with-time-daemon.conf
   Active: active (running) since Thu 2020-01-09 21:01:50 EST; 2 weeks 1 days ago
     Docs: man:systemd-timesyncd.service(8)
  Main PID: 1032 (systemd-timesyn)
    Status: "Synchronized to time server for the first time 91.189.89.198:123
(ntp.ubuntu.com)."
```

```
   Tasks: 2 (limit: 4915)
  Memory: 3.0M
   CGroup: /system.slice/systemd-timesyncd.service
            └─1032 /lib/systemd/systemd-timesyncd

Jan 11 13:06:18 NeoMex systemd-timesyncd[1032]: Synchronized to time server for the first
time 91.189.91.157:123 (ntp.ubuntu.com).
...
```

The status of `timedatectl` SNTP synchronisation can be verified using `show-timesync`:

```
$ timedatectl show-timesync --all
LinkNTPServers=
SystemNTPServers=
FallbackNTPServers=ntp.ubuntu.com
ServerName=ntp.ubuntu.com
ServerAddress=91.189.89.198
RootDistanceMaxUsec=5s
PollIntervalMinUsec=32s
PollIntervalMaxUsec=34min 8s
PollIntervalUsec=34min 8s
NTPMessage={ Leap=0, Version=4, Mode=4, Stratum=2, Precision=-23, RootDelay=8.270ms,
RootDispersion=18.432ms, Reference=91EECB0E, OriginateTimestamp=Sat 2020-01-25 18:35:49 EST,
ReceiveTimestamp=Sat 2020-01-25 18:35:49 EST, TransmitTimestamp=Sat 2020-01-25 18:35:49 EST,
DestinationTimestamp=Sat 2020-01-25 18:35:49 EST, Ignored=no PacketCount=263, Jitter=2.751ms
}
```

```
Frequency=-211336
```

This configuration might be adequate for most situations, but as noted before it will be insufficient if one is hoping to synchronise several clients in a network. In this case it is recommended to install a full NTP client.

## NTP Daemon

The system time is compared to network time on a regular schedule. For this to work we must have a *daemon* running in the background. For many Linux systems, the name of this daemon is `ntpd`. `ntpd` will allow a machine to not only be a *time consumer* (that is, able to sync its own clock from an outside source), but also to *provide* time to other machines.

Let us assume that our computer is `systemd`-based and it uses `systemctl` to control daemons. We will install `ntp` packages using the appropriate package manager and then ensure that our `ntpd` daemon is running by checking its status:

```
$ systemctl status ntpd
```

- `ntpd.service` - Network Time Service
  - Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; vendor preset: disabled)
  - Active: active (running) since Fri 2019-12-06 03:27:21 EST; 7h ago
  - Process: 856 ExecStart=/usr/sbin/ntpd -u ntp:ntp \$OPTIONS (code=exited, status=0/SUCCESS)
  - Main PID: 867 (ntpd)
  - CGroup: /system.slice/ntpd.service
    - └─867 /usr/sbin/ntpd -u ntp:ntp -g

In some cases it might be required to both start and enable `ntpd`. On most Linux machines this is accomplished with:

```
# systemctl enable ntpd && systemctl start ntpd
```

NTP queries happen on TCP Port 123. If NTP fails, ensure that this port is open and listening.

## NTP Configuration

NTP is able to poll several sources and to select the best candidates to use for setting system time. If a network connection is lost, NTP uses previous adjustments from its history to estimate future adjustments.

Depending on your distribution of Linux, the list of network time servers will be stored in different places. Let us assume that `ntp` is installed on your machine.

The file `/etc/ntp.conf` contains configuration information about how your system synchronises with network time. This file can be read and modified using `vi` or `nano`.

By default, the NTP servers used will be specified in a section like this:

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
server 0.centos.pool.ntp.org iburst
server 1.centos.pool.ntp.org iburst
server 2.centos.pool.ntp.org iburst
server 3.centos.pool.ntp.org iburst
```

The syntax for adding NTP servers looks like this:

```
server (IP Address)
server server.url.localhost
```

Server addresses can be IP addresses or URLs if DNS has been properly configured. In this case, the server will always be queried.

A network administrator might also consider using (or setting up) a *pool*. In this case, we assume that there are several NTP providers, all running NTP daemons and with the same time. When a client queries a pool, a provider is selected at random. This helps to distribute network load among many machines so that no one machine in the pool is handling all NTP queries.

Commonly, `/etc/ntp.conf` will be populated with a server pool called `pool.ntp.org`. So for example, `server 0.centos.pool.ntp.org` is a default NTP pool provided to CentOS machines.

## pool.ntp.org

The NTP servers used by default are an open source project. More information can be found at [ntp.pool.org](http://ntp.pool.org).

Consider if the NTP Pool is appropriate for your use. If business, organization or human life depends on having correct time or can be harmed by it being wrong, you shouldn't "just get it off the internet". The NTP Pool is generally very high quality, but it is a service run by volunteers in their spare time. Please talk to your equipment and service vendors about getting local and reliable service setup for you. See also our terms of service. We recommend

time servers from Meinberg, but you can also find time servers from End Run, Spectracom and many others.

— ntpool.org

## ntpdate

During initial setup, system time and NTP might be seriously de-synchronised. If the *offset* between system and NTP time is greater than 17 minutes, then the NTP daemon will not make changes to system time. In this scenario manual intervention will be required.

Firstly, if `ntpd` is running it will be necessary to *stop* the service. Use `systemctl stop ntpd` to do so.

Next, use `ntpdate pool.ntp.org` to perform an initial one-time synchronisation, where `pool.ntp.org` refers to the IP address or URL of an NTP server. More than one sync may be required.

## ntpq

`ntpq` is a utility for monitoring the status of NTP. Once the NTP daemon has been started and configured, `ntpq` can be used to check on its status:

```
$ ntpq -p
      remote                refid                st t when poll reach  delay  offset  jitter
=====
+37.44.185.42    91.189.94.4         3 u  86 128 377 126.509 -20.398  6.838
+ntp2.0x00.lv   193.204.114.233    2 u  82 128 377 143.885  -8.105  8.478
*inspektor-vlan1 121.131.112.137    2 u  17 128 377 112.878 -23.619  7.959
b1-66er.matrix. 18.26.4.105        2 u 484 128  10  34.907  -0.811 16.123
```

In this case `-p` is for *print* and it will print a summary of peers. Host addresses can also be returned as IP addresses using `-n`.

### remote

hostname of the NTP provider.

### refid

Reference ID of the NTP provider.

**st**

Stratum of the provider.

**when**

Number of seconds since the last query.

**poll**

Number of seconds between queries.

**reach**

Status ID to indicate whether a server was reached. Successful connections will increase this number by 1.

**delay**

Time in ms between query and response by the server.

**offset**

Time in ms between system time and NTP time.

**jitter**

Offset in ms between system time and NTP in the last query.

`ntpq` also has an interactive mode, which is accessed when it is run without options or argument. The `?` option will return a list of commands that `ntpq` will recognize.

## chrony

`chrony` is yet another way to implement NTP. It is installed by default on some Linux systems, but is available to download on all major distributions. `chronyd` is the `chrony` daemon, and `chronyc` is the command line interface. It may be required to start and enable `chronyd` before interacting with `chronyc`.

If the `chrony` installation has a default configuration then using the command `chronyc tracking` will provide information about NTP and system time:

```
$ chronyc tracking
Reference ID      : 3265FB3D (bras-vprn-toroon2638w-lp130-11-50-101-251-61.ds1.)
Stratum          : 3
Ref time (UTC)   : Thu Jan 09 19:18:35 2020
System time      : 0.000134029 seconds fast of NTP time
Last offset      : +0.000166506 seconds
```

```
RMS offset      : 0.000470712 seconds
Frequency      : 919.818 ppm slow
Residual freq  : +0.078 ppm
Skew           : 0.555 ppm
Root delay     : 0.006151616 seconds
Root dispersion : 0.010947504 seconds
Update interval : 129.8 seconds
Leap status    : Normal
```

This output contains a lot of information, more than what is available from other implementations.

### **Reference ID**

The reference ID and name to which the computer is currently synced.

### **Stratum**

Number of hops to a computer with an attached reference clock.

### **Ref time**

This is the UTC time at which the last measurement from the reference source was made.

### **System time**

Delay of system clock from synchronized server.

### **Last offset**

Estimated offset of the last clock update.

### **RMS offset**

Long term average of the offset value.

### **Frequency**

This is the rate by which the system's clock would be wrong if chronyd is not correcting it. It is provided in ppm (parts per million).

### **Residual freq**

Residual frequency indicating the difference between the measurements from reference source and the frequency currently being used.

### **Skew**

Estimated error bound of the frequency.



## Root delay

Total of the network path delays to the stratum computer, from which the computer is being synced.

## Leap status

This is the leap status which can have one of the following values – normal, insert second, delete second or not synchronized.

We can also look at detailed information about the last valid NTP update:

```
# chrony ntpdata
Remote address  : 172.105.97.111 (AC69616F)
Remote port    : 123
Local address   : 192.168.122.81 (C0A87A51)
Leap status    : Normal
Version        : 4
Mode           : Server
Stratum        : 2
Poll interval   : 6 (64 seconds)
Precision      : -25 (0.000000030 seconds)
Root delay     : 0.000381 seconds
Root dispersion : 0.000092 seconds
Reference ID    : 61B7CE58 ( )
Reference time  : Mon Jan 13 21:50:03 2020
Offset         : +0.000491960 seconds
Peer delay     : 0.004312567 seconds
Peer dispersion : 0.000000068 seconds
Response time  : 0.000037078 seconds
Jitter asymmetry: +0.00
NTP tests      : 111 111 1111
Interleaved    : No
Authenticated  : No
TX timestamping : Daemon
RX timestamping : Kernel
Total TX       : 15
Total RX       : 15
Total valid RX : 15
```

Finally, `chronyc sources` will return information about the NTP servers that are used to synchronise time:

```
$ chronyc sources
```

```
210 Number of sources = 0
MS Name/IP address      Stratum Poll Reach LastRx Last sample
=====
```

At the moment, this machine has no sources configured. We can add sources from `pool.ntp.org` by opening the `chrony` configuration file. This will usually be located at `/etc/chrony.conf`. When we open this file, we should see that some servers are listed by default:

```
210 Number of sources = 0
MS Name/IP address      Stratum Poll Reach LastRx Last sample
=====
# Most computers using chrony will send measurement requests to one or
# more 'NTP servers'. You will probably find that your Internet Service
# Provider or company have one or more NTP servers that you can specify.
# Failing that, there are a lot of public NTP servers. There is a list
# you can access at http://support.ntp.org/bin/view/Servers/WebHome or
# you can use servers from the 3.arch.pool.ntp.org project.

! server 0.arch.pool.ntp.org iburst iburst
! server 1.arch.pool.ntp.org iburst iburst
! server 2.arch.pool.ntp.org iburst iburst

! pool 3.arch.pool.ntp.org iburst
```

These servers will also serve as a syntax guide when entering our own servers. However, in this case we will simply remove the `!` s at the beginning of each line, thus uncommenting out these lines and using the default servers from the `pool.ntp.org` project.

In addition, in this file we can choose to change the default configuration regarding skew and drift as well as the location of the driftfile and keyfile.

On this machine, we need to make a large initial clock correction. We will choose to uncomment the following line:

```
! makestep 1.0 3
```

After making changes to the configuration file, restart the `chronyd` service and then use `chronyc` `makestep` to manually step the system clock:

```
# chronyc makestep
```

```
200 OK
```

And then use `chronyc tracking` as before to verify that the changes have taken place.

## Guided Exercise

1. Enter the appropriate term for each definition:

Definition	Term
A computer that will share network time with you	
Distance from a reference clock, in hops or steps	
Difference between system time and network time	
Difference between system time and network time since the last NTP poll	
Group of servers that provide network time and share the load between them	

2. Specify which of the commands you would use to output the following values:

Value	<code>chronyc tracking</code>	<code>timedatectl show-timesync --all</code>	<code>ntpq -pn</code>	<code>chrony ntpdata</code>	<code>chronyc sources</code>
Jitter					
Drift					
Interval of Poll					
Offset					
Stratum					
IP Address of Provider					
Root Delay					

3. You are setting up an enterprise network consisting of a Linux server and several Linux desktops. The server has a static IP address of 192.168.0.101. You decide that the server will connect to `pool.ntp.org` and then provide NTP time to the desktops. Describe the configuration of the server and of the desktops.

4. A Linux machine has the incorrect time. Describe the steps you would take to troubleshoot NTP.

# Explorational Exercise

1. Research the differences between SNTP and NTP.

SNTP	NTP

2. Why might a system administrator choose *not* to use `pool.ntp.org`?

3. How would a system administrator choose to join or otherwise contribute to the `pool.ntp.org` project?

# Summary

In this lesson you learned:

- What NTP is and why it is important.
- Configuring the NTP daemon from the `pool.ntp.org` project.
- Using `ntpq` to verify NTP configuration.
- Using `chrony` as an alternative NTP service.

Commands used in this lesson:

## `timedatectl show-timesync --all`

Display SNTP information if using `timedatectl`.

## `ntpdate <address>`

Perform a manual one-time NTP step update.

## `ntpq -p`

Print a history of NTP recent polls. `-n` will replace URLs with IP addresses.

## `chronyc tracking`

Displays NTP status if using `chrony`.

## `chronyc ntpdata`

Displays NTP information about the last poll.

## `chronyc sources`

Displays informations about NTP providers.

## `chronyc makestep`

Perform a manual one-time NTP step update if using `chrony`.

# Answers Guided Exercise

1. Enter the appropriate term for each definition:

Definition	Term
A computer that will share network time with you	Provider
Distance from a reference clock, in hops or steps	Stratum
Difference between system time and network time	Offset
Difference between system time and network time since the last NTP poll	Jitter
Group of servers that provide network time and share the load between them	Pool

2. Specify which of the commands you would use to output the following values:

Value	<code>chronyc tracking</code>	<code>timedatectl show-timesync --all</code>	<code>ntpq -pn</code>	<code>chrony ntpdata</code>	<code>chronyc sources</code>
Jitter		X	X		
Drift					
Interval of Poll	X	X	X (when column)	X	X
Offset	X		X	X	
Stratum	X	X	X	X	X
IP Address of Provider		X	X	X	X
Root Delay	X			X	

3. You are setting up an enterprise network consisting of a Linux server and several Linux desktops. The server has a static IP address of 192.168.0.101. You decide that the server will connect to `pool.ntp.org` and then provide NTP time to the desktops. Describe the



configuration of the server and of the desktops.

Ensure that the server has an `ntpd` service running, rather than `SNTP`. Use `pool.ntp.org` pools in the `/etc/ntp.conf` or `/etc/chrony.conf` file. For each client, specify `192.168.0.101` in each `/etc/ntp.conf` or `/etc/chrony.conf` file.

4. A Linux machine has the incorrect time. Describe the steps you would take to troubleshoot NTP.

First, ensure that the machine is connected to the Internet. Use `ping` for this. Check that an `ntpd` or `SNTP` service is running using `systemctl status ntpd` or `systemctl status systemd-timesyncd`. You may see error messages that provide useful information. Finally, use a command such as `ntpq -p` or `chrony tracking` to verify if any requests have been made. If the system time is drastically different from network time, it may be that system time is considered “insane” and will not be changed without manual intervention. In this case, use a command from the previous lesson or a command such as `ntpdate pool.ntp.org` to perform a one-time ntp synchronisation.

# Answers to Explorational Exercises

1. Research the differences between SNTP and NTP.

SNTP	NTP
less accurate	more accurate
requires fewer resources	requires more resources
cannot act as a time provider	can act as a time provider
steps time only	steps or slews time
requests time from a single source	can monitor multiple NTP servers and use the optimal provider

2. Why might a system administrator choose *not* to use `pool.ntp.org`?

From `ntppool.org`: If it is absolutely crucial to have correct time, you should consider an alternative. Similarly, if your Internet provider has a time server, it is recommended to use that instead.

3. How would a system administrator choose to join or otherwise contribute to the `pool.ntp.org` project?

From `www.ntppool.org`: Your server must have a static IP address and a permanent internet connection. The static IP address must not change at all or at least less than once a year. Beyond that, the bandwidth requirements are modest: 384 - 512 Kbit bandwidth. Stratum 3 or 4 servers are welcome to join.



## 108.2 System logging

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 108.2](#)

### Weight

4

### Key knowledge areas

- Basic configuration of rsyslog.
- Understanding of standard facilities, priorities and actions.
- Query the systemd journal.
- Filter systemd journal data by criteria such as date, service or priority
- Configure persistent systemd journal storage and journal size
- Delete old systemd journal data
- Retrieve systemd journal data from a rescue system or file system copy
- Understand interaction of rsyslog with systemd-journald
- Configuration of logrotate.
- Awareness of syslog and syslog-ng.

### Partial list of the used files, terms and utilities

- `/etc/rsyslog.conf`
- `/var/log/`
- `logger`
- `logrotate`

- `/etc/logrotate.conf`
- `/etc/logrotate.d/`
- `journalctl`
- `systemd-cat`
- `/etc/systemd/journald.conf`
- `/var/log/journal/`



# 108.2 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	108 Essential System Services
<b>Objective:</b>	108.2 System logging
<b>Lesson:</b>	1 of 2

## Introduction

Logs can be a system administrator's best friend. Logs are files (usually text files) where all system and network events are chronologically registered from the moment your system is booted up. Thus, the range of information that can be found in logs includes virtually every aspect of the system: failed authentication attempts, program and service errors, hosts blocked by the firewall, etc. As you can imagine, logs make system administrators' lives a lot easier when it comes to troubleshooting, resource-checking, detection of anomalous behaviour of programs, and so on.

In this lesson we will discuss one of the most common logging facilities currently found in GNU/Linux distributions: `rsyslog`. We will study the different types of logs that exist, where they are stored, what information they include and how that information can be obtained and filtered. We will also discuss how logs can be kept in centralized servers across IP networks, log rotation and the kernel ring buffer.

## System Logging

The moment the kernel and the different processes in your system start running and communicating with one another, a lot of information is generated in the form of messages that

are — for the most part — sent to the logs.

Without logging, searching for an event that happened on a server would give system administrators a headache, hence the importance of having a standardized and centralized way of keeping track of any system events. Logs are determinant and telling when it comes to troubleshooting and security and are reliable data sources for understanding system statistics and making trend predictions.

Leaving aside `systemd-journald` (which we will discuss in the next lesson), logging has traditionally been handled by three main dedicated services: `syslog`, `syslog-ng` (syslog new generation) and `rsyslog` (“the rocket-fast system for log processing”). `rsyslog` brought along important improvements (such as RELP support) and has become the most popular choice nowadays. Each of these services collects messages from other services and programs and stores them in log files, typically under `/var/log`. However, some services take care of their own logs (take — for example — the Apache HTTPD web server or the CUPS printing system). Likewise, the Linux kernel uses an in-memory ring buffer for storing its log messages.

**NOTE** RELP stands for *Reliable Event Logging Protocol* and extends the functionality of the syslog protocol to provide reliable delivery of messages.

Since `rsyslog` has become the *de facto* standard logging facility in all major distros, we will focus on it for the present lesson. `rsyslog` uses a client-server model. The client and the server can live on the same host or in different machines. Messages are sent and received in a particular format and can be kept in centralized `rsyslog` servers across IP networks. `rsyslog`’s daemon — `rsyslogd` — works together with `klogd` (which manages kernel messages). In the next sections `rsyslog` and its logging infrastructure will be discussed.

**NOTE** A daemon is a service that runs in the background. Note the final `d` in daemon names: `klogd` or `rsyslogd`.

## Log Types

Because logs are *variable* data, they are normally found in `/var/log`. Roughly speaking, they can be classified into *system logs* and *service or program logs*.

Let us see some system logs and the information they keep:

### `/var/log/auth.log`

Activities related to authentication processes: logged users, `sudo` information, cron jobs, failed login attempts, etc.

**/var/log/syslog**

A centralized file for practically all of the logs captured by `rsyslogd`. Because it includes so much information, logs are distributed across other files according to the configuration supplied in `/etc/rsyslog.conf`.

**/var/log/debug**

Debug information from programs.

**/var/log/kern.log**

Kernel messages.

**/var/log/messages**

Informative messages which are not related to the kernel but to other services. It is also the default remote client log destination in a centralized log server implementation.

**/var/log/daemon.log**

Information related to daemons or services running in the background.

**/var/log/mail.log**

Information related to the email server, e.g. postfix.

**/var/log/Xorg.0.log**

Information related to the graphics card.

**/var/run/utmp and /var/log/wtmp**

Successful logins.

**/var/log/btmp**

Failed login attempts, e.g. brute force attack via ssh.

**/var/log/faillog**

Failed authentication attempts.

**/var/log/lastlog**

Date and time of recent user logins.

Now let us see a few examples of service logs:

**/var/log/cups/**

Directory for logs of the *Common Unix Printing System*. It commonly includes the following default log files: `error_log`, `page_log` and `access_log`.

## **/var/log/apache2/** or **/var/log/httpd**

Directory for logs of the *Apache Web Server*. It commonly includes the following default log files: `access.log`, `error_log`, and `other_vhosts_access.log`.

## **/var/log/mysql**

Directory for logs of the *MySQL Relational Database Management System*. It commonly includes the following default log files: `error_log`, `mysql.log` and `mysql-slow.log`.

## **/var/log/samba/**

Directory for logs of the *Session Message Block* (SMB) protocol. It commonly includes the following default log files: `log.`, `log.nmbd` and `log.smbd`.

### **NOTE**

The exact name and contents of log files may vary across Linux distributions. There are also logs particular to specific distributions such as `/var/log/dpkg.log` (containing information related to `dpkg` packages) in Debian GNU/Linux and its derivatives.

## **Reading Logs**

To read log files, first ensure you are the root user or have reading permissions on the file. You can use a variety of utilities such as:

### **less or more**

Pagers that allow viewing and scrolling one page at a time:

```

root@debian:~# less /var/log/auth.log
Sep 12 18:47:56 debian sshd[441]: Received SIGHUP; restarting.
Sep 12 18:47:56 debian sshd[441]: Server listening on 0.0.0.0 port 22.
Sep 12 18:47:56 debian sshd[441]: Server listening on :: port 22.
Sep 12 18:47:56 debian sshd[441]: Received SIGHUP; restarting.
Sep 12 18:47:56 debian sshd[441]: Server listening on 0.0.0.0 port 22.
Sep 12 18:47:56 debian sshd[441]: Server listening on :: port 22.
Sep 12 18:49:46 debian sshd[905]: Accepted password for carol from 192.168.1.65 port
44296 ssh2
Sep 12 18:49:46 debian sshd[905]: pam_unix(sshd:session): session opened for user carol
by (uid=0)
Sep 12 18:49:46 debian systemd-logind[331]: New session 2 of user carol.
Sep 12 18:49:46 debian systemd: pam_unix(systemd-user:session): session opened for user
carol by (uid=0)
(...)

```



## zless or zmore

The same as `less` and `more`, but used for logs that are compressed with `gzip` (a common function of `logrotate`):

```

root@debian:~# zless /var/log/auth.log.3.gz
Aug 19 20:05:57 debian sudo:    carol : TTY=pts/0 ; PWD=/home/carol ; USER=root ;
COMMAND=/sbin/shutdown -h now
Aug 19 20:05:57 debian sudo: pam_unix(sudo:session): session opened for user root by
carol(uid=0)
Aug 19 20:05:57 debian lightdm: pam_unix(lightdm-greeter:session): session closed for
user lightdm
Aug 19 23:50:49 debian systemd-logind[333]: Watching system buttons on /dev/input/event2
(Power Button)
Aug 19 23:50:49 debian systemd-logind[333]: Watching system buttons on /dev/input/event3
(Sleep Button)
Aug 19 23:50:49 debian systemd-logind[333]: Watching system buttons on /dev/input/event4
(Video Bus)
Aug 19 23:50:49 debian systemd-logind[333]: New seat seat0.
Aug 19 23:50:49 debian sshd[409]: Server listening on 0.0.0.0 port 22.
(...)

```

## tail

View the last lines in a file (the default is 10 lines). The power of `tail` lies — to a great extent — in the `-f` switch, which will dynamically show new lines as they are appended:

```

root@suse-server:~# tail -f /var/log/messages
2019-09-14T13:57:28.962780+02:00 suse-server sudo: pam_unix(sudo:session): session closed
for user root
2019-09-14T13:57:38.038298+02:00 suse-server sudo:    carol : TTY=pts/0 ; PWD=/home/carol
; USER=root ; COMMAND=/usr/bin/tail -f /var/log/messages
2019-09-14T13:57:38.039927+02:00 suse-server sudo: pam_unix(sudo:session): session opened
for user root by carol(uid=0)
2019-09-14T14:07:22+02:00 debian carol: appending new message from client to remote
server...

```

## head

View the first lines in a file (the default is 10 lines):

```

root@suse-server:~# head -5 /var/log/mail
2019-06-29T11:47:59.219806+02:00 suse-server postfix/postfix-script[1732]: the Postfix
mail system is not running

```

```

2019-06-29T11:48:01.355361+02:00 suse-server postfix/postfix-script[1925]: starting the
Postfix mail system
2019-06-29T11:48:01.391128+02:00 suse-server postfix/master[1930]: daemon started --
version 3.3.1, configuration /etc/postfix
2019-06-29T11:55:39.247462+02:00 suse-server postfix/postfix-script[3364]: stopping the
Postfix mail system
2019-06-29T11:55:39.249375+02:00 suse-server postfix/master[1930]: terminating on signal
15

```

## grep

Filtering utility which allows you to search for specific strings:

```

root@debian:~# grep "dhclient" /var/log/syslog
Sep 13 11:58:48 debian dhclient[448]: DHCPREQUEST of 192.168.1.4 on enp0s3 to 192.168.1.1
port 67
Sep 13 11:58:49 debian dhclient[448]: DHCPACK of 192.168.1.4 from 192.168.1.1
Sep 13 11:58:49 debian dhclient[448]: bound to 192.168.1.4 -- renewal in 1368 seconds.
(...)

```

As you may have noticed, the output is printed in the following format:

- Timestamp
- Hostname from which the message originated
- Name of program/service that generated the message
- The PID of the program that generated the message
- Description of the action that took place

There are a few examples in which logs are not text, but binary files and—consequently—you must use special commands to parse them:

### `/var/log/wtmp`

Use `who` (or `w`):

```

root@debian:~# who
root pts/0 2020-09-14 13:05 (192.168.1.75)
root pts/1 2020-09-14 13:43 (192.168.1.75)

```

### `/var/log/btmp`

Use `utmpdump` or `last -f`:

```
root@debian:~# utmpdump /var/log/btmp
Utmp dump of /var/log/btmp
[6] [01287] [  ] [dave  ] [ssh:notty  ] [192.168.1.75      ] [192.168.1.75  ]
[2019-09-07T19:33:32,000000+0000]
```

## /var/log/faillog

Use faillog:

```
root@debian:~# faillog -a | less
Login          Failures Maximum Latest          On

root           0          0  01/01/70 01:00:00 +0100
daemon        0          0  01/01/70 01:00:00 +0100
bin           0          0  01/01/70 01:00:00 +0100
sys           0          0  01/01/70 01:00:00 +0100
sync          0          0  01/01/70 01:00:00 +0100
games         0          0  01/01/70 01:00:00 +0100
man           0          0  01/01/70 01:00:00 +0100
lp            0          0  01/01/70 01:00:00 +0100
mail          0          0  01/01/70 01:00:00 +0100
(...)
```

## /var/log/lastlog

Use lastlog:

```
root@debian:~# lastlog | less
Username      Port    From          Latest
root          Never logged in
daemon        Never logged in
bin           Never logged in
sys           Never logged in
(...)
sync          Never logged in
avahi         Never logged in
colord        Never logged in
saned         Never logged in
hplip         Never logged in
carol         pts/1   192.168.1.75  Sat Sep 14 13:43:06 +0200 2019
dave          pts/3   192.168.1.75  Mon Sep  2 14:22:08 +0200 2019
```

**NOTE**

There are also graphical tools for reading log files, for example: `gnome-logs` and `KSystemLog`.

## How Messages are Turned into Logs

The following process illustrates how a message is written to a log file:

1. Applications, services and the kernel write messages in special files (sockets and memory buffers), e.g. `/dev/log` or `/dev/kmsg`.
2. `rsyslogd` gets the information from the sockets or memory buffers.
3. Depending on the rules found in `/etc/rsyslog.conf` and/or the files in `/etc/rsyslog.d/`, `rsyslogd` moves the information to the corresponding log file (typically found in `/var/log`).

**NOTE**

A socket is a special file used to transfer information between different processes. To list all sockets on your system, you can use the command `systemctl list-sockets --all`.

## Facilities, Priorities and Actions

`rsyslog` configuration file is `/etc/rsyslog.conf` (in some distributions you can also find configuration files in `/etc/rsyslog.d/`). It is normally divided into three sections: `MODULES`, `GLOBAL DIRECTIVES` and `RULES`. Let us have a look at them by exploring the `rsyslog.conf` file in our Debian GNU/Linux 10 (buster) host—you can use `sudo less /etc/rsyslog.conf` to do so.

`MODULES` includes module support for logging, message capability, and UDP/TCP log reception:

```
#####
#### MODULES ####
#####

module(load="imuxsock") # provides support for local system logging
module(load="imklog")   # provides kernel logging support
#module(load="immark")  # provides --MARK-- message capability

# provides UDP syslog reception
#module(load="imudp")
#input(type="imudp" port="514")

# provides TCP syslog reception
#module(load="imtcp")
```

```
#input(type="imtcp" port="514")
```

GLOBAL DIRECTIVES allow us to configure a number of things such as logs and log directory permissions:

```
#####
#### GLOBAL DIRECTIVES ####
#####

#
# Use traditional timestamp format.
# To enable high precision timestamps, comment out the following line.
#
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat

#
# Set the default permissions for all log files.
#
$FileOwner root
$FileGroup adm
$FileCreateMode 0640
$DirCreateMode 0755
$Umask 0022

#
# Where to place spool and state files
#
$WorkDirectory /var/spool/rsyslog

#
# Include all config files in /etc/rsyslog.d/
#
$IncludeConfig /etc/rsyslog.d/*.conf
```

RULES is where *facilities*, *priorities* and *actions* come in. The settings in this section tell the logging daemon to filter messages according to certain rules and log them or send them where required. To understand these rules, we should first explain the concepts of `rsyslog` facilities and priorities. Each log message is given a *facility* number and keyword that are associated with the Linux internal subsystem that produces the message:

Number	Keyword	Description
0	kern	Linux kernel messages
1	user	User-level messages
2	mail	Mail system
3	daemon	System daemons
4	auth, authpriv	Security/Authorization messages
5	syslog	syslogd messages
6	lpr	Line printer subsystem
7	news	Network news subsystem
8	uucp	UUCP (Unix-to-Unix Copy Protocol) subsystem
9	cron	Clock daemon
10	auth, authpriv	Security/Authorization messages
11	ftp	FTP (File Transfer Protocol) daemon
12	nntp	NTP (Network Time Protocol) daemon
13	security	Log audit
14	console	Log alert
15	cron	Clock daemon
16 - 23	local0 through local7	Local use 0 - 7

Furthermore, each message is assigned a *priority* level:

Code	Severity	Keyword	Description
0	Emergency	emerg, panic	System is unusable
1	Alert	alert	Action must be taken immediately
2	Critical	crit	Critical conditions
3	Error	err, error	Error conditions

Code	Severity	Keyword	Description
4	Warning	warn, warning	Warning conditions
5	Notice	notice	Normal but significant condition
6	Informational	info	Informational messages
7	Debug	debug	Debug-level messages

Here is an excerpt of `rsyslog.conf` from our Debian GNU/Linux 10 (buster) system which including some sample rules:

```
#####
#### RULES ####
#####

# First some standard log files.  Log by facility.
#
auth,authpriv.*          /var/log/auth.log
*.*;auth,authpriv.none  -/var/log/syslog
#cron.*                  /var/log/cron.log
daemon.*                 -/var/log/daemon.log
kern.*                   -/var/log/kern.log
lpr.*                    -/var/log/lpr.log
mail.*                   -/var/log/mail.log
user.*                   -/var/log/user.log

#
# Logging for the mail system.  Split it up so that
# it is easy to write scripts to parse these files.
#
mail.info                 -/var/log/mail.info
mail.warn                 -/var/log/mail.warn
mail.err                  /var/log/mail.err

#
# Some "catch-all" log files.
#
*.=debug;\
    auth,authpriv.none;\
    news.none;mail.none  -/var/log/debug
*.=info;*.=notice;*.=warn;\
```

```
auth,authpriv.none;\
cron,daemon.none;\
mail,news.none      -/var/log/messages
```

The rule format is as follows: `<facility>.<priority> <action>`

The `<facility>.<priority>` selector filters messages to match. Priority levels are hierarchically inclusive, which means rsyslog will match messages of the specified priority and higher. The `<action>` shows what action to take (where to send the log message). Here are a few examples for clarity:

```
auth,authpriv.*      /var/log/auth.log
```

Regardless of their priority (\*), all messages from the `auth` or `authpriv` facilities will be sent to `/var/log/auth.log`.

```
*.*;auth,authpriv.none  -/var/log/syslog
```

All messages — irrespective of their priority (\*) — from all facilities (\*) — discarding those from `auth` or `authpriv` (hence the `.none` suffix) — will be written to `/var/log/syslog` (the minus sign (-) before the path prevents excessive disk writes). Note the semicolon (;) to split the selector and the comma (,) to concatenate two facilities in the same rule (`auth,authpriv`).

```
mail.err             /var/log/mail.err
```

Messages from the `mail` facility with a priority level of `error` or higher (`critical`, `alert` or `emergency`) will be sent to `/var/log/mail.err`.

```
*.=debug;\
    auth,authpriv.none;\
    news.none;mail.none  -/var/log/debug
```

Messages from all facilities with the `debug` priority and no other (=) will be written to `/var/log/debug` — excluding any messages coming from the `auth`, `authpriv`, `news` and `mail` facilities (note the syntax: `;\`).



## Manual Entries into the System Log: `logger`

The `logger` command comes in handy for shell scripting or for testing purposes. `logger` will append any message it receives to `/var/log/syslog` (or to `/var/log/messages` when logging to a remote central log server as you will see later in this lesson):

```
carol@debian:~$ logger this comment goes into "/var/log/syslog"
```

To print the last line in `/var/log/syslog`, use the `tail` command with the `-1` option:

```
root@debian:~# tail -1 /var/log/syslog
Sep 17 17:55:33 debian carol: this comment goes into /var/log/syslog
```

## `rsyslog` as a Central Log Server

To explain this topic we are going to add a new host to our setup. The layout is as follows:

Role	Hostname	OS	IP Address
Central Log Server	suse-server	openSUSE Leap 15.1	192.168.1.6
Client	debian	Debian GNU/Linux 10 (buster)	192.168.1.4

Let us start by configuring the server. First of all, we make sure that `rsyslog` is up and running:

```
root@suse-server:~# systemctl status rsyslog
rsyslog.service - System Logging Service
  Loaded: loaded (/usr/lib/systemd/system/rsyslog.service; enabled; vendor preset: enabled)
  Active: active (running) since Thu 2019-09-17 18:45:58 CEST; 7min ago
  Docs: man:rsyslogd(8)
        http://www.rsyslog.com/doc/
  Main PID: 832 (rsyslogd)
  Tasks: 5 (limit: 4915)
  CGroup: /system.slice/rsyslog.service
          └─832 /usr/sbin/rsyslogd -n -iNONE
```

openSUSE ships with a dedicated configuration file for remote logging: `/etc/rsyslog.d/remote.conf`. Let us enable receiving messages from clients (remote hosts) via TCP. We must uncomment the lines which load the module and start the TCP server on port 514:

```
# ##### Receiving Messages from Remote Hosts #####
# TCP Syslog Server:
# provides TCP syslog reception and GSS-API (if compiled to support it)
$ModLoad imtcp.so # load module
##$UDPServerAddress 10.10.0.1 # force to listen on this IP only
$InputTCPServerRun 514 # Starts a TCP server on selected port

# UDP Syslog Server:
#$ModLoad imudp.so # provides UDP syslog reception
##$UDPServerAddress 10.10.0.1 # force to listen on this IP only
#$UDPServerRun 514 # start a UDP syslog server at standard port 514
```

Once this is done, we must restart the rsyslog service and check that the server is listening on port 514:

```
root@suse-server:~# systemctl restart rsyslog
root@suse-server:~# netstat -nltp | grep 514
[sudo] password for root:
tcp        0      0 0.0.0.0:514          0.0.0.0:*           LISTEN
2263/rsyslogd
tcp6      0      0 :::514              :::*                 LISTEN
2263/rsyslogd
```

Next, we should open the ports in the firewall and reload the configuration:

```
root@suse-server:~# firewall-cmd --permanent --add-port 514/tcp
success
root@suse-server:~# firewall-cmd --reload
success
```

#### NOTE

With the arrival of openSUSE Leap 15.0, `firewalld` replaced the classic `SuSEfirewall2` completely.

## Templates and Filter Conditions

By default, the client's logs will be written to the server's `/var/log/messages` file—together with those of the server themselves. However, we will create a *template* and a *filter condition* to have our client's logs stored in clear-cut directories of their own. To do so, we will add the following to `/etc/rsyslog.conf` (or `/etc/rsyslog.d/remote.conf`):

```
$template RemoteLogs, "/var/log/remotehosts/%HOSTNAME%/%$NOW%.%syslogseverity-text%.log"
if $FROMHOST-IP=='192.168.1.4' then ?RemoteLogs
& stop
```

## Template

The template corresponds to the first line and allows you to specify a format for log names using dynamic file name generation. A template consists of:

- Template directive (`$template`)
- Template name (`RemoteLogs`)
- Template text (`"/var/log/remotehosts/%HOSTNAME%/%$NOW%.%syslogseverity-text%.log"`)
- Options (optional)

Our template is called `RemoteLogs` and its text consists of a path in `/var/log`. All of our remote host's logs will go into the `remotehosts` directory, where a subdirectory will be created based on the machine's hostname (`%HOSTNAME%`). Each filename in this directory will consist of the date (`%$NOW%`), the severity (aka priority) of the message in text format (`%syslogseverity-text%`) and the `.log` suffix. The words between percent signs are *properties* and allow you access to the contents of the log message (date, priority, etc.). A `syslog` message has a number of well-defined properties which can be used in templates. These properties are accessed—and can be modified—by the so-called *property replacer* which implies writing them between percent signs.

## Filter Condition

The remaining two lines correspond to the filter condition and its associated action:

- Expression-Based Filter (`if $FROMHOST-IP=='192.168.1.4'`)
- Action (`then ?RemoteLogs, & stop`)

The first line checks the IP address of the remote host sending the log and—if it equals that of our Debian client—it applies the `RemoteLogs` template. The last line (`& stop`) guarantees that messages are not being sent simultaneously to `/var/log/messages` (but only to files in the `/var/log/remotehosts` directory).

### NOTE

To learn more about templates, properties and rules, you can consult the manual page for `rsyslog.conf`.

With the configuration updated, we will restart `rsyslog` again and confirm there is no `remotehosts` directory in `/var/log` yet:

```

root@suse-server:~# systemctl restart rsyslog
root@suse-server:~# ls /var/log/
acpid          chronty    localmessages  pbl.log      Xorg.0.log
alternatives.log cups       mail           pk_backend_zypp Xorg.0.log.old
apparmor      firebird  mail.err       samba        YaST2
audit        firewall  mail.info     snapper.log  zypp
boot.log     firewalld mail.warn     tallylog     zypper.log
boot.msg     krb5      messages     tuned
boot.omsg   lastlog  mysql        warn
btmpt      lightdm  NetworkManager wtmp

```

The server has now been configured. Next, we will configure the client.

Again, we must ensure that `rsyslog` is installed and running:

```

root@debian:~# sudo systemctl status rsyslog
rsyslog.service - System Logging Service
   Loaded: loaded (/lib/systemd/system/rsyslog.service; enabled; vendor preset:
   Active: active (running) since Thu 2019-09-17 18:47:54 CEST; 7min ago
     Docs: man:rsyslogd(8)
           http://www.rsyslog.com/doc/
   Main PID: 351 (rsyslogd)
     Tasks: 4 (limit: 4915)
   CGroup: /system.slice/rsyslog.service
           └─351 /usr/sbin/rsyslogd -n

```

In our sample environment we have implemented name resolution on the client by adding the line `192.168.1.6 suse-server` to `/etc/hosts`. Thus, we can refer to the server by either name (`suse-server`) or IP address (`192.168.1.6`).

Our Debian client does not come with a `remote.conf` file in `/etc/rsyslog.d/`, so we will apply our configurations in `/etc/rsyslog.conf`. We will write the following line at the end of the file:

```
*.* @@suse-server:514
```

Finally, we restart `rsyslog`.

```
root@debian:~# systemctl restart rsyslog
```

Now, let us go back to our `suse-server` machine and check for the existence of `remotehosts` in

`/var/log:`

```
root@suse-server:~# ls /var/log/remotehosts/debian/
2019-09-17.info.log  2019-09-17.notice.log
```

We already have two logs inside `/var/log/remotehosts` as described in our template. To complete this section, we run `tail -f 2019-09-17.notice.log` on `suse-server` while we send a log *manually* from our Debian client and confirm that messages are appended to the log file as expected (the `-t` option supplies a tag for our message):

```
root@suse-server:~# tail -f /var/log/remotehosts/debian/2019-09-17.notice.log
2019-09-17T20:57:42+02:00 debian dbus[323]: [system] Successfully activated service
'org.freedesktop.nm_dispatcher'
2019-09-17T21:01:41+02:00 debian anacron[1766]: Anacron 2.3 started on 2019-09-17
2019-09-17T21:01:41+02:00 debian anacron[1766]: Normal exit (0 jobs run)
```

```
carol@debian:~$ logger -t DEBIAN-CLIENT Hi from 192.168.1.4
```

```
root@suse-server:~# tail -f /var/log/remotehosts/debian/2019-09-17.notice.log
2019-09-17T20:57:42+02:00 debian dbus[323]: [system] Successfully activated service
'org.freedesktop.nm_dispatcher'
2019-09-17T21:01:41+02:00 debian anacron[1766]: Anacron 2.3 started on 2019-09-17
2019-09-17T21:01:41+02:00 debian anacron[1766]: Normal exit (0 jobs run)
2019-09-17T21:04:21+02:00 debian DEBIAN-CLIENT: Hi from 192.168.1.4
```

## The Log Rotation Mechanism

Logs are rotated on a regular basis, which serves two main purposes:

- Prevent older log files from using more disk space than necessary.
- Keep logs to a manageable length for ease of consultation.

The utility in charge of log rotation (or cycling) is `logrotate` and its job includes actions such as moving log files to a new name, archiving and/or compressing them, sometimes emailing them to the system administrator and eventually deleting them as they grow old. There are various conventions for naming these rotated log files (adding a suffix with the date to the filename, for example); however, simply adding a suffix with an integer is the common practice:

```
root@debian:~# ls /var/log/messages*
/var/log/messages /var/log/messages.1 /var/log/messages.2.gz /var/log/messages.3.gz
/var/log/messages.4.gz
```

Let us now explain what will happen in the next log rotation:

1. `messages.4.gz` will be deleted and lost.
2. The content of `messages.3.gz` will be moved to `messages.4.gz`.
3. The content of `messages.2.gz` will be moved to `messages.3.gz`.
4. The content of `messages.1` will be moved to `messages.2.gz`.
5. The content of `messages` will be moved to `messages.1` and `messages` will be empty and ready to register new log entries.

Note how — according to `logrotate` directives that you will see shortly — the three older log files are compressed, whereas the two most recent ones are not. Also, we will be retaining the logs from the last 4-5 weeks. To read messages that are 1 week old, we will consult `messages.1` (and so on).

`logrotate` is run as an automated process or cron job daily through the script `/etc/cron.daily/logrotate` and reads the configuration file `/etc/logrotate.conf`. This file includes some global options and is well commented with each option introduced by a brief explanation of its purpose:

```
carol@debian:~$ sudo less /etc/logrotate.conf
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# uncomment this if you want your log files compressed
#compress

# packages drop log rotation information into this directory
include /etc/logrotate.d
```

(...)

As you can see, the configuration files in `/etc/logrotate.d` for specific packages are also included. These files contain—for the most part—local definitions and specify the logfiles to rotate (remember, local definitions take precedence over global ones, and later definitions override earlier ones). What follows is an excerpt of a definition in `/etc/logrotate.d/rsyslog`:

```
/var/log/messages
{
    rotate 4
    weekly
    missingok
    notifempty
    compress
    delaycompress
    sharedscripts
    postrotate
        invoke-rc.d rsyslog rotate > /dev/null
    endscript
}
```

As you can see, every directive is separated from its value by whitespace and/or an optional equals sign (=). The lines between `postrotate` and `endscript` must appear on lines by themselves, though. The explanation is as follows:

#### **rotate 4**

Retain 4 weeks worth of logs.

#### **weekly**

Rotate log files weekly.

#### **missingok**

Do not issue an error message if the log file is missing; just go on to the next one.

#### **notifempty**

Do not rotate the log if it is empty.

#### **compress**

Compress log files with `gzip` (default).

## delaycompress

Postpone compression of the previous log file to the next rotation cycle (only effective when used in combination with `compress`). It is useful when a program cannot be told to close its log file and thus might continue writing to the previous log file for some time.

## sharedscripts

Related to `prerotate` and `postrotate` scripts. To prevent a script from being executed multiple times, run the scripts only once regardless of how many log files match a given pattern (e.g. `/var/log/mail/*`). The scripts will not be run if none of the logs in the pattern require rotating, though. On top of that, if the scripts exit with errors, any remaining actions will not be executed for any logs.

## postrotate

Indicate the beginning of a `postrotate` script.

## invoke-rc.d rsyslog rotate > /dev/null

Use `/bin/sh` to run `invoke-rc.d rsyslog rotate > /dev/null` after rotating the logs.

## endscript

Indicate the end of the `postrotate` script.

### NOTE

For a complete list of directives and explanations, consult the manual page for `logrotate.conf`.

## The Kernel Ring Buffer

Since the kernel generates several messages before `rsyslogd` becomes available on boot, a mechanism to register those messages becomes necessary. This is where the *kernel ring buffer* comes into play. It is a fixed-size data structure and — therefore — as it grows with new messages, the oldest will disappear.

The `dmesg` command prints the kernel ring buffer. Because of the buffer's size, this command is normally used in combination with the text filtering utility `grep`. For instance, to search for messages related to Universal Serial Bus devices:

```
root@debian:~# dmesg | grep "usb"
[ 1.241182] usbcore: registered new interface driver usbfs
[ 1.241188] usbcore: registered new interface driver hub
[ 1.250968] usbcore: registered new device driver usb
[ 1.339754] usb usb1: New USB device found, idVendor=1d6b, idProduct=0001, bcdDevice=
4.19
```



```
[ 1.339756] usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1  
(...)
```

## Guided Exercises

1. What utilities/commands would you use in the following scenarios:

Purpose and log file	Utility
Read <code>/var/log/syslog.7.gz</code>	
Read <code>/var/log/syslog</code>	
Filter for the word <code>renewal</code> in <code>/var/log/syslog</code>	
Read <code>/var/log/faillog</code>	
Read <code>/var/log/syslog</code> dynamically	

2. Rearrange the following log entries in such a way that they represent a valid log message with the proper structure:

- `debian-server`
- `sshd`
- `[515]:`
- `Sep 13 21:47:56`
- `Server listening on 0.0.0.0 port 22`

The correct order is:

3. What rules would you add to `/etc/rsyslog.conf` in order to accomplish each of the following:

- Send all messages from the `mail` facility and a priority/severity of `crit` (and above) to `/var/log/mail.crit`:

- Send all messages from the `mail` facility with priorities of `alert` and `emergency` to `/var/log/mail.urgent`:

- Except for those coming from the `cron` and `ntp` facilities, send all messages — irrespective of their facility and priority — to `/var/log/allmessages`:

- With all required settings properly configured first, send all messages from the `mail` facility to a remote host whose IP address is `192.168.1.88` using TCP and specifying the default port:

---

- Irrespective of their facility, send all messages with the `warning` priority (*only with the warning priority*) to `/var/log/warnings` preventing excessive writing to the disk:

---

4. Consider following stanza from `/etc/logrotate.d/samba` and explain the different options:

```
carol@debian:~$ sudo head -n 11 /etc/logrotate.d/samba
/var/log/samba/log.smbd {
    weekly
    missingok
    rotate 7
    postrotate
        [ ! -f /var/run/samba/smbd.pid ] || /etc/init.d/smbd reload > /dev/null
    endscript
    compress
    delaycompress
    notifempty
}
```

Option	Meaning
<code>weekly</code>	
<code>missingok</code>	
<code>rotate 7</code>	
<code>postrotate</code>	
<code>endscript</code>	
<code>compress</code>	
<code>delaycompress</code>	
<code>notifyempty</code>	

## Explorational Exercises

1. In section “Templates and Filter Conditions” we used an *expression-based filter* as a filter condition. *Property-based filters* are another type of filter unique to `rsyslogd`. Translate our *expression-based filter* into a *property-based filter*:

Expression-Based Filter	Property-Based Filter
<pre>if \$FROMHOST-IP=='192.168.1.4' then ?RemoteLogs</pre>	

2. `omusrmsg` is an `rsyslog` built-in module which facilitates notifying users (it sends log messages to the user terminal). Write a rule to send all *emergency* messages of all facilities to both `root` and the regular user `carol`.

# Summary

In this lesson you learned:

- Logging is crucial for system administration.
- `rsyslogd` is the utility in charge of keeping logs neat and orderly.
- Some services take care of their own logs.
- Roughly speaking, logs can be classified into system logs and service/program logs.
- There are a number of utilities which are convenient for log reading: `less`, `more`, `zless`, `zmore`, `grep`, `head` and `tail`.
- Most log files are plain-text files; however, there are a small number of binary log files.
- Log-wise, `rsyslogd` receives the relevant information from special files (sockets and memory buffers) before processing it.
- To classify logs, `rsyslogd` uses rules in `/etc/rsyslog.conf` or `/etc/rsyslog.d/*`.
- Any user may enter their own messages into the system log manually with the `logger` utility.
- `rsyslog` allows you to keep all logs across IP networks in a centralized log server.
- Templates come in handy for formatting log file names dynamically.
- The purpose of log rotation is twofold: preventing old logs from using excessive disk space and making consulting logs manageable.

# Answers to Guided Exercises

1. What utilities/commands would you use in the following scenarios:

Purpose and log file	Utility
Read <code>/var/log/syslog.7.gz</code>	<code>zmore</code> or <code>zless</code>
Read <code>/var/log/syslog</code>	<code>more</code> or <code>less</code>
Filter for the word <code>renewal</code> in <code>/var/log/syslog</code>	<code>grep</code>
Read <code>/var/log/faillog</code>	<code>faillog -a</code>
Read <code>/var/log/syslog</code> dynamically	<code>tail -f</code>

2. Rearrange the following log entries in such a way that they represent a valid log message with the proper structure:

- `debian-server`
- `sshd`
- `[515]:`
- `Sep 13 21:47:56`
- `Server listening on 0.0.0.0 port 22`

The correct order is:

```
Sep 13 21:47:56 debian-server sshd[515]: Server listening on 0.0.0.0 port 22
```

3. What rules would you add to `/etc/rsyslog.conf` in order to accomplish each of the following:

- Send all messages from the `mail` facility and a priority/severity of `crit` (and above) to `/var/log/mail.crit`:

```
mail.crit                /var/log/mail.crit
```

- Send all messages from the `mail` facility with priorities of `alert` and `emergency` to `/var/log/mail.urgent`:

```
mail.alert /var/log/mail.urgent
```

- Except for those coming from the `cron` and `ntp` facilities, send all messages — irrespective of their facility and priority — to `/var/log/allmessages`:

```
*.*;cron.none;ntp.none /var/log/allmessages
```

- With all required settings properly configured first, send all messages from the `mail` facility to a remote host whose IP address is `192.168.1.88` using TCP and specifying the default port:

```
mail.* @@192.168.1.88:514
```

- Irrespective of their facility, send all messages with the `warning` priority (*only with the warning priority*) to `/var/log/warnings` preventing excessive writing to the disk:

```
*.=warning -/var/log/warnings
```

#### 4. Consider following stanza from `/etc/logrotate.d/samba` and explain the different options:

```
carol@debian:~$ sudo head -n 11 /etc/logrotate.d/samba
/var/log/samba/log.smbd {
    weekly
    missingok
    rotate 7
    postrotate
        [ ! -f /var/run/samba/smbd.pid ] || /etc/init.d/smbd reload > /dev/null
    endscript
    compress
    delaycompress
    notifempty
}
```

Option	Meaning
<code>weekly</code>	Rotate log files on a weekly basis.
<code>missingok</code>	Do not issue an error message if the log is missing; just continue to the next one.

Option	Meaning
rotate 7	Keep 7 weeks worth of backlogs.
postrotate	Run the script on the following line after rotating the logs.
endscript	Indicate the end of the <i>postrotate</i> script.
compress	Compress the logs with gzip.
delaycompress	In combination with <code>compress</code> , postpone compression to the next rotation cycle.
notifyempty	Do not rotate the log if it is empty.



## Answers to Explorational Exercises

- In section “Templates and Filter Conditions” we used an *expression-based filter* as a filter condition. *Property-based filters* are another type of filter unique to `rsyslogd`. Translate our *expression-based filter* into a *property-based filter*:

Expression-Based Filter	Property-Based Filter
if \$FROMHOST-IP=='192.168.1.4' then ?RemoteLogs	:fromhost-ip, isequal, "192.168.1.4" ?RemoteLogs

- `omusrmsg` is an `rsyslog` built-in module which facilitates notifying users (it sends log messages to the user terminal). Write a rule to send all *emergency* messages of all facilities to both `root` and the regular user `carol`.

```
*.emerg                                :omusrmsg:root,carol
```



## 108.2 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	108 Essential System Services
<b>Objective:</b>	108.2 System logging
<b>Lesson:</b>	2 of 2

### Introduction

With the general adoption of `systemd` by all major distributions, the journal daemon (`systemd-journald`) has become the standard logging service. In this lesson we will discuss how it operates and how you can use it to do a number of things: query it, filter its information by different criteria, configure its storage and size, delete old data, retrieve its data from a rescue system or filesystem copy and — last but not least — understand its interaction with `rsyslogd`.

### Basics of `systemd`

First introduced in Fedora, `systemd` has progressively replaced SysV Init as the *de facto* system and service manager in most major Linux distributions. Amongst its strengths are the following:

- Ease of configuration: unit files as opposed to SysV Init scripts.
- Versatile management: apart from daemons and processes, it also manages devices, sockets and mount points.
- Backward compatibility with both SysV Init and Upstart.
- Parallel loading during boot-up: services are loaded in parallel as opposed to Sysv Init loading

them sequentially.

- It features a logging service called the *journal* that presents the following advantages:
  - It centralizes all logs in one place.
  - It does not require log rotation.
  - Logs can be disabled, loaded in RAM or made persistent.

## Units and Targets

`systemd` operates on *units*. A unit is any resource that `systemd` can manage (e.g. network, bluetooth, etc.). Units — in turn — are governed by *unit files*. These are plain text files which live in `/lib/systemd/system` and include the configuration settings—in the form of *sections* and *directives*—for a particular resource to be managed. There are a number of unit types: `service`, `mount`, `automount`, `swap`, `timer`, `device`, `socket`, `path`, `timer`, `snapshot`, `slice`, `scope` and `target`. Thus, every unit file name follows the pattern `<resource_name>.<unit_type>` (e.g. `reboot.service`).

A *target* is a special type of unit which resembles the classic runlevels of SysV Init. This is because a *target unit* brings together various resources to represent a particular system state (e.g. `graphical.target` is similar to `runlevel 5`, etc.). To check the current target in your system, use the `systemctl get-default` command:

```
carol@debian:~$ systemctl get-default
graphical.target
```

On the other hand, targets and runlevels differ in that the former are mutually inclusive, whereas the latter are not. Thus a target can bring up other targets — which is not possible with runlevels.

**NOTE** An explanation of how `systemd` units work is beyond the scope of this lesson.

## The System Journal: `systemd-journald`

`systemd-journald` is the system service which takes care of receiving logging information from a variety of sources: kernel messages, simple and structured system messages, standard output and standard error of services as well as audit records from the kernel audit subsystem (for further details see the manual page for `systemd-journald`). Its mission is that of creating and maintaining a structured and indexed journal.

Its configuration file is `/etc/systemd/journald.conf` and—as with any other service—you can use the `systemctl` command to *start* it, *restart* it, *stop* it or—simply—check its *status*:

```

root@debian:~# systemctl status systemd-journald
systemd-journald.service - Journal Service
  Loaded: loaded (/lib/systemd/system/systemd-journald.service; static; vendor preset:
enabled)
  Active: active (running) since Sat 2019-10-12 13:43:06 CEST; 5min ago
    Docs: man:systemd-journald.service(8)
          man:journald.conf(5)
 Main PID: 178 (systemd-journal)
  Status: "Processing requests..."
   Tasks: 1 (limit: 4915)
  CGroup: /system.slice/systemd-journald.service
          └─178 /lib/systemd/systemd-journald
 (...)

```

Configuration files of the type `journal.conf.d/*.conf`—which can include package-specific configurations—are also possible (consult the manual page of `journald.conf` to learn more).

If enabled, the journal can be stored either persistently on disk or in a volatile manner on a RAM-based filesystem. The journal is not a plain text file, it is binary. Thus, you cannot use text parsing tools such as `less` or `more` to read its contents; the `journalctl` command is used instead.

## Querying the Journal Content

`journalctl` is the utility that you use to query `systemd` journal. You have to either be root or use `sudo` to invoke it. If queried without options, it will print the entire journal chronologically (with the oldest entries listed first):

```

root@debian:~# journalctl
-- Logs begin at Sat 2019-10-12 13:43:06 CEST, end at Sat 2019-10-12 14:19:46 CEST. --
Oct 12 13:43:06 debian kernel: Linux version 4.9.0-9-amd64 (debian-kernel@lists.debian.org)
 (...)
Oct 12 13:43:06 debian kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64
root=UUID=b6be6117-5226-4a8a-bade-2db35ccf4cf4 ro qu
 (...)

```

You can make more specific queries by using a number of switches:

### **-r**

The messages of the journal will be printed in reverse order:

```

root@debian:~# journalctl -r

```

```
-- Logs begin at Sat 2019-10-12 13:43:06 CEST, end at Sat 2019-10-12 14:30:30 CEST. --
Oct 12 14:30:30 debian sudo[1356]: pam_unix(sudo:session): session opened for user root
by carol(uid=0)
Oct 12 14:30:30 debian sudo[1356]:    carol : TTY=pts/0 ; PWD=/home/carol ; USER=root ;
COMMAND=/bin/journalctl -r
Oct 12 14:19:53 debian sudo[1348]: pam_unix(sudo:session): session closed for user root
(...)
```

**-f**

It will print the most recent journal messages and keep printing new entries as they are appended to the journal — much like `tail -f`:

```
root@debian:~# journalctl -f
-- Logs begin at Sat 2019-10-12 13:43:06 CEST. --
(...)
Oct 12 14:44:42 debian sudo[1356]: pam_unix(sudo:session): session closed for user root
Oct 12 14:44:44 debian sudo[1375]:    carol : TTY=pts/0 ; PWD=/home/carol ; USER=root ;
COMMAND=/bin/journalctl -f
Oct 12 14:44:44 debian sudo[1375]: pam_unix(sudo:session): session opened for user root
by carol(uid=0)
(...)
```

**-e**

It will jump to the end of the journal so that the latest entries are visible within the pager:

```
root@debian:~# journalctl -e
(...)
Oct 12 14:44:44 debian sudo[1375]:    carol : TTY=pts/0 ; PWD=/home/carol ; USER=root ;
COMMAND=/bin/journalctl -f
Oct 12 14:44:44 debian sudo[1375]: pam_unix(sudo:session): session opened for user root
by carol(uid=0)
Oct 12 14:45:57 debian sudo[1375]: pam_unix(sudo:session): session closed for user root
Oct 12 14:48:39 debian sudo[1378]:    carol : TTY=pts/0 ; PWD=/home/carol ; USER=root ;
COMMAND=/bin/journalctl -e
Oct 12 14:48:39 debian sudo[1378]: pam_unix(sudo:session): session opened for user root
by carol(uid=0)
```

**-n <value>, --lines=<value>**

It will print the *value* most recent lines (if no `<value>` is specified, it defaults to 10):

```

root@debian:~# journalctl -n 5
(...)
Oct 12 14:44:44 debian sudo[1375]:    carol : TTY=pts/0 ; PWD=/home/carol ; USER=root ;
COMMAND=/bin/journalctl -f
Oct 12 14:44:44 debian sudo[1375]: pam_unix(sudo:session): session opened for user root
by carol(uid=0)
Oct 12 14:45:57 debian sudo[1375]: pam_unix(sudo:session): session closed for user root
Oct 12 14:48:39 debian sudo[1378]:    carol : TTY=pts/0 ; PWD=/home/carol ; USER=root ;
COMMAND=/bin/journalctl -e
Oct 12 14:48:39 debian sudo[1378]: pam_unix(sudo:session): session opened for user root
by carol(uid=0)

```

## **-k, --dmesg**

Equivalent to using the `dmesg` command:

```

root@debian:~# journalctl -k
-- Logs begin at Sat 2019-10-12 13:43:06 CEST, end at Sat 2019-10-12 14:53:20 CEST. --
Oct 12 13:43:06 debian kernel: Linux version 4.9.0-9-amd64 (debian-
kernel@lists.debian.org) (gcc version 6.3.0 20170516 (Debian 6.3.0-18
Oct 12 13:43:06 debian kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64
root=UUID=b6be6117-5226-4a8a-bade-2db35ccf4cf4 ro qu
Oct 12 13:43:06 debian kernel: x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating
point registers'
Oct 12 13:43:06 debian kernel: x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
(...)

```

## **Navigating and Searching Through the Journal**

You can navigate the journal's output with:

- PageUp, PageDown and arrow keys to move up, down, left and right.
- `>` to go to the end of the output.
- `<` to go to the beginning of the output.

You can search for strings both forward and backward from your current view position:

- Forward search: Press `/` and enter the string to be searched, then press Enter.
- Backward search: Press `?` and enter the string to be searched, then press Enter.

To navigate through matches in the searches, use `N` to go to the next occurrence of the match and

`Shift` + `N` to go to the previous one.

## Filtering the Journal Data

The journal allows you to filter log data by different criteria:

### Boot number

#### `--list-boots`

It lists all available boots. The output consists of three columns; the first one specifies the boot number (`0` refers to the current boot, `-1` is the previous one, `-2` the one prior to the previous one and so on); the second column is the boot ID; the third shows the time stamps:

```
root@debian:~# journalctl --list-boots
0 83df3e8653474ea5aed19b41cdb45b78 Sat 2019-10-12 18:55:41 CEST–Sat 2019-10-12
19:02:24 CEST
```

#### `-b, --boot`

It shows all messages from the current boot. To see log messages from previous boots, just add an offset parameter as explained above. For example, to have messages from the previous boot printed, you will type `journalctl -b -1`. Remember, though, that to recover information from previous logs, persistence of the journal must be enabled (you will learn how to do so in the next section):

```
root@debian:~# journalctl -b -1
Specifying boot ID has no effect, no persistent journal was found
```

### Priority

#### `-p`

Interestingly enough, you can also filter by severity/priority with the `-p` option:

```
root@debian:~# journalctl -b -0 -p err
-- No entries --
```

The journal informs us that — so far — there have not been any messages with a priority of error (or above) from the current boot. Note: `-b -0` can be omitted when referring to the current boot.

#### NOTE

Refer to the previous lesson for a complete list of all `syslog` severities (aka

priorities).

## Time Interval

You can have `journalctl` print only the messages logged within a specific time frame by using the `--since` and `--until` switches. The date specification should follow the format `YYYY-MM-DD HH:MM:SS`. Midnight will be assumed if we omit the time component. By the same token, if the date is omitted, the current day is assumed. For instance, to see messages logged from 7:00pm to 7:01pm, you will type:

```
root@debian:~# journalctl --since "19:00:00" --until "19:01:00"
-- Logs begin at Sat 2019-10-12 18:55:41 CEST, end at Sat 2019-10-12 20:10:50 CEST. --
Oct 12 19:00:14 debian systemd[1]: Started Run anacron jobs.
Oct 12 19:00:14 debian anacron[1057]: Anacron 2.3 started on 2019-10-12
Oct 12 19:00:14 debian anacron[1057]: Normal exit (0 jobs run)
Oct 12 19:00:14 debian systemd[1]: anacron.timer: Adding 2min 47.988096s random time.
```

Likewise you can use a slightly different time specification: `"integer time-unit ago"`. Thus, to see messages logged two minutes ago you will type `sudo journalctl --since "2 minutes ago"`. It is also possible to use `+` and `-` to specify times relative to the current time so `--since "-2 minutes"` and `--since "2 minutes ago"` are equivalent.

Apart from numeric expressions, you can specify a number of keywords:

### yesterday

As of midnight of the day before the current day.

### today

As of midnight of the current day.

### tomorrow

As of midnight of the day after the current day.

### now

The current time.

Let us see all messages since last midnight until today at 21:00pm:

```
root@debian:~# journalctl --since "today" --until "21:00:00"
-- Logs begin at Sat 2019-10-12 20:45:29 CEST, end at Sat 2019-10-12 21:06:15 CEST. --
Oct 12 20:45:29 debian sudo[1416]:    carol : TTY=pts/0 ; PWD=/home/carol ; USER=root
; COMMAND=/bin/systemctl r
```



```
Oct 12 20:45:29 debian sudo[1416]: pam_unix(sudo:session): session opened for user
root by carol(uid=0)
Oct 12 20:45:29 debian systemd[1]: Stopped Flush Journal to Persistent Storage.
(...)
```

**NOTE**

To learn more about the different syntaxes for time specifications, consult the manual page `systemd.time`.

**Program**

To see journal messages related to a specific executable the following syntax is used: `journalctl /path/to/executable`:

```
root@debian:~# journalctl /usr/sbin/sshd
-- Logs begin at Sat 2019-10-12 20:45:29 CEST, end at Sat 2019-10-12 21:54:49 CEST. --
Oct 12 21:16:28 debian sshd[1569]: Accepted password for carol from 192.168.1.65 port
34050 ssh2
Oct 12 21:16:28 debian sshd[1569]: pam_unix(sshd:session): session opened for user carol
by (uid=0)
Oct 12 21:16:54 debian sshd[1590]: Accepted password for carol from 192.168.1.65 port
34052 ssh2
Oct 12 21:16:54 debian sshd[1590]: pam_unix(sshd:session): session opened for user carol
by (uid=0)
```

**Unit**

Remember, a unit is any resource handled by `systemd` and you can filter by them, too.

**-u**

It shows messages about a specified unit:

```
root@debian:~# journalctl -u ssh.service
-- Logs begin at Sun 2019-10-13 10:50:59 CEST, end at Sun 2019-10-13 12:22:59 CEST. --
Oct 13 10:51:00 debian systemd[1]: Starting OpenBSD Secure Shell server...
Oct 13 10:51:00 debian sshd[409]: Server listening on 0.0.0.0 port 22.
Oct 13 10:51:00 debian sshd[409]: Server listening on :: port 22.
(...)
```

**NOTE**

To print all loaded and active units, use `systemctl list-units`; to see all installed unit files use `systemctl list-unit-files`.

## Fields

The journal can also be filtered by specific *fields* through any of the following syntaxes:

- `<field-name>=<value>`
- `_<field-name>=<value>_`
- `__<field-name>=<value>`

### **PRIORITY=**

One of eight possible `syslog` priority values formatted as a decimal string:

```
root@debian:~# journalctl PRIORITY=3
-- Logs begin at Sun 2019-10-13 10:50:59 CEST, end at Sun 2019-10-13 14:30:50 CEST.
--
Oct 13 10:51:00 debian avahi-daemon[314]: chroot.c: open() failed: No such file or
directory
```

Note how you could have achieved the same output by using the command `sudo journalctl -p err` that we saw above.

### **SYSLOG\_FACILITY=**

Any of the possible facility code numbers formatted as a decimal string. For example, to see all user-level messages:

```
root@debian:~# journalctl SYSLOG_FACILITY=1
-- Logs begin at Sun 2019-10-13 10:50:59 CEST, end at Sun 2019-10-13 14:42:52 CEST.
--
Oct 13 10:50:59 debian mtp-probe[227]: checking bus 1, device 2:
"/sys/devices/pci0000:00/0000:00:06.0/usb1/1-1"
Oct 13 10:50:59 debian mtp-probe[227]: bus: 1, device: 2 was not an MTP device
Oct 13 10:50:59 debian mtp-probe[238]: checking bus 1, device 2:
"/sys/devices/pci0000:00/0000:00:06.0/usb1/1-1"
Oct 13 10:50:59 debian mtp-probe[238]: bus: 1, device: 2 was not an MTP device
```

### **\_PID=**

Show messages produced by a specific process ID. To see all messages produced by `systemd`, you would type:

```
root@debian:~# journalctl _PID=1
-- Logs begin at Sun 2019-10-13 10:50:59 CEST, end at Sun 2019-10-13 14:50:15 CEST.
```

```
--
Oct 13 10:50:59 debian systemd[1]: Mounted Debug File System.
Oct 13 10:50:59 debian systemd[1]: Mounted POSIX Message Queue File System.
Oct 13 10:50:59 debian systemd[1]: Mounted Huge Pages File System.
Oct 13 10:50:59 debian systemd[1]: Started Remount Root and Kernel File Systems.
Oct 13 10:50:59 debian systemd[1]: Starting Flush Journal to Persistent Storage...
(...)
```

### **`_BOOT_ID`**

Based on its boot ID you can single out the messages from a specific boot, for example:  
`sudo journalctl _BOOT_ID=83df3e8653474ea5aed19b41cdb45b78.`

### **`_TRANSPORT`**

Show messages received from a specific transport. Possible values are: `audit` (kernel audit subsystem), `driver` (generated internally), `syslog` (syslog socket), `journal` (native journal protocol), `stdout` (services' standard output or standard error), `kernel` (kernel ring buffer — the same as `dmesg`, `journalctl -k` or `journalctl --dmesg`):

```
root@debian:~# journalctl _TRANSPORT=journal
-- Logs begin at Sun 2019-10-13 20:19:58 CEST, end at Sun 2019-10-13 20:46:36 CEST.
--
Oct 13 20:19:58 debian systemd[1]: Started Create list of required static device
nodes for the current kernel.
Oct 13 20:19:58 debian systemd[1]: Starting Create Static Device Nodes in /dev...
Oct 13 20:19:58 debian systemd[1]: Started Create Static Device Nodes in /dev.
Oct 13 20:19:58 debian systemd[1]: Starting udev Kernel Device Manager...
(...)
```

## **Combining Fields**

Fields are not mutually exclusive so you can use more than one in the same query. However, only messages that match the value of both fields simultaneously will be shown:

```
root@debian:~# journalctl PRIORITY=3 SYSLOG_FACILITY=0
-- No entries --
root@debian:~# journalctl PRIORITY=4 SYSLOG_FACILITY=0
-- Logs begin at Sun 2019-10-13 20:19:58 CEST, end at Sun 2019-10-13 20:21:55 CEST. --
Oct 13 20:19:58 debian kernel: acpi PNP0A03:00: fail to add MMCONFIG information, can't
access extended PCI configuration (...)
```

Unless you use the `+` separator to combine two expressions in the manner of a logical *OR*:

```
root@debian:~# journalctl PRIORITY=3 + SYSLOG_FACILITY=0
-- Logs begin at Sun 2019-10-13 20:19:58 CEST, end at Sun 2019-10-13 20:24:02 CEST. --
Oct 13 20:19:58 debian kernel: Linux version 4.9.0-9-amd64 (debian-kernel@lists.debian.org)
(...9
Oct 13 20:19:58 debian kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64
root=UUID= (...)
(...)
```

On the other hand, you can supply two values for the same field and all entries matching either value will be shown:

```
root@debian:~# journalctl PRIORITY=1
-- Logs begin at Sun 2019-10-13 17:16:24 CEST, end at Sun 2019-10-13 17:30:14 CEST. --
-- No entries --
root@debian:~# journalctl PRIORITY=1 PRIORITY=3
-- Logs begin at Sun 2019-10-13 17:16:24 CEST, end at Sun 2019-10-13 17:32:12 CEST. --
Oct 13 17:16:27 debian connmand[459]: __connman_inet_get_pnp_nameservers: Cannot read /pro
Oct 13 17:16:27 debian connmand[459]: The name net.connman.vpn was not provided by any .se
```

#### NOTE

Journal fields fall in any of the following categories: “User Journal Fields”, “Trusted Journal Fields”, “Kernel Journal Fields”, “Fields on behalf of a different program” and “Address Fields”. For more information on this topic—including a complete list of fields—see the `man` page for `systemd.journal-fields(7)`.

## Manual Entries in the System Journal: `systemd-cat`

Just like how the `logger` command is used to send messages from the command line to the system log (as we saw in the previous lesson), the `systemd-cat` command serves a similar—but more well-rounded—purpose with the system journal. It allows us to send standard input (*stdin*), output (*stdout*) and error (*stderr*) to the journal.

If invoked with no parameters, it will send everything it reads from *stdin* to the journal. Once you are done, press `Ctrl` + `C`.

```
carol@debian:~$ systemd-cat
This line goes into the journal.
^C
```

If it is passed the output of a piped command, this will be sent to the journal too:

```
carol@debian:~$ echo "And so does this line." | systemd-cat
```

If followed by a command, that command's output will also be sent to the journal — together with *stderr* (if any):

```
carol@debian:~$ systemd-cat echo "And so does this line too."
```

There is also the possibility of specifying a priority level with the `-p` option:

```
carol@debian:~$ systemd-cat -p emerg echo "This is not a real emergency."
```

Refer to `systemd-cat` man page to learn about its other options.

To see the last four lines in the journal:

```
carol@debian:~$ journalctl -n 4
(...)
-- Logs begin at Sun 2019-10-20 13:43:54 CEST. --
Nov 13 23:14:39 debian cat[1997]: This line goes into the journal.
Nov 13 23:19:16 debian cat[2027]: And so does this line.
Nov 13 23:23:21 debian echo[2030]: And so does this line too.
Nov 13 23:26:48 debian echo[2034]: This is not a real emergency.
```

#### NOTE

Journal entries with a priority level of *emergency* will be printed in bold red on most systems.

## Persistent Journal Storage

As mentioned previously, you have three options when it comes to the location of the journal:

- Journaling can be turned off altogether (redirection to other facilities such as the console are still possible, though).
- Keep it in memory — which makes it volatile — and get rid of the logs with every system reboot. In this scenario, the directory `/run/log/journal` will be created and used.
- Make it persistent so that it writes logs to disk. In this case, log messages will go into the `/var/log/journal` directory.

The default behaviour is as follows: if `/var/log/journal/` does not exist, logs will be saved in a volatile way to a directory in `/run/log/journal/` and — therefore — lost at reboot. The name of the directory—the `/etc/machine-id`—is a newline-terminated, hexadecimal, 32-character, lowercase string:

```
carol@debian:~$ ls /run/log/journal/8821e1fdf176445697223244d1dfbd73/
system.journal
```

If you try to read it with `less` you will get a warning, so instead use the command `journalctl`:

```
root@debian:~# less /run/log/journal/9a32ba45ce44423a97d6397918de1fa5/system.journal
"/run/log/journal/9a32ba45ce44423a97d6397918de1fa5/system.journal" may be a binary file.
See it anyway?
root@debian:~# journalctl
-- Logs begin at Sat 2019-10-05 21:26:38 CEST, end at Sat 2019-10-05 21:31:27 CEST. --
(...)
Oct 05 21:26:44 debian systemd-journald[1712]: Runtime journal
(/run/log/journal/9a32ba45ce44423a97d6397918de1fa5) is 4.9M, max 39.5M, 34.6M free.
Oct 05 21:26:44 debian systemd[1]: Started Journal Service.
(...)
```

If `/var/log/journal/` exists, logs will be stored persistently there. Should this directory be deleted, `systemd-journald` would not recreate it but write to `/run/log/journal` instead. As soon as we create `/var/log/journal/` again and restart the daemon, persistent logging will be reestablished:

```
root@debian:~# mkdir /var/log/journal/
root@debian:~# systemctl restart systemd-journald
root@debian:~# journalctl
(...)
Oct 05 21:33:49 debian systemd-journald[1712]: Received SIGTERM from PID 1 (systemd).
Oct 05 21:33:49 debian systemd[1]: Stopped Journal Service.
Oct 05 21:33:49 debian systemd[1]: Starting Journal Service...
Oct 05 21:33:49 debian systemd-journald[1768]: Journal started
Oct 05 21:33:49 debian systemd-journald[1768]: System journal
(/var/log/journal/9a32ba45ce44423a97d6397918de1fa5) is 8.0M, max 1.1G, 1.1G free.
Oct 05 21:33:49 debian systemd[1]: Started Journal Service.
Oct 05 21:33:49 debian systemd[1]: Starting Flush Journal to Persistent Storage...
(...)
```

**NOTE**

By default, there will be specific journal files for every logged in user, located in `/var/log/journal/`, so—`together with system.journal` files—you will also find files of the type `user-1000.journal`.

On top of what we have just mentioned, the way the journal daemon deals with log storage can be changed after installation by tweaking its configuration file: `/etc/systemd/journald.conf`. The key option is `Storage=` and can have the following values:

**Storage=volatile**

Log data will be stored exclusively in memory—under `/run/log/journal/`. If not present, the directory will be created.

**Storage=persistent**

By default log data will be stored on disk —under `/var/log/journal/`— with a fallback to memory (`/run/log/journal/`) during early boot stages and if the disk is not writable. Both directories will be created if needed.

**Storage=auto**

`auto` is similar to `persistent`, but the directory `/var/log/journal` is not created if needed. This is the default.

**Storage=none**

All log data will be discarded. Forwarding to other targets such as the console, the kernel log buffer, or a syslog socket are still possible, though.

For instance, to have `systemd-journald` create `/var/log/journal/` and switch to persistent storage, you would edit `/etc/systemd/journald.conf` and set `Storage=persistent`, save the file and restart the daemon with `sudo systemctl restart systemd-journald`. To ensure the restart went flawlessly, you can always check the daemon's status:

```
root@debian:~# systemctl status systemd-journald
systemd-journald.service - Journal Service
  Loaded: loaded (/lib/systemd/system/systemd-journald.service; static; vendor preset:
enabled)
  Active: active (running) since Wed 2019-10-09 10:03:40 CEST; 2s ago
    Docs: man:systemd-journald.service(8)
          man:journald.conf(5)
 Main PID: 1872 (systemd-journal)
  Status: "Processing requests..."
    Tasks: 1 (limit: 3558)
  Memory: 1.1M
  CGroup: /system.slice/systemd-journald.service
```

```
└─1872 /lib/systemd/systemd-journald
```

```
Oct 09 10:03:40 debian10 systemd-journald[1872]: Journal started
Oct 09 10:03:40 debian10 systemd-journald[1872]: System journal
(/var/log/journal/9a32ba45ce44423a97d6397918de1fa5) is 8.0M, max 1.2G, 1.2G free.
```

**NOTE**

The journal files in `/var/log/journal/<machine-id>/` or `/run/log/journal/<machine-id>/` have the `.journal` suffix (e.g. `system.journal`). However, if they are found to be corrupted or the daemon is stopped in an unclean fashion, they will be renamed appending `~` (e.g. `system.journal~`) and the daemon will start writing to a new, clean file.

## Deleting Old Journal Data: Journal Size

Logs are saved in *journal files* whose file names end with `.journal` or `.journal~` and are located in the appropriate directory (`/run/log/journal` or `/var/log/journal` as configured). To check how much disk space is currently being occupied by journal files (both archived and active), use the `--disk-usage` switch:

```
root@debian:~# journalctl --disk-usage
Archived and active journals take up 24.0M in the filesystem.
```

`systemd` logs default to a maximum of 10% of the size of the filesystem where they are stored. For instance on a 1GB filesystem they will not take up more than 100MB. Once this limit is reached, old logs will start to disappear to stay near this value.

However, size limit enforcement on stored journal files can be managed by tweaking a series of configuration options in `/etc/systemd/journald.conf`. These options fall into two categories depending on the filesystem type used: `persistent` (`/var/log/journal`) or `in-memory` (`/run/log/journal`). The former uses options that are prefixed with the word `System` and will only apply if persistent logging is properly enabled and once the system is fully booted up. The option names in the latter start with the word `Runtime` and will apply in the following scenarios:

### **SystemMaxUse=, RuntimeMaxUse=**

They control the amount of disk space that can be taken up by the journal. It defaults to 10% of the filesystem size but can be modified (e.g. `SystemMaxUse=500M`) as long as it does not surpass a maximum of 4GiB.

### **SystemKeepFree=, RuntimeKeepFree=**

They control the amount of disk space that should be left free for other users. It defaults to 15%



of the filesystem size but can be modified (e.g. `SystemKeepFree=500M`) as long as it does not surpass a maximum of 4GiB.

Regarding precedence of `*MaxUse` and `*KeepFree`, `systemd-journald` will satisfy both by using the smaller of the two values. Likewise, bear in mind that only archived (as opposed to active) journal files are deleted.

### `SystemMaxFileSize=`, `RuntimeMaxFileSize=`

They control the maximum size to which individual journal files can grow. The default is 1/8 of `*MaxUse`. Size reduction is carried out in a synchronous way and values can be specified in bytes or using K, M, G, T, P, E for Kibibytes, Mebibytes, Gibibyte, Tebibytes, Pebibytes and Exbibytes, respectively.

### `SystemMaxFiles=`, `RuntimeMaxFiles=`

They establish the maximum number of individual and archived journal files to store (active journal files are not affected). It defaults to 100.

Apart from size-based deletion and rotation of log messages, `systemd-journald` also allows for time-based criteria by using the following two options: `MaxRetentionSec=` and `MaxFileSec=`. Refer to the manual page of `journald.conf` for more information about these and other options.

#### NOTE

Whenever you modify the default behaviour of `systemd-journald` by uncommenting and editing options in `/etc/systemd/journald.conf`, you must restart the daemon for the changes to take effect.

## Vacuuming the Journal

You can manually clean archived journal files at any time with any of the following three options:

### `--vacuum-time=`

This time-based option will eliminate all messages in journal files with a timestamp older than the specified timeframe. Values must be written with any of the following suffixes: s, m, h, days (or d), months, weeks (or w) and years (or y). For instance, to get rid of all messages in archived journal files that are older than 1 month:

```
root@debian:~# journalctl --vacuum-time=1months
Deleted archived journal
/var/log/journal/7203088f20394d9c8b252b64a0171e08/system@27dd08376f71405a91794e632ede97ed
-0000000000000001-00059475764d46d6.journal (16.0M).
Deleted archived journal /var/log/journal/7203088f20394d9c8b252b64a0171e08/user-
1000@e7020d80d3af42f0bc31592b39647e9c-000000000000008e-00059479df9677c8.journal (8.0M).
```

**--vacuum-size=**

This size-based option will delete archived journal files until they occupy a value below the specified size. Values must be written with any of the following suffixes: K, M, G or T. For instance, to eliminate archived journal files until they are below 100 Mebibytes:

```
root@debian:~# journalctl --vacuum-size=100M
Vacuuming done, freed 0B of archived journals from
/run/log/journal/9a32ba45ce44423a97d6397918de1fa5.
```

**--vacuum-files=**

This option will take care that no more archived journal files than the specified number remain. The value is an integer. For instance, to limit the number of archived journal files to 10:

```
root@debian:~# journalctl --vacuum-files=10
Vacuuming done, freed 0B of archived journals from
/run/log/journal/9a32ba45ce44423a97d6397918de1fa5.
```

Vacuuming only removes archived journal files. If you want to get rid of everything (including active journal files), you need to use a signal (SIGUSR2) that requests immediate rotation of the journal files with the `--rotate` option. Other important signals can be invoked with the following options:

**--flush (SIGUSR1)**

It requests flushing of journal files from `/run/` to `/var/` to make the journal persistent. It requires that persistent logging is enabled and `/var/` is mounted.

**--sync (SIGRTMIN+1)**

It is used to request that all unwritten log data be written to disk.

**NOTE**

To check the internal consistency of the journal file, use `journalctl` with the `--verify` option. You will see a progress bar as the check is done and any possible issues will be shown.

## Retrieving Journal Data from a Rescue System

As a system administrator, you may find yourself in a situation where you need to access journal files on the hard drive of a faulty machine through a rescue system (a bootable CD or USB key containing a live Linux distribution).

`journalctl` looks for the journal files in `/var/log/journal/<machine-id>/`. Because the machine IDs on the rescue and faulty systems will be different, you must use the following option:

**-D </path/to/dir>, --directory=</path/to/dir>**

With this option, we specify a directory path where `journalctl` will search for journal files instead of the default runtime and system locations.

Thus, it is necessary that you mount the faulty system's `rootfs` (`/dev/sda1`) on the rescue system's filesystem and proceed to read the journal files like so:

```
root@debian:~# journalctl -D /media/carol/faulty.system/var/log/journal/
-- Logs begin at Sun 2019-10-20 12:30:45 CEST, end at Sun 2019-10-20 12:32:57 CEST. --
oct 20 12:30:45 suse-server kernel: Linux version 4.12.14-lp151.28.16-default
(geeko@buildhost) (...)
oct 20 12:30:45 suse-server kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.12.14-
lp151.28.16-default root=UUID=7570f67f-4a08-448e-aa09-168769cb9289 splash=>
oct 20 12:30:45 suse-server kernel: x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating
point registers'
oct 20 12:30:45 suse-server kernel: x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
(...)
```

Other options that can be useful in this scenario are:

**-m, --merge**

It merges entries from all available journals under `/var/log/journal`, including remote ones.

**--file**

It will show the entries in a specific file, for example: `journalctl --file /var/log/journal/64319965bda04dfa81d3bc4e7919814a/user-1000.journal`.

**--root**

A directory path meaning the root directory is passed as an argument. `journalctl` will search there for journal files (e.g. `journalctl --root /faulty.system/`).

See `journalctl` man page for more information.

## Forwarding Log Data to a Traditional `syslog` Daemon

Log data from the journal can be made available to a traditional `syslog` daemon by:

- Forwarding messages to the socket file `/run/systemd/journal/syslog` for `syslogd` to read.

This facility is enabled with the `ForwardToSyslog=yes` option.

- Having a `syslog` daemon behaving like `journalctl`, therefore reading log messages directly from the journal files. In this case, the relevant option is that of `Storage`; it must have a value other than `none`.

**NOTE**

Likewise, you can forward log messages to other destinations with the following options: `ForwardToKMsg` (kernel log buffer—`kmsg`), `ForwardToConsole` (the system console) or `ForwardToWall` (all logged-in users via `wall`). For more information, consult the man page for `journald.conf`.

## Guided Exercises

1. Assuming you are `root`, complete the table with the appropriate `journalctl` command:

Purpose	Command
Print kernel entries	
Print messages from the second boot starting at beginning of journal	
Print messages from the second boot starting at end of journal	
Print most recent log messages and keep watching for new ones	
Print only new messages since now, and update the output continuously	
Print messages from the previous boot with a priority of warning and in reverse order	

2. The behaviour of the journal daemon concerning storage is mostly controlled by the value of the `Storage` option in `/etc/systemd/journald.conf`. Indicate what behaviour is related to what value in the following table:

Behaviour	Storage=auto	Storage=none	Storage=persistent	Storage=volatile
Log data is thrown away but forwarding is possible.				
Once the system has booted up, log data will be stored under <code>/var/log/journal</code> . If not already present, the directory will be created.				

Behaviour	Storage=auto	Storage=none	Storage=persistent	Storage=volatile
Once the system has booted up, log data will be stored under <code>/var/log/journal</code> . If not already present, the directory will not be created.				
Log data will be stored under <code>/var/run/journal</code> but will not survive reboots.				

3. As you learned, the journal can be manually vacuumed based on time, size and number of files. Complete the following tasks using `journalctl` and the appropriate options:

- Check how much disk space is taken up by journal files:

- Cut down on the quantity of space reserved for archived journal files and set it to 200MiB:

- Check on disk space again and explain the results:

## Explorational Exercises

1. What options should you modify in `/etc/systemd/journald.conf` so that messages are forwarded to `/dev/tty5`? What values should the options have?

2. Provide the correct `journalctl` filter to print the following:

Purpose	Filter + Value
Print messages belonging to a specific user	
Print messages from a host named <code>debian</code>	
Print messages belonging to a specific group	
Print messages belongin to <code>root</code>	
Based on the executable path, print <code>sudo</code> messages	
Based on the command name, print <code>sudo</code> messages	

3. When filtering by priority, logs with a higher priority than indicated will also be included in the listing; for instance `journalctl -p err` will print *error*, *critical*, *alert* and *emergency* messages. However, you can have `journalctl` show only a specific range. What command would you use to have `journalctl` print only messages in the *warning*, *error* and *critical* priority levels?

4. Priority levels can also be specified numerically. Rewrite the command in the previous exercise using the numeric representation of priority levels:

# Summary

In this lesson you learned:

- The advantages of using `systemd` as a system and service manager.
- The basics of `systemd` units and targets.
- Where `systemd-journald` gets logging data from.
- The options you can pass `systemctl` to control `systemd-journald`: `start`, `status`, `restart` and `stop`.
- Where the journal's configuration file is located — `/etc/systemd/journald.conf` — and its main options.
- How to query the journal in a general way and for specific data by using filters.
- How to navigate and search through the journal.
- How to deal with storage of journal files: in memory vs. on disk.
- How to disable journaling altogether.
- How to check the disk space taken up by the journal, enforce size limits on stored journal files and clean archived journal files manually (*vacuuming*).
- How to retrieve journal data from a rescue system.
- How to forward log data to a traditional `syslog` daemon.

Commands used in this lesson:

## **systemctl**

Control the `systemd` system and service manager.

## **journalctl**

Query the `systemd` journal.

## **ls**

List directory contents.

## **less**

View file contents.

## **mkdir**

Make directories.



## Answers to Guided Exercises

1. Assuming you are `root`, complete the table with the appropriate `journalctl` command:

Purpose	Command
Print kernel entries	<code>journalctl -k</code> or <code>journalctl --dmesg</code>
Print messages from the second boot starting at beginning of journal	<code>journalctl -b 2</code>
Print messages from the second boot starting at end of journal	<code>journalctl -b -2 -r</code> or <code>journalctl -r -b -2</code>
Print most recent log messages and keep watching for new ones	<code>journalctl -f</code>
Print only new messages since now, and update the output continuously	<code>journalctl --since "now" -f</code>
Print messages from the previous boot with a priority of warning and in reverse order	<code>journalctl -b -1 -p warning -r</code>

2. The behaviour of the journal daemon concerning storage is mostly controlled by the value of the `Storage` option in `/etc/systemd/journald.conf`. Indicate what behaviour is related to what value in the following table:

Behaviour	Storage=auto	Storage=none	Storage=persistent	Storage=volatile
Log data is thrown away but forwarding is possible		x		
Once the system has booted up, log data will be stored under <code>/var/log/journal</code> . If not already present, the directory will be created			x	

Behaviour	Storage=auto	Storage=none	Storage=persistent	Storage=volatile
Once the system has booted up, log data will be stored under <code>/var/log/journal</code> . If not already present, the directory will not be created	x			
Log data will be stored under <code>/var/run/journal</code> but will not survive reboots				x

3. As you learned, the journal can be manually vacuumed based on time, size and number of files. Complete the following tasks using `journalctl` and the appropriate options:

- Check how much disk space is taken up by journal files:

```
journalctl --disk-usage
```

- Cut down on the quantity of space reserved for archived journal files and set it to 200MiB:

```
journalctl --vacuum-size=200M
```

- Check on disk space again and explain the results:

```
journalctl --disk-usage
```

There is no correlation because `--disk-usage` shows space occupied by both active and archived journal files whereas `--vacuum-size` only applies to archived files.

## Answers to Explorational Exercises

1. What options should you modify in `/etc/systemd/journald.conf` so that messages are forwarded to `/dev/tty5`? What values should the options have?

```
ForwardToConsole=yes
TTYPath=/dev/tty5
```

2. Provide the correct `journalctl` filter to print the following:

Purpose	Filter + Value
Print messages belonging to a specific user	<code>_ID=&lt;user-id&gt;</code>
Print messages from a host named <code>debian</code>	<code>_HOSTNAME=debian</code>
Print messages belonging to a specific group	<code>_GID=&lt;group-id&gt;</code>
Print messages belonging to <code>root</code>	<code>_UID=0</code>
Based on the executable path, print <code>sudo</code> messages	<code>_EXE=/usr/bin/sudo</code>
Based on the command name, print <code>sudo</code> messages	<code>_COMM=sudo</code>

3. When filtering by priority, logs with a higher priority than indicated will also be included in the listing; for instance `journalctl -p err` will print *error*, *critical*, *alert* and *emergency* messages. However, you can have `journalctl` show only a specific range. What command would you use to have `journalctl` print only messages in the *warning*, *error* and *critical* priority levels?

```
journalctl -p warning..crit
```

4. Priority levels can also be specified numerically. Rewrite the command in the previous exercise using the numeric representation of priority levels:

```
journalctl -p 4..2
```



## 108.3 Mail Transfer Agent (MTA) basics

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 108.3](#)

### Weight

3

### Key knowledge areas

- Create e-mail aliases.
- Configure e-mail forwarding.
- Knowledge of commonly available MTA programs (postfix, sendmail, exim) (no configuration)

### Partial list of the used files, terms and utilities

- `~/ .forward`
- sendmail emulation layer commands
- `newaliases`
- `mail`
- `mailq`
- `postfix`
- `sendmail`
- `exim`



# 108.3 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	108 Essential System Services
<b>Objective:</b>	108.3 Mail Transfer Agent (MTA) basics
<b>Lesson:</b>	1 of 1

## Introduction

In Unix-like operating systems, such as Linux, every user has their own *inbox*: a special location on the filesystem that is inaccessible by other non-root users and stores the user's personal email messages. New incoming messages are added to the user's inbox by the *Mail Transfer Agent* (MTA). The MTA is a program running as a system service which collects messages sent by other local accounts as well as messages received from the network, sent from remote user accounts.

The same MTA is also responsible for sending messages to the network, if the destination address refers to a remote account. It does so by using a filesystem location as an email *outbox* for all system users: as soon as a user places a new message in the outbox, the MTA will identify the target network node from the domain name given by the destination email address — the portion after the @ sign — and then it will try to transfer the message to the remote MTA using the *Simple Mail Transfer Protocol* (SMTP). SMTP was designed with unreliable networks in mind, so it will try to establish alternative delivery routes if the primary mail destination node is unreachable.

## Local and Remote MTA

Traditional user accounts in network connected machines make up the simplest email exchange

scenario, where every network node runs its own MTA daemon. No software other than the MTA is required to send and to receive email messages. In practice, however, it is more common to use a remote email account and not have an active local MTA service (i.e. instead using an email client application to access the remote account).

Unlike local accounts, a remote email account—also called *remote mailbox*—requires user authentication to grant access to the user’s mailbox and to the remote MTA (in this case, simply called the *SMTP server*). While the user interacting with a local inbox and MTA is already identified by the system, a remote system must verify the user’s identity before handling its messages through IMAP or POP3.

**NOTE**

Nowadays the most common method to send and receive email is through a hosted account on a remote server, e.g. a company’s centralized email server hosting all employee accounts or a personal email service, such as Google’s *Gmail*. Instead of collecting locally-delivered messages, the email client application will connect to the remote mailbox and retrieve the messages from there. The POP3 and IMAP protocols are commonly used to retrieve the messages from the remote server, but other non-standard proprietary protocols may be used as well.

When an MTA daemon is running on the local system, local users can send an email to other local users or to users on a remote machine, provided their system also has an MTA service that is accepting network connections. TCP port 25 is the standard port for SMTP communication, but other ports may be used as well, depending on the authentication and/or encryption schema being used (if any).

Leaving aside topologies involving the access to remote mailboxes, an email exchange network between ordinary Linux user accounts can be implemented as long as all network nodes have an active MTA that is able to perform the following tasks:

- Maintain the outbox queue of messages to be sent. For each queued message, the local MTA will assess the destination MTA from the recipient’s address.
- Communicate with remote MTA daemons using SMTP. The local MTA should be able to use the Simple Mail Transfer Protocol (SMTP) over the TCP/IP stack to receive, send and redirect messages from/to other remote MTA daemons.
- Maintain an individual inbox for every local account. The MTA will usually store the messages in the *mbox* format: a single text file containing all email messages in sequence.

Normally, email addresses specify a domain name as the location, e.g. `lpi.org` in `info@lpi.org`. When this is the case the sender’s MTA will query the DNS service for the corresponding MX record. The DNS MX record contains the IP address of the MTA handling the email for that domain. If the same domain has more than one MX record specified in the DNS, the MTA should

try to contact them according to their priority values. If the recipient's address does not specify a domain name or the domain does not have an MX record, then the part after the @ symbol will be treated as the host of the destination MTA.

Security aspects must be considered if the MTA hosts will be visible to hosts on the Internet. For instance, it is possible for an unknown user to use the local MTA to impersonate another user and send potentially harmful emails. An MTA that blindly relays an email is known as an *open relay*, when it can be used as an intermediary to potentially disguise the true sender of the message. To prevent these misuses, the recommendation is to accept connections from authorized domains only and to implement a secure authentication schema.

In addition to that, there are many different MTA implementations for Linux, each one focusing on specific aspects like compatibility, performance, security, etc. Nevertheless, all MTAs will follow the same basic principles and provide similar features.

## Linux MTAs

The traditional MTA available for Linux systems is *Sendmail*, a very flexible general purpose MTA used by many Unix-like operating systems. Some of the other common MTAs are *Postfix*, *qmail* and *Exim*. The main reason to choose an alternative MTA is to implement advanced features more easily, as configuring custom email servers in Sendmail can be a complicated task. Also, each distribution may have its preferred MTA, with predefined settings appropriate for most common setups. All MTAs intend to be drop-in Sendmail replacements, so all Sendmail-compatible applications should work regardless of what MTA is being used.

If the MTA is running but not accepting network connections, it will only be able to deliver email messages on the local machine. For the `sendmail` MTA, the `/etc/mail/sendmail.mc` file should be modified to accept non-local connections. To do so, the entry

```
DAEMON_OPTIONS(`Port=smtp,Addr=127.0.0.1, Name=MTA')dnl
```

should be modified to the correct network address and the service should be restarted. Some Linux distributions, such as Debian, may offer configuration tools to help bring up the email server with a predefined set of commonly used features.

### TIP

Due to security issues most Linux distributions do not install an MTA by default. To test the examples given in this lesson, make sure there's a running MTA on every machine and that they are accepting connections on TCP port 25. For security purposes, these systems should not be exposed to incoming connections from the public internet during testing.

Once the MTA is running and accepting connections from the network, new email messages are passed to it with SMTP commands that are sent through a TCP connection. The command `nc` — a networking utility which reads and writes generic data across the network — can be used to send SMTP commands directly to the MTA. If the command `nc` is not available, it will be installed with the `ncat` or `nmap-ncat` package, depending on the package management system in use. Writing SMTP commands directly to the MTA will help you understand the protocol and other general email concepts better, but it can also help to diagnose problems in the mail delivery process.

If, for example, user `emma` at the `lab1.campus` host wants to send a message to user `dave` at the `lab2.campus` host, then she can use the `nc` command to directly connect to the `lab2.campus` MTA, assuming it is listening on the TCP port 25:

```
$ nc lab2.campus 25
220 lab2.campus ESMTP Sendmail 8.15.2/8.15.2; Sat, 16 Nov 2019 00:16:07 GMT
HELO lab1.campus
250 lab2.campus Hello lab1.campus [10.0.3.134], pleased to meet you
MAIL FROM: emma@lab1.campus
250 2.1.0 emma@lab1.campus... Sender ok
RCPT TO: dave@lab2.campus
250 2.1.5 dave@lab2.campus... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Subject: Recipient MTA Test

Hi Dave, this is a test for your MTA.
.
250 2.0.0 xAG0G7Y0000595 Message accepted for delivery
QUIT
221 2.0.0 lab2.campus closing connection
```

Once the connection is established, the remote MTA identifies itself and then it's ready to receive SMTP commands. The first SMTP command in the example, `HELO lab1.campus`, indicates `lab1.campus` as the exchange initiator. The next two commands, `MAIL FROM: emma@lab1.campus` and `RCPT TO: dave@lab2.campus`, indicate the sender and the recipient. The proper email message starts after the `DATA` command and ends with a period on a line by itself. To add a subject field to the email, it should be in the first line after the `DATA` command, as shown in the example. When the subject field is used, there must be an empty line separating it from the email content. The `QUIT` command finishes the connection with the MTA at the `lab2.campus` host.

On the `lab2.campus` host, user `dave` will receive a message similar to `You have new mail in`



`/var/spool/mail/dave` as soon as he enters a shell session. This file will contain the raw email message sent by `emma` as well as the headers added by the MTA:

```
$ cat /var/spool/mail/dave
From emma@lab1.campus Sat Nov 16 00:19:13 2019
Return-Path: <emma@lab1.campus>
Received: from lab1.campus (lab1.campus [10.0.3.134])
    by lab2.campus (8.15.2/8.15.2) with SMTP id xAG0G7Y0000595
    for dave@lab2.campus; Sat, 16 Nov 2019 00:17:06 GMT
Date: Sat, 16 Nov 2019 00:16:07 GMT
From: emma@lab1.campus
Message-Id: <201911160017.xAG0G7Y0000595@lab2.campus>
Subject: Recipient MTA Test

Hi Dave, this is a test for your MTA.
```

The header `Received:` shows that the message from `lab1.campus` was received directly by `lab2.campus`. By default, MTAs will only accept messages to local recipients. The following error will probably occur if user `emma` tries to send an email to user `henry` at the `lab3.campus` host, but using the `lab2.campus` MTA instead of the proper `lab3.campus` MTA:

```
$ nc lab2.campus 25
220 lab2.campus ESMTP Sendmail 8.15.2/8.15.2; Sat, 16 Nov 2019 00:31:44 GMT
HELO lab1.campus
250 lab2.campus Hello lab1.campus [10.0.3.134], pleased to meet you
MAIL FROM: emma@lab1.campus
250 2.1.0 emma@lab1.campus... Sender ok
RCPT TO: henry@lab3.campus
550 5.7.1 henry@lab3.campus... Relaying denied
```

SMTP response numbers beginning with 5, such as the `Relaying denied` message, indicate an error. There are legitimate situations when relaying is desirable, like when the hosts sending and receiving emails are not connected all the time: an intermediate MTA can be configured to accept emails intended to other hosts, acting as a *relay* SMTP server that can forward messages between MTAs.

The ability to route email traffic through intermediate SMTP servers discourages the attempt to connect directly to the host indicated by the recipient's email address, as shown in the previous examples. Moreover, email addresses often have a domain name as the location (after the @), so the actual name of the corresponding MTA host must be retrieved through DNS. Therefore, it is recommended to delegate the task of identifying the appropriate destination host to the local MTA

or to the remote SMTP server, when remote mailboxes are used.

Sendmail provides `sendmail` command to perform many email-related operations, including assisting in composing new messages. It also requires the user to type the email headers by hand, but in a friendlier way than using SMTP commands directly. So a more adequate method for user `emma@lab1.campus` to send an email message to `dave@lab2.campus` would be:

```
$ sendmail dave@lab2.campus
From: emma@lab1.campus
To: dave@lab2.campus
Subject: Sender MTA Test

Hi Dave, this is a test for my MTA.
.
```

Here again, the dot in a line by itself finishes the message. The message should be immediately sent to the recipient, unless the local MTA was unable to contact the remote MTA. The command `mailq`, if executed by root, will show all non-delivered messages. If, for example, the MTA at `lab2.campus` did not respond, then command `mailq` will list the undelivered message and the cause of the failure:

```
# mailq
/var/spool/mqueue (1 request)
-----Q-ID----- --Size-- -----Q-Time----- -----Sender/Recipient-----
xAIK3D9S000453      36 Mon Nov 18 20:03 <emma@lab1.campus>
                    (Deferred: Connection refused by lab2.campus.)
                    <dave@lab2.campus>
Total requests: 1
```

The default location for the outbox queue is `/var/spool/mqueue/`, but different MTAs may use different locations in the `/var/spool/` directory. Postfix, for instance, will create a directory tree under `/var/spool/postfix/` to manage the queue. The command `mailq` is equivalent to `sendmail -bp`, and they should be present regardless of the MTA installed in the system. To ensure backward compatibility, most MTAs provide these traditional mail administration commands.

If the primary email destination host—when it is provided from an MX DNS record for the domain—is unreachable, the MTA will try to contact the entries with lower priority (if there are any specified). If none of them are reachable, the message will stay in the local outbox queue to be sent later. If configured to do so, the MTA can periodically check the availability of the remote hosts and perform a new delivery attempt. If using a Sendmail-compatible MTA, a new attempt

will immediately take place with the command `sendmail -q`.

Sendmail will store incoming messages in a file named after the corresponding inbox owner, for example `/var/spool/mail/dave`. Other MTAs, like Postfix, may store the incoming email messages in locations like `/var/mail/dave`, but the file content is the same. In the example, the command `sendmail` was used in the sender's host to compose the message, so the raw message headers show that the email took extra steps before reaching the final destination:

```
$ cat /var/spool/mail/dave
From emma@lab1.campus Mon Nov 18 20:07:39 2019
Return-Path: <emma@lab1.campus>
Received: from lab1.campus (lab1.campus [10.0.3.134])
    by lab2.campus (8.15.2/8.15.2) with ESMTPS id xAIK7c1C000432
    (version=TLSv1.3 cipher=TLS_AES_256_GCM_SHA384 bits=256 verify=NOT)
    for <dave@lab2.campus>; Mon, 18 Nov 2019 20:07:38 GMT
Received: from lab1.campus (localhost [127.0.0.1])
    by lab1.campus (8.15.2/8.15.2) with ESMTPS id xAIK3D9S000453
    (version=TLSv1.3 cipher=TLS_AES_256_GCM_SHA384 bits=256 verify=NOT)
    for <dave@lab2.campus>; Mon, 18 Nov 2019 20:03:13 GMT
Received: (from emma@localhost)
    by lab1.campus (8.15.2/8.15.2/Submit) id xAIK0doL000449
    for dave@lab2.campus; Mon, 18 Nov 2019 20:00:39 GMT
Date: Mon, 18 Nov 2019 20:00:39 GMT
Message-Id: <201911182000.xAIK0doL000449@lab1.campus>
From: emma@lab1.campus
To: dave@lab2.campus
Subject: Sender MTA Test

Hi Dave, this is a test for my MTA.
```

From bottom to top, the lines starting with `Received:` show the route taken by the message. The message was submitted by user `emma` with the command `sendmail dave@lab2.campus` issued on `lab1.campus`, as stated by the first `Received:` header. Then, still on `lab1.campus`, the MTA uses ESMTPS — a superset of the SMTP, which adds encryption extensions — to send the message to the MTA at `lab2.campus`, as stated by the last (top) `Received:` header.

The MTA finishes its job after the message is saved in the user's inbox. It is common to perform some kind of email filtering, such as spam blockers or the enforcement of user-defined filtering rules. These tasks are executed by third-party applications, working together with the MTA. The MTA could, for example, call the *SpamAssassin* utility to mark suspicious messages using its text analysis features.

Although possible, it is not convenient to read the mailbox file directly. It is recommended to use an email client program instead (e.g. Thunderbird, Evolution, or KMail), which will parse the file and appropriately manage the messages. Such programs also offer extra features, like shortcuts to common actions, inbox sub-directories, etc.

## The `mail` Command and Mail User Agents (MUA)

It is possible to write an email message directly in its raw format, but it is much more practical to use a client application — also known as an MUA (*Mail User Agent*) — to speed up the process and to avoid mistakes. The MUA takes care of the work under the hood, that is, the email client presents and organizes the received messages and handles the proper communication with the MTA after the user composes an email.

There are many distinct types of Mail User Agents. Desktop applications like *Mozilla Thunderbird* and *Gnome's Evolution* support both local and remote email accounts. Even *Webmail* interfaces can be seen as a type of MUA, as they intermediate the interaction between the user and the underlying MTA. Email clients are not restricted to graphical interfaces though: console email clients are widely used to access mailboxes not integrated with a graphical interface and to automate email related tasks within shell scripts.

Originally, the Unix `mail` command was only intended to share messages between local system users (the first `mail` command dates back to the first Unix edition, released in 1971). When network email exchanges became more prominent, other programs were created to deal with the new delivery system and gradually replaced the old `mail` program.

Nowadays, the most commonly used `mail` command is provided by the *mailx* package, which is compatible with all modern email features. In most Linux distributions, the command `mail` is just a symbolic link to the `mailx` command. Other implementations, like the *GNU Mailutils* package, basically provide the same features as `mailx`. There are, however, slight differences between them, especially with respect to command-line options.

Regardless of their implementation, all modern variations of the `mail` command operate in two modes: *normal mode* and *send mode*. If an email address is provided as an argument to the command `mail`, it will enter the send mode, otherwise it will enter the normal (read) mode. In normal mode, the received messages are listed with a numerical index for each one, so the user can refer to them individually when typing commands in the interactive prompt. The command `print 1` can be used to display the contents of message number 1, for example. Interactive commands can be abbreviated, so commands like `print`, `delete` or `reply` can be replaced by `p`, `d` or `r`, respectively. The `mail` command will always consider the last received or the last viewed message when the message index number is omitted. The command `quit` or `q` will exit the program.

The *send mode* is especially useful for sending automated email messages. It can be used, for example, to send an email to the system administrator if a scheduled maintenance script fails to perform its task. In send mode, `mail` will use the contents from the *standard input* as the message body:

```
$ mail -s "Maintenance fail" henry@lab3.campus <<<"The maintenance script failed at `date`"
```

In this example, the option `-s` was added to include a subject field to the message. The message body was provided by the *Hereline* redirection to the standard input, but the contents of a file or the output of a command could also be piped to the program's *stdin*. If no content is provided by a redirection to the standard input, then the program will wait for the user to enter the message body. In this case, keystroke `Ctrl + D` will end the message. The `mail` command will immediately exit after the message is added to the outbox queue.

## Delivery Customization

By default, the email accounts on a Linux system are associated with the standard system accounts. For example, If user Carol has the login name `carol` on the host `lab2.campus` then her email address will be `carol@lab2.campus`. This one-to-one association between system accounts and mailboxes can be extended by standard methods provided by most Linux distributions, in particular the email routing mechanism provided by the `/etc/aliases` file.

An email alias is a “virtual” email recipient whose receiving messages are redirected to existing local mailboxes or to other types of message storage or processing destinations. Aliases are useful, for example, to place messages sent to `postmaster@lab2.campus` in Carol's mailbox, which is an ordinary local mailbox in the `lab2.campus` system. To do so, the line `postmaster: carol` should be added to the `/etc/aliases` file in `lab2.campus`. After modifying the `/etc/aliases` file, the command `newaliases` should be executed to update the MTA's aliases database and make the changes effective. Commands `sendmail -bi` or `sendmail -I` can also be used to update the aliases database.

Aliases are defined one per line, in the format `<alias>: <destination>`. In addition to the ordinary local mailboxes, indicated by the corresponding username, other destination types are available:

- A full path (starting with `/`) to a file. Messages sent to the corresponding alias will be appended to the file.
- A command to process the message. The `<destination>` must start with a pipe character and, if the command contains special characters (like blank spaces), it must be enclosed in double quotes. For example, the alias `subscribe: |subscribe.sh in lab2.campus` will forward all

messages sent to `subscribe@lab2.campus` to the standard input of the command `subscribe.sh`. If `sendmail` is running in *restricted shell mode*, the allowed commands — or the links to them — should be in `/etc/smrsh/`.

- An include file. A single alias can have multiple destinations (separated by commas), so it may be more practical to keep them in an external file. The `:include:` keyword must indicate the file path, as in `:include:/var/local/destinations`
- An external address. Aliases can also forward messages to external email addresses.
- Another alias.

An unprivileged local user can define aliases for their own email by editing the file `.forward` in their home directory. As the aliases can only affect their own mailbox, only the `<destination>` part is necessary. To forward all incoming emails to an external address, for example, user `dave` in `lab2.campus` could create the following `~/ .forward` file:

```
$ cat ~/.forward
emma@lab1.campus
```

It will forward all email messages sent to `dave@lab2.campus` to `emma@lab1.campus`. As with the `/etc/aliases` file, other redirection rules can be added to `.forward`, one per line. Nevertheless, the `.forward` file must be writable by its owner only and it is not necessary to execute the `newaliases` command after modifying it. Files starting with a dot do not appear in regular file listings, which could make the user unaware of active aliases. Therefore, it is important to verify if the file exists when diagnosing email delivery issues.

## Guided Exercises

1. Without further options or arguments, the command `mail henry@lab3.campus` enters the input mode so the user can type the message to `henry@lab3.campus`. After finishing the message, which keystroke will close the input mode and dispatch the email?

2. Which command can the root user execute to list the undelivered messages that originated on the local system?

3. How can an unprivileged user use the standard MTA method to automatically forward all of their incoming mail to the address `dave@lab2.campus`?

## Explorational Exercises

1. Using the `mail` command provided by `mailx`, what command will send a message to `emma@lab1.campus` with the file `logs.tar.gz` as an attachment and the output of the command `uname -a` as the email body?

2. An email service administrator wants to monitor email transfers through the network, but they don't want to clutter their mailbox with test messages. How could this administrator configure a system-wide email alias to redirect all email sent to the user `test` to the file `/dev/null`?

3. What command, besides `newaliases`, could be used to update the aliases database after adding a new alias to `/etc/aliases`?



## Summary

This lesson covered the role and usage of Mail Transfer Agents in Linux systems. The MTA provides a standard method for communication amongst user accounts and can be combined with other software to provide extra functionality. The lesson discussed the following topics:

- Concepts about email-related technologies, mailboxes and protocols.
- How Linux MTAs exchange messages over the network.
- Console email clients and MUAs (Mail User Agents).
- Local email aliasing and forwarding.

The technologies, commands and procedures addressed were:

- SMTP and related protocols.
- MTAs available for Linux: Sendmail, Postfix, qmail, Exim.
- MTA and MUA commands: `sendmail` and `mail`.
- Administrative files and commands: `mailq`, `/etc/aliases`, `newaliases`, `~/ .forward`.

## Answers to Guided Exercises

1. Without further options or arguments, the command `mail henry@lab3.campus` enters the input mode so the user can type the message to `henry@lab3.campus`. After finishing the message, which keystroke will close the input mode and dispatch the email?

Pressing `Ctrl + D` will cause the program to close and dispatch the email.

2. Which command can the root user execute to list the undelivered messages that originated on the local system?

The command `mailq` or `sendmail -bp`.

3. How can an unprivileged user use the standard MTA method to automatically forward all of their incoming mail to the address `dave@lab2.campus`?

The user should add `dave@lab2.campus` to `~/ .forward`.

## Answers to Explorational Exercises

1. Using the `mail` command provided by `mailx`, what command will send a message to `emma@lab1.campus` with the file `logs.tar.gz` as an attachment and the output of the command `uname -a` as the email body?

```
uname -a | mail -a logs.tar.gz emma@lab1.campus
```

2. An email service administrator wants to monitor email transfers through the network, but they don't want to clutter their mailbox with test messages. How could this administrator configure a system-wide email alias to redirect all email sent to the user `test` to the file `/dev/null`?

The `test: /dev/null` line in `/etc/aliases` will redirect all messages sent to the local mailbox `test` to the file `/dev/null`.

3. What command, besides `newaliases`, could be used to update the aliases database after adding a new alias to `/etc/aliases`?

Command `sendmail -bi` or `sendmail -I`.



## 108.4 Manage printers and printing

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 108.4](#)

### Weight

2

### Key knowledge areas

- Basic CUPS configuration (for local and remote printers).
- Manage user print queues.
- Troubleshoot general printing problems.
- Add and remove jobs from configured printer queues.

### Partial list of the used files, terms and utilities

- CUPS configuration files, tools and utilities
- `/etc/cups/`
- lpd legacy interface (`lpr`, `lprm`, `lpq`)



# 108.4 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	108 Essential System Services
<b>Objective:</b>	108.4 Manage printers and printing
<b>Lesson:</b>	1 of 1

## Introduction

Declarations of a “paperless society” brought on by the advent of computers have, up to current times, proven to be false. Many organizations still rely on printed, or “hard copy”, pages of information. With this in mind we can see how important it is for a computer user to know how to print from a system as well as an administrator needing to know how to maintain a computer’s ability to work with printers.

On Linux, as well as many other operating systems, the *Common Unix Printing System* (CUPS) software stack allows for printing and printer management from a computer. Here is a very simplified outline of how a file is printed in Linux using CUPS:

1. A user submits a file to be printed.
2. The CUPS daemon, `cupsd`, then *spools* the print job. This print job is given a job number by CUPS, along with information about which print queue holds the job as well as the name of the document to print.
3. CUPS utilizes *filters* that are installed on the system to generate a formatted file that the printer can use.

4. CUPS then sends the re-formatted file to the printer for printing.

We will look at these steps in more detail, as well as how to install and manage a printer in Linux.

## The CUPS Service

Most Linux desktop installations will have the CUPS packages already installed. On minimal Linux installations the CUPS packages may not be installed, depending on the distribution. A basic CUPS installation can be performed on a Debian system with the following:

```
$ sudo apt install cups
```

On Fedora systems the installation process is just as easy. You will need to start the CUPS service manually after installation on Fedora and other Red Hat-based distributions:

```
$ sudo dnf install cups
...
$ sudo systemctl start cups.service
```

After the installation has completed, you can verify that the CUPS service is running with the use of the `systemctl` command:

```
$ systemctl status cups.service
● cups.service - CUPS Scheduler
   Loaded: loaded (/lib/systemd/system/cups.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2020-06-25 14:35:47 EDT; 41min ago
     Docs: man:cupsd(8)
  Main PID: 3136 (cupsd)
    Tasks: 2 (limit: 1119)
   Memory: 3.2M
    CGroup: /system.slice/cups.service
            └─3136 /usr/sbin/cupsd -l
               └─3175 /usr/lib/cups/notifier/dbus dbus://
```

As with many other Linux daemons, CUPS relies on a set of configuration files for its operations. Listed below are the main ones that are of interest to the system administrator:

### `/etc/cups/cupsd.conf`

This file contains the configuration settings for the CUPS service itself. If you are at all familiar with the Apache web server configuration file then the CUPS configuration file will look quite

similar to you as it uses a very similar syntax. The `cupsd.conf` file contains settings for things such as controlling access to the various print queues in use on the system, if whether or not the CUPS web interface is enabled, as well as the level of logging that the daemon will use.

### `/etc/printcap`

This is the legacy file that was used by the LPD (*Line Printer Daemon*) protocol before the advent of CUPS. CUPS will still create this file on systems for backwards compatibility and it is often-times a symbolic link to `/run/cups/printcap`. Each line in this file contains a printer that the system has access to.

### `/etc/cups/printers.conf`

This file contains each printer that is configured to be used by the CUPS system. Each printer and its associated print queue in this file is enclosed within a `<Printer></Printer>` stanza. This file provides the individual printer listings found within `/etc/printcap`.

#### WARNING

No modifications to the `/etc/cups/printers.conf` file should be made at the command line while the CUPS service is running.

### `/etc/cups/ppd/`

This is not a configuration file but a directory that holds the *PostScript Printer Description* (PPD) files for the printers that use them. Each printer's operating capabilities will be stored within a PPD file (ending in the `.ppd` extension). These are plain-text files and follow a specific format.

The CUPS service also utilizes logging in much the same manner as the Apache 2 service. The logs are stored within `/var/log/cups/` and contain an `access_log`, `page_log`, and an `error_log`. The `access_log` keeps a record of access to the CUPS web interface as well as actions taken within it, such as printer management. The `page_log` keeps track of print jobs that have been submitted to the print queues managed by the CUPS installation. The `error_log` will contain messages about print jobs that have failed and other errors recorded by the web interface.

We will next look at the tools and utilities that are used to manage the CUPS service.

## Using the Web Interface

As stated earlier, the `/etc/cups/cupsd.conf` configuration file determines if the web interface for the CUPS system is enabled. The configuration option looks like this:

```
# Web interface setting...
WebInterface Yes
```

If the web interface is enabled, then CUPS can be managed from a browser at the default URL of

`http://localhost:631`. By default a user on the system can view printers and print queues but any form of configuration modification requires a user with root access to authenticate with the web service. The configuration stanza within the `/etc/cups/cupsd.conf` file for restricting access to administrative capabilities will resemble the following:

```
# All administration operations require an administrator to authenticate...
<Limit CUPS-Add-Modify-Printer CUPS-Delete-Printer CUPS-Add-Modify-Class CUPS-Delete-Class
CUPS-Set-Default>
  AuthType Default
  Require user @SYSTEM
  Order deny,allow
</Limit>
```

Here is a breakdown of these options:

### **AuthType Default**

will use a basic authentication prompt when an action requires root access.

### **Require user @SYSTEM**

indicates that a user with administrative privileges will be required for the operation. This could be changed to `@groupname` where members of `groupname` can administer the CUPS service or individual users could be provided with a list as in `Require user carol, tim`.

### **Order deny,allow**

works much like the Apache 2 configuration option where the action is denied by default unless a user (or member of a group) is authenticated.

The web interface for CUPS can be disabled by first stopping the CUPS service, changing the `WebInterface` option from `Yes` to `No`, then restarting the CUPS service.

The CUPS web interface is built like a basic web site with navigation tabs for various sections of CUPS system. The tabs in the web interface include the following:

### **Home**

The home page will list the current version of CUPS that is installed. It also breaks down CUPS into sections such as:

### **CUPS for Users**

Provides a description of CUPS, command-line options for working with printers and print queues, and a link to the CUPS user forum.



## CUPS for Administrators

Provides links in the interface to install and manage printers and links to information about working with printers on a network.

## CUPS for Developers

Provides links to developing for CUPS itself as well as creating PPD files for printers.

## Administration

The administration page is also broken down into sections:

### Printers

Here an administrator can add new printers to the system, locate printers connected to the system and manage printers that are already installed.

### Classes

Classes are a mechanism where printers can be added to groups with specific policies. For example, a class can contain a group of printers that belong to a specific floor of a building that only users within a particular department can print to. Another class can have limitations on how many pages a user can print. Classes are not created by default on a CUPS installation and have to be defined by an administrator. This is the section in the CUPS web interface where new classes can be created and managed.

### Jobs

This is where an administrator can view all print jobs that are currently in queue for all printers that this CUPS installation manages.

### Server

This is where an administrator can make changes to the `/etc/cups/cupsd.conf` file. Also, further configuration options are available via check boxes such as allowing printers connected to this CUPS installation to be shared on a network, advanced authentication, and allowing remote printer administration.

## Classes

If printer classes are configured on the system they will be listed on this page. Each printer class will have options to manage all of the printers in the class at once, as well as view all jobs that are in queue for the printers in this class.

## Help

This tab provides links for all of the available documentation for CUPS that is installed on the system.

## Jobs

The Jobs tab allows for the searching of individual print jobs as well as listing out all of the current print jobs managed by the server.

## Printers

The Printers tab lists all of the printers currently managed by the system as well as a quick overview of each printer's status. Each printer listed can be clicked on and the administrator will be taken to the page where the individual printer can be further managed. The information for the printers on this tab comes from the `/etc/cups/printers.conf` file.

# Installing a Printer

Adding a printer queue to the system is a simple process within the CUPS web interface:

1. Click on the **Administration** tab then the **Add Printer** button.
2. The next page will provide various options depending on how your printer is connected to your system. If it is a local printer select the most relevant option such as which port the printer is connected to or which third-party printer software may be installed. CUPS will also try to detect printers that are connected to the network and display those here. You can also choose a direct connection option to a network printer depending on which network printing protocols the printer supports. Select your appropriate option and click on the **Continue** button.
3. The next page will allow you to provide a name, description, and location (such as “back office” or “front desk” etc.) for the printer. If you wish to share this printer over the network you can select the check box for that option on this page as well. Once your settings have been entered, click on the **Continue** button.
4. The next page is where the printer's make and model can be selected. This allows CUPS to search its locally installed database for the most suitable drivers and PPD files to use with the printer. If you have a PPD file provided by the printer manufacturer, browse to its location and select it for use here. Once this is done, click on the **Add Printer** button.
5. The final page is where you would set your default options such as the page size that the printer will use and the resolution of the characters printed to the page. Click on the **Set Default Options** button and your printer is now installed on your system.

### NOTE

Many desktop installations of Linux will have different tools that can be used to install a printer. The GNOME and KDE desktop environments have their own applications built-in that can be used to install and manage printers. Also, some distributions provide separate printer management applications. However when dealing with a server installation that many users will print to, the CUPS web interface may provide the best tools for the task.

A printer's queue may also be installed using the legacy LPD/LPR commands. Here is an example using the `lpadmin` command:

```
$ sudo lpadmin -p ENVY-4510 -L "office" -v socket://192.168.150.25 -m everywhere
```

We will break down the command to illustrate the options used here:

- Since the addition of a printer to the system requires a user with administrative privileges, we prepend the `lpadmin` command with `sudo`.
- The `-p` option is the destination for your print jobs. It is essentially a friendly name for the user to know where the print jobs will land. Typically you can provide the name of the printer.
- The `-L` option is the location of the printer. This is optional but helpful should you need to manage a number of printers in various locations.
- The `-v` option is for the printer device's URI. The device URI is what the CUPS print queue needs in order to send rendered print jobs to a specific printer. In our example, we are using a network location using the IP address provided.
- The final option, `-m`, is set to "everywhere". This sets the model of the printer for CUPS to determine which PPD file to use. In modern versions of CUPS, it is best to use "everywhere" so that CUPS can check the device URI (set with the previous `-v` option) to automatically determine the correct PPD file to use for the printer. In modern situations, CUPS will just use IPP as explained below.

As stated previously, it is best to let CUPS automatically determine which PPD file to use for a particular print queue. However, the legacy `lpinfo` command can be used to query the locally installed PPD files to see what are available. Just provide the `--make-and-model` option for the printer you wish to install and the `-m` option:

```
$ lpinfo --make-and-model "HP Envy 4510" -m
hplip:0/ppd/hplip/HP/hp-envy_4510_series-hpijs.ppd HP Envy 4510 Series hpijs, 3.17.10
hplip:1/ppd/hplip/HP/hp-envy_4510_series-hpijs.ppd HP Envy 4510 Series hpijs, 3.17.10
hplip:2/ppd/hplip/HP/hp-envy_4510_series-hpijs.ppd HP Envy 4510 Series hpijs, 3.17.10
drv:///hpcups.crv/hp-envy_4510_series.ppd HP Envy 4510 Series, hpcups 3.17.10
everywhere IPP Everywhere
```

Note that the `lpinfo` command is deprecated. It is shown here as an example of listing out what print driver files a printer could use.

## WARNING

Future versions of CUPS have deprecated drivers and will instead focus on using IPP (*Internet Printing Protocol*) and standard file formats. The output of

the previous command illustrates this with the `everywhere` IPP Everywhere printing capability. IPP can perform the same tasks that a print driver is used for. IPP, just like the CUPS web interface, utilizes network port 631 with the TCP protocol.

A default printer can be set using the `lpoptions` command. This way, should the majority (or all) print jobs be sent to a particular printer then the one specified with the `lpoptions` command will be the default. Just specify the printer along with the `-d` option:

```
$ lpoptions -d ENVY-4510
```

## Managing Printers

Once a printer has been installed an administrator could use the web interface to manage the options that are available to the printer. A more direct approach to managing a printer is through the use of the `lpadmin` command.

One option is allowing the ability for a printer to be shared on the network. This can be achieved with the `printer-is-shared` option, and by specifying the printer with the `-p` option:

```
$ sudo lpadmin -p FRONT-DESK -o printer-is-shared=true
```

An administrator can also configure a print queue to only accept print jobs from specific users with each user separated by a comma:

```
$ sudo lpadmin -p FRONT-DESK -u allow:carol,frank,grace
```

Inversely, only specific users could be denied access to a specific print queue:

```
$ sudo lpadmin -p FRONT-DESK -u deny:dave
```

User groups could also be used to allow or deny access to a printer's queue provided that the group's name is preceded by an "at" (@) character:

```
$ sudo lpadmin -p FRONT-DESK -u deny:@sales,@marketing
```

A print queue can also have an error policy should it encounter issues printing a job. With the use

of policies, a print job could be aborted (`abort-job`) or another attempt to print it could occur at a later time (`retry-job`). Other policies include the ability to stop the printer immediately should an error occur (`stop-printer`) as well as the ability to retry the job immediately after a failure is detected (`retry-current-job`). Here is an example where the printer policy is set to abort the print job should an error occur on the `FRONT-DESK` printer:

```
$ sudo lpadmin -p FRONT-DESK -o printer-error-policy=abort-job
```

Be sure to review the manual pages for the `lpadmin` command located at `lpadmin(8)` for further details on the usage of this command.

## Submitting Print Jobs

Many desktop applications will allow you to submit print jobs from a menu item or by using the `Ctrl + p` keyboard shortcut. Should you find yourself on a Linux system that does not utilize a desktop environment, you can still send files to a printer by way of the legacy LPD/LPR commands.

The `lpr` (“line printer remote”) command is used to send a print job to a printer’s queue. In the command’s most basic form, a file name along with the `lpr` command is all that is required:

```
$ lpr report.txt
```

The above command will send the file `report.txt` to the default print queue for the system (as identified by the `/etc/cups/printers.conf` file).

If a CUPS installation has multiple printers installed then the `lpstat` command can be used to print out a list of available printers using the `-p` option and the `-d` option will indicate which is the default printer:

```
$ lpstat -p -d
printer FRONT-DESK is idle.  enabled since Mon 03 Aug 2020 10:33:07 AM EDT
printer PostScript_oc0303387803 disabled since Sat 07 Mar 2020 08:33:11 PM EST -
    reason unknown
printer ENVY-4510 is idle.  enabled since Fri 31 Jul 2020 10:08:31 AM EDT
system default destination: ENVY-4510
```

So in our example here, the `report.txt` file will be sent to the `ENVY-4510` printer as it is set as the default. Should the file need to be printed at a different printer, specify the printer along with

the `-P` option:

```
$ lpr -P FRONT-DESK report.txt
```

When a print job is submitted to CUPS the daemon will figure out which backend is best suited to handle the task. CUPS can make use of various printer drivers, filters, hardware port monitors and other software to properly render the document. There will be times when a user printing a document will need to make modifications to *how* the document should be printed. Many graphical applications make this task rather easy. There are also command line options that can be utilized to change how a document should be printed. When a print job is submitted via the command line, the `-o` switch (for “options”) can be used in conjunction with specific terms to adjust the document’s layout for printing. Here is a short list of commonly used options:

### landscape

The document is printed where the page is rotated 90 degrees clockwise. The option `orientation-requested=4` will achieve the same result.

### two-sided-long-edge

The printer will print the document in portrait mode on both sides of the paper, provided that the printer supports this capability.

### two-sided-short-edge

The printer will print the document in landscape mode on both sides of the paper, provided that the printer supports this capability.

### media

The printer will print the job to the specified media size. The media sizes available for a print job to use depend on the printer, but here is a list of common sizes:

Size Option	Purpose
A4	ISO A4
Letter	US Letter
Legal	US Legal
DL	ISO DL Envelope
COM10	US #10 Envelope

### collate

Collate the printed document. This is helpful if you have a multi-page document that will be

printed more than once as all of the pages for each document will then be printed in order. Set this option to either `true` to enable it or `false` to disable it.

### page-ranges

This option can be used to select a single page to print, or a specific set of pages to print from a document. An example would look like: `-o page-ranges=5-7,9,15`. This would print pages 5, 6 and 7 then pages 9 and 15.

### fit-to-page

Print the document so that the file is scaled to fit the paper. If no information about the page size is provided by the file to be printed, it is possible that the printed job will be scaled incorrectly and portions of the document could be scaled off the page or the document could be scaled too small.

### outputorder

Print the document in either `reverse` order or `normal` to start the printing on page one. If a printer prints its pages face-down, the default is for the order to be `-o outputorder=normal` whereas printers that print with their pages facing up will print with `-o outputorder=reverse`.

Taking a sampling of the options above, the following example command can be constructed:

```
$ lpr -P ACCOUNTING-LASERJET -o landscape -o media=A4 -o two-sided-short-edge finance-report.pdf
```

More than one copy of a document can be printed by using the number option in the following format: `-#N` where `N` is equal to the number of copies to print. Here is an example with the `collate` option where seven copies of a report are to be printed on the default printer:

```
$ lpr -#7 -o collate=true status-report.pdf
```

Aside from the `lpr` command, the `lp` command can also be used. Many of the options that are used with the `lpr` command can also be used with the `lp` command, but there are some differences. Be sure to consult the man page at `lp(1)` for reference. Here is how we can run the previous example `lpr` command using the syntax of the `lp` command while also specifying the destination printer with the `-d` option:

```
$ lp -d ACCOUNTING-LASERJET -n 7 -o collate=true status-report.pdf
```

## Managing Print Jobs

As stated earlier, each print job submitted to the print queue receives a job ID from CUPS. A user can view the print jobs that they have submitted with the `lpq` command. Passing in the `-a` option will show the queues of all printers that are managed by the CUPS installation:

```
$ lpq -a
Rank      Owner      Job      File(s)      Total Size
1st       carol      20       finance-report.pdf  5072 bytes
```

The same `lpstat` command used previously also has an option to view printer queues. The `-o` option by itself will show all print queues, or a print queue can be specified by name:

```
$ lp -o
ACCOUNTING-LASERJET-4          carol      19456   Wed 05 Aug 2020 04:29:44 PM EDT
```

The print job ID will be prepended with the name of the queue where the job was sent, then the name of the user that submitted the job, the file's size, and the time it was submitted.

Should a print job get stuck on a printer or a user wishes to cancel their print job, use the `lprm` command along with the job ID found from the `lpq` command:

```
$ lprm 20
```

All jobs in a print queue could be deleted at once by providing just a dash `-`:

```
$ lprm -
```

Alternatively, the CUPS `cancel` command could also be used by a user to stop their current print job:

```
$ cancel
```

A specific print job can be cancelled by its job ID prepended by the printer name:

```
$ cancel ACCOUNTING-LASERJET-20
```



A print job can also be moved from one print queue to another. This is often helpful should a printer stop responding or the document to be printed requires features available on a different printer. Take note that this procedure typically requires a user with elevated privileges. Using the same print job from the previous example, we could move it to the queue of the FRONT-DESK printer:

```
$ sudo lpmove ACCOUNTING-LASERJET-20 FRONT-DESK
```

## Removing Printers

To remove a printer, it is often helpful to first list out all of the printers that are currently managed by the CUPS service. This can be done with the `lpstat` command:

```
$ lpstat -v
device for FRONT-DESK: socket://192.168.150.24
device for ENVY-4510: socket://192.168.150.25
device for PostScript_oc0303387803: ///dev/null
```

The `-v` option not only lists out the printers but also where (and how) they are attached. It is good practice to first reject any new jobs going to the printer and provide a reason as to why the printer will not be accepting new jobs. This can be done with the following:

```
$ sudo cupsreject -r "Printer to be removed" FRONT-DESK
```

Note the use of `sudo` as this task requires a user with elevated privileges.

To remove a printer, we utilize the `lpadmin` command with the `-x` option to delete the printer:

```
$ sudo lpadmin -x FRONT-DESK
```

## Guided Exercises

1. A new printer was just installed on a local workstation named `office-mgr`. What command could be used to set this printer as the default for this workstation?

2. Which command and option would be used to determine what printers are available for printing from a workstation?

3. Using the `cancel` command, how would you remove a print job with ID 15 that is stuck in the queue for the printer named `office-mgr`?

4. You have a print job destined for a printer that does not have enough paper to print the full file. What command would you use to move the print job with ID 2 queued to print on the `FRONT-DESK` printer over to the print queue for the `ACCOUNTING-LASERJET` printer?

## Explorational Exercises

Using your distribution's package manager, install the `cups` and the `printer-driver-cups-pdf` packages. Note that if you are using a Red Hat based distribution (such as Fedora) the CUPS PDF driver is called `cups-pdf`. Also install the `cups-client` package to utilize the System V style printing commands We will use these packages to practice managing a CUPS printer without physically installing a real printer.

1. Verify that the CUPS daemon is running, then verify that the PDF printer is enabled and set to the default.

2. Run a command that will print the `/etc/services` file. You should now have a directory named `PDF` within your home directory.

3. Use a command that will only disable the printer, then run a separate command that shows all status information to verify that the PDF printer is disabled. Then try to print a copy of your `/etc/fstab` file. What happens?

4. Now try to print a copy of the `/etc/fstab` file to the PDF printer. What happens?

5. Cancel the print job, then remove the PDF printer.

## Summary

The CUPS daemon is a widely-used platform for printing to local and remote printers. While it supersedes the legacy LPD protocol it still provides backwards compatibility for its tools.

The files and commands discussed in this lesson were:

### **/etc/cups/cupsd.conf**

The primary configuration file for the CUPS service itself. This file also controls access to the web interface for CUPS.

### **/etc/printcap**

A legacy file used by LPD that contains a line for each printer connected to the system.

### **/etc/cups/printers.conf**

The configuration file used by CUPS for printer information.

The CUPS web interface, which on a default installation can be found at `http://localhost:631`. Remember that the default network port for the web interface is 631/TCP.

The following legacy LPD/LPR commands were also discussed:

### **lpadmin**

Used to install and remove printers and printer classes.

### **lpoptions**

Used to print out printer options and to modify a printer's settings.

### **lpstat**

Used to display status information of the printers connected to a CUPS installation.

### **lpr**

Used to submit print jobs to a printer's queue.

### **lp**

Used to submit print jobs to a printer's queue.

### **lpq**

This command lists out the print jobs within the print queue.

**lprm**

Used to cancel print jobs by ID. The ID for a job can be obtained with the output of the `lpq` command.

**cancel**

An alternative to the `lprm` command to cancel print jobs by their ID.

Be sure to review the following man pages for the various tools and utilities for cups: `lpadmin(8)`, `lpoptions(1)`, `lpr(1)`, `lpq(1)`, `lprm(1)`, `cancel(1)`, `lpstat(1)`, `cupsenable(8)` and `cupsaccept(8)`. Reviewing the online help documentation at <http://localhost:631/help> is also recommended.

## Answers to Guided Exercises

1. A new printer was just installed on a local workstation named `office-mgr`. What command could be used to set this printer as the default for this workstation?

```
$ lpoptions -d office-mgr
```

2. Which command and option would be used to determine what printers are available for printing from a workstation?

```
$ lpstat -p
```

The `-p` option lists out all available printers and if they are enabled for printing.

3. Using the `cancel` command, how would you remove a print job with ID 15 that is stuck in the queue for the printer named `office-mgr`?

```
$ cancel office-mgr-15
```

4. You have a print job destined for a printer that does not have enough paper to print the full file. What command would you use to move the print job with ID 2 queued to print on the `FRONT-DESK` printer over to the print queue for the `ACCOUNTING-LASERJET` printer?

```
$ sudo lpmove FRONT-DESK-2 ACCOUNTING-LASERJET
```

## Answers to Explorational Exercises

Using your distribution's package manager, install the `cups` and the `printer-driver-cups-pdf` packages. Note that if you are using a Red Hat based distribution (such as Fedora) the CUPS PDF driver is called `cups-pdf`. We will use these packages to practice managing a CUPS printer without physically installing a real printer.

1. Verify that the CUPS daemon is running, then verify that the PDF printer is enabled and set to the default.

One method to check the availability and status of the PDF printer would be to run the following command:

```
$ lpstat -p -d

printer PDF is idle.  enabled since Thu 25 Jun 2020 02:36:07 PM EDTi

system default destination: PDF
```

2. Run a command that will print the `/etc/services` file. You should now have a directory named `PDF` within your home directory.

```
$ lp -d PDF /etc/services
```

would work. You will now have a PDF version of this file within the `PDF` directory.

3. Use a command that will only disable the printer, then run a separate command that shows all status information to verify that the PDF printer is disabled.

```
$ sudo cupsdisable PDF
```

will disable the printer.

Next run the `lpstat -t` command to get a full listing of the printer's condition. It should look similar to the following output:

```
$ scheduler is running

system default destination: PDFi
```

```
device for PDF: cups-pdf:/  
  
PDF accepting requests since Wed 05 Aug 2020 04:19:15 PM EDTi  
  
printer PDF disabled since Wed 05 Aug 2020 04:19:15 PM EDT -  
  
Paused
```

4. Now try to print a copy of the `/etc/fstab` file to the PDF printer. What happens?

After attempting the command `lp -d PDF /etc/fstab` you should get output showing the job ID information. However, if you check the PDF folder in your home directory, the new file is not there. You can then check the print queue with the `lpstat -o` command, and you will find your job listed there.

5. Cancel the print job, then remove the PDF printer.

Using the output from the previous `lp` command, use the `cancel` command to delete the job. For example:

```
$ cancel PDF-4
```

Then run the `lpstat -o` command to verify that the job has been deleted.

Remove the PDF printer with the following: `sudo lpadmin -x PDF`. Then verify that the printer has been removed: `lpstat -a`.





**Linux  
Professional  
Institute**

## **Topic 109: Networking Fundamentals**



## 109.1 Fundamentals of internet protocols

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 109.1](#)

### Weight

4

### Key knowledge areas

- Demonstrate an understanding of network masks and CIDR notation.
- Knowledge of the differences between private and public "dotted quad" IP addresses.
- Knowledge about common TCP and UDP ports and services (20, 21, 22, 23, 25, 53, 80, 110, 123, 139, 143, 161, 162, 389, 443, 465, 514, 636, 993, 995).
- Knowledge about the differences and major features of UDP, TCP and ICMP.
- Knowledge of the major differences between IPv4 and IPv6.
- Knowledge of the basic features of IPv6.

### Partial list of the used files, terms and utilities

- `/etc/services`
- IPv4, IPv6
- Subnetting
- TCP, UDP, ICMP



Linux  
Professional  
Institute

# 109.1 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	109 Networking Fundamentals
<b>Objective:</b>	109.1 Fundamentals of internet protocols
<b>Lesson:</b>	1 of 2

## Introduction

The TCP/IP (*Transmission Control Protocol/Internet Protocol*) is a stack of protocols used to enable communication between computers. Despite the name, the stack consists of several protocols such as IP, TCP, UDP, ICMP, DNS, SMTP, ARP and others.

## IP (Internet Protocol)

The IP is the protocol responsible for the logical addressing of a host, enabling the packet to be sent from one host to another. For this each device on the network is assigned a unique IP address, and it is possible to assign more than one address to the same device.

In the version 4 of the IP protocol, usually called IPv4, the address is formed by a set of 32 bits separated into 4 groups of 8 bits, represented in decimal form, called “dotted quad”. For example:

### Binary format (4 groups of 8 bits)

```
11000000.10101000.00001010.00010100
```

## Decimal format

192.168.10.20

In IPv4, the values for each octet can range from 0 to 255, which is the equivalent of 11111111 in binary format.

## Address Classes

Theoretically, IP addresses are separated by classes, which are defined by the range of the first octet as shown in the table below:

Class	First Octect	Range	Example
A	1-126	1.0.0.0 – 126.255.255.255	10.25.13.10
B	128-191	128.0.0.0 – 191.255.255.255	141.150.200.1
C	192-223	192.0.0.0 – 223.255.255.255	200.178.12.242

## Public and Private IPs

As mentioned earlier, for communication to occur each device on the network must be associated with at least one unique IP address. However, if each device connected to the Internet in the world had a unique IP address, there would not be enough IPs (v4) for everyone. For this, *private* IP addresses were defined.

Private IPs are ranges of IP addresses that have been reserved for use in the internal (private) networks of companies, institutions, homes, etc. Within the same network, the use of an IP address remains unique. However, the same private IP address can be used within any private network.

Thus, on the Internet we have data traffic using public IP addresses, which are recognizable and routed over the Internet, while within private networks these reserved IP ranges are used. The router is responsible for converting traffic from the private network to the public network and vice versa.

The ranges of Private IPs, separated by classes, can be seen in the table below:

Class	First Octet	Range	Private IPs
A	1-126	1.0.0.0 – 126.255.255.255	10.0.0.0 – 10.255.255.255
B	128-191	128.0.0.0 – 191.255.255.255	172.16.0.0 – 172.31.255.255
C	192-223	192.0.0.0 – 223.255.255.255	192.168.0.0 – 192.168.255.255

## Converting from Decimal Format to Binary

For the subjects of this topic, it is important to know how to convert IP addresses between binary and decimal formats.

The conversion from decimal format to binary is done through consecutive divisions by 2. As an example, let's convert the value 105 by the following steps:

1. Dividing the value 105 by 2 we have:

```
105/2
Quotient = 52
Rest = 1
```

2. Divide the quotient sequentially by 2, until the quotient is equal to 1:

```
52/2
Rest = 0
Quotient = 26
```

```
26/2
Rest = 0
Quotient = 13
```

```
13/2
Rest = 1
Quotient = 6
```

```
6/2
```

Rest = 0  
 Quotient = 3

3/2  
 Rest = 1  
 Quotient = 1

3. Group the last quotient followed by the remainder of all divisions:

1101001

4. Fill in 0s to the left until 8 bits are completed:

01101001

5. In the end, we have that the value 105 in decimal is equal to 01101001 in binary.

## Converting from Binary Format to Decimal

In this example, we will use the binary value 10110000.

1. Each bit is associated with a value with a base power of two. The powers are started at 0, and are incremented from right to left. In this example we will have:

1	0	1	1	0	0	0	0
$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

2. When the bit is 1, we assign the value of the respective power, when the bit is 0 the result is 0.

1	0	1	1	0	0	0	0
$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
128	0	32	16	0	0	0	0

3. Add up all values:

$$128 + 32 + 16 = 176$$

4. Thus, 10110000 in binary is equal to 176 in decimal.

## Netmask

The network mask (or *netmask*) is used in conjunction with the IP address to determine which part of the IP represents the network and which represents the hosts. It has the same format as the IP address, that is, there are 32 bits in 4 groups of 8. For example:

Decimal	Binary	CIDR
255.0.0.0	11111111.00000000.00000000 0.00000000	8
255.255.0.0	11111111.11111111.00000000 0.00000000	16
255.255.255.0	11111111.11111111.11111111 1.00000000	24

Using the 255.255.0.0 mask as an example, it indicates that in the IP associated with it, the first 16 bits (2 first decimals) identify the network/subnet and the last 16 bits are used to uniquely identify the hosts within the network.

The CIDR (*Classless Inter-Domain Routing*) mentioned above is related to a simplified mask notation, which indicates the number of bits (1) associated with the network/subnet. This notation is commonly used to replace the decimal format, for example /24 instead of 255.255.255.0.

It is interesting to note that each class of IP has a standard mask, as follows:

Class	First Octet	Range	Default Mask
A	1-126	1.0.0.0 – 126.255.255.255	255.0.0.0 / 8
B	128-191	128.0.0.0 – 191.255.255.255	255.255.0.0 / 16
C	192-223	192.0.0.0 – 223.255.255.255	255.255.255.0 / 24

However, this pattern does not mean that this is the mask that will always be used. It is possible to use any mask with any IP address, as we will see below.

Here are some examples of using IPs and Masks:

```
192.168.8.12 / 255.255.255.0 / 24
```

**Range**`192.168.8.0 - 192.168.8.255`**Network Address**`192.168.8.0`**Broadcast Address**`192.168.8.255`**Hosts**`192.168.8.1 - 192.168.8.254`

In this case we have the first 3 digits (first 24 bits) of the IP address define the network and the final digit identifies the addresses of the hosts, that is, the range of this network goes from `192.168.8.0` to `192.168.8.255`.

We now have two important concepts: Every network/subnet has 2 reserved addresses, the first address in the range is called the *network address*. In this case `192.168.8.0`, which is used to identify the network/subnet itself. The last address in the range is called the *broadcast address*, in this case `192.168.8.255`. This destination address is used to send the same message (packet) to all IP hosts on that network/subnet.

The network and broadcast addresses cannot be used by the machines on the network. Therefore, the list of IPs that can be effectively configured ranges from `192.168.8.1` to `192.168.8.254`.

Now the example of the same IP, but with a different mask:

```
192.168.8.12 / 255.255.0.0 / 16
```

**Range**`192.168.0.0 - 192.168.255.255`**Network Address**`192.168.0.0`**Broadcast Address**`192.168.255.255`**Hosts**`192.168.0.1 - 192.168.255.254`



See how the different mask changes the range of IPs that are within the same network/subnet.

The division of networks by masks is not restricted to the default values (8, 16, 24). We can create subdivisions as desired, adding or removing bits in the network identification, creating the new subnets.

For example:

$$11111111.11111111.11111111.00000000 = 255.255.255.0 = 24$$

If we want to subdivide the network above into 2, just add another bit to the network identification in the mask, like this:

$$11111111.11111111.11111111.10000000 = 255.255.255.128 = 25$$

We have then the following subnets:

$$192.168.8.0 - 192.168.8.127$$

$$192.168.8.128 - 192.168.8.255$$

If we further increase the subdivision of the network:

$$11111111.11111111.11111111.11000000 = 255.255.255.192 = 26$$

We will have:

$$192.168.8.0 - 192.168.8.63$$

$$192.168.8.64 - 192.168.8.127$$

$$192.168.8.128 - 192.168.8.191$$

$$192.168.8.192 - 192.168.8.255$$

Note that in each subnet we will have the reserved network (the first in the range) and broadcast (the last in the range) addresses, so the more the network is subdivided, the fewer IPs can be effectively used by the hosts.

## Identifying the Network and Broadcast Addresses

Through an IP Address and a Mask, we can identify the network address and the broadcast

address, and thus define the range of IPs for the network/subnet.

The network address is obtained by using a “Logical AND” between the IP address and the mask in their binary formats. Let’s take the example using IP 192 . 168 . 8 . 12 and mask 255 . 255 . 255 . 192.

Converting from decimal to binary format, as we saw earlier, we have:

```
11000000.10101000.00001000.00001100 (192.168.8.12)
11111111.11111111.11111111.11000000 (255.255.255.192)
```

With “Logical AND”, we have 1 and 1 = 1, 0 and 0 = 0, 1 and 0 = 0, so:

```
11000000.10101000.00001000.00001100 (192.168.8.12)
11111111.11111111.11111111.11000000 (255.255.255.192)
11000000.10101000.00001000.00000000
```

So the network address for that subnet is 192 . 168 . 8 . 0.

Now to obtain the broadcast address, we must use the network address where all bits related to the host address to 1:

```
11000000.10101000.00001000.00000000 (192.168.8.0)
11111111.11111111.11111111.11000000 (255.255.255.192)
11000000.10101000.00001000.00111111
```

The broadcast address is then 192 . 168 . 8 . 63.

In conclusion, we have:

```
192.168.8.12 / 255.255.255.192 / 26
```

### Range

```
192.168.8.0 - 192.168.8.63
```

### Network Address

```
192.168.8.0
```

### Broadcast Address

```
192.168.8.63
```

## Hosts

192.168.8.1 – 192.168.8.62

## Default Route

As we have seen so far, machines that are within the same logical network/subnet can communicate directly via the IP protocol.

But let's consider the example below:

### Network 1

192.168.10.0/24

### Network 2

192.168.200.0/24

In this case, the 192.168.10.20 machine cannot directly send a packet to the 192.168.200.100, as they are on different logical networks.

To enable this communication a router (or a set of routers) is used. A router in this configuration can also be called a gateway as it provides a gateway between two networks. This device has access to both networks as it is configured with IPs from both networks. For example 192.168.10.1 and 192.168.200.1, and for this reason it manages to be the intermediary in this communication.

To enable this, each host on the network must have configured what is called the *default route*. The default route indicates the IP to which all packets whose destination is an IP that is not part of the host's logical network must be sent.

In the example above, the default route for machines on the 192.168.10.0/24 network will be the IP 192.168.10.1, which is the router/gateway IP, while the default route for machines on the 192.168.200.0/24 network will be 192.168.200.1.

The default route is also used so that machines on the private network (LAN) have access to the Internet (WAN), through a router.

## Guided Exercises

1. Using the IP 172 . 16 . 30 . 230 and netmask 255 . 255 . 255 . 224, identify:

The CIDR notation for the netmask	
Network address	
Broadcast address	
Number of IPs that can be used for hosts in this subnet	

2. Which setting is required on a host to allow an IP communication with a host in a different logical network?

## Explorational Exercises

1. Why are the IP ranges starting with 127 and the range after 224 not included in the IP address classes A, B or C?

2. One of the fields belonging to an IP packet that is very important is TTL (*Time To Live*). What is the function of this field and how does it work?

3. Explain the function of NAT and when it is used.

## Summary

This lesson covered the main concepts for the IPv4 protocol, which is responsible for enabling communication between hosts on a network.

The main operations that the professional must know in order to convert the IPs in different formats, and to be able to analyze and perform the logical configurations on networks and subnets were also studied.

The following subjects were addressed:

- IP addresses classes
- Public and private IPs
- How to convert IPs from decimal to binary format, and vice versa
- The network mask (netmask)
- How to identify the network and broadcast addresses from IP and netmask
- Default route

## Answers to Guided Exercises

1. Using the IP 172.16.30.230 and netmask 255.255.255.224, identify:

The CIDR notation for the netmask	27
Network address	172.16.30.224
Broadcast address	172.16.30.255
Number of IPs that can be used for hosts in this subnet	30

2. Which setting is required on a host to allow an IP communication with a host in a different logical network?

Default route

## Answers to Explorational Exercises

1. Why are the IP ranges starting with 127 and the range after 224 not included in the IP address classes A, B or C?

The range that starts with 127 is reserved for loopback addresses, used for tests and internal communication between processes, such as the address 127.0.0.1. In addition, addresses above 224 are also not used as host addresses, but for multicast and other purposes.

2. One of the fields belonging to an IP packet that is very important is TTL (*Time To Live*). What is the function of this field and how does it work?

TTL defines the lifetime of a packet. This is implemented through a counter in which the initial value defined at the source is decremented in each gateway/router through which the packet passes, which is also called a “hop”. If this counter reaches 0 the packet is discarded.

3. Explain the function of NAT and when it is used.

The NAT (*Network Address Translation*) feature allows hosts on an internal network, which uses private IPs, to have access to the Internet as if they were directly connected to it, with the Public IP used on the gateway.





## 109.1 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	109 Networking Fundamentals
<b>Objective:</b>	109.1 Fundamentals of internet protocols
<b>Lesson:</b>	2 of 2

### Introduction

At the beginning of this subtopic we saw that the TCP/IP stack is composed of a series of different protocols. So far we have studied the IP protocol, which allows communication between machines through IP addresses, masks, routes, etc.

For a host to be able to access a service available on another host, in addition to the IP addressing protocol at the network layer, it will be necessary to use a protocol at the transport layer such as the TCP and UDP protocols.

These protocols carry out this communication through network ports. So in addition to defining a source and destination IP, source and destination ports will be used to access a service.

The port is identified by a 16-bit field thus providing a limit of 65,535 possible ports. The services (destination) use ports 1 to 1023, which are called *privileged ports* because they have root access to the system. The origin of the connection will use the range of ports from 1024 to 65,535, called *non-privileged ports*, or socket ports.

The ports used by each type of service are standardized and controlled by IANA (*Internet Assigned Numbers Authority*). This means that on any system, port 22 is used by the SSH service, port 80 by

the HTTP service and so on.

The table below contains the main services and their respective ports.

Port	Service
20	FTP (data)
21	FTP (control)
22	SSH (Secure Socket Shell)
23	Telnet (Remote connection without encryption)
25	SMTP (Simple Mail Transfer Protocol), Sending Mails
53	DNS (Domain Name System)
80	HTTP (Hypertext Transfer Protocol)
110	POP3 (Post Office Protocol), Receiving Mails
123	NTP (Network Time Protocol)
139	Netbios
143	IMAP (Internet Message Access Protocol), Accessing Mails
161	SNMP (Simple Network Management Protocol)
162	SNMPTRAP, SNMP Notifications
389	LDAP (Lightweight Directory Access Protocol)
443	HTTPS (Secure HTTP)
465	SMTPTS (Secure SMTP)
514	RSH (Remote Shell)
636	LDAPS (Secure LDAP)
993	IMAPS (Secure IMAP)
995	POP3S (Secure POP3)

On a Linux system, standard service ports are listed in the `/etc/services` file.

The identification of the desired destination port in a connection is done using the character `:` (colon) after the IPv4 address. Thus, when seeking access to the HTTPS service that is served by the IP host `200.216.10.15`, the client must send the request to the destination

200.216.10.15:443.

The services listed above, and all others, use a transport protocol according to the characteristics required by the service, where TCP and UDP are the main ones.

## Transmission Control Protocol (TCP)

TCP is a connection-oriented transport protocol. This means that a connection is established between the client through the socket port, and the service through the service standard port. The protocol is in charge of ensuring that all packets are delivered properly, verifying the integrity and order of the packets, including the re-transmission of packets lost due to network errors.

Thus the application does not need to implement this data flow control as it is already guaranteed by the TCP protocol.

## User Datagram Protocol (UDP)

UDP establishes a connection between the client and the service, but does not control the data transmission of that connection. In other words, it does not check if packages have been lost, or if they are out of order, etc. The application is responsible for implementing the controls that are necessary.

As there is less control, UDP enables better performance in the data flow which is important for some types of services.

## Internet Control Message Protocol (ICMP)

ICMP is a network layer protocol in the TCP/IP stack and its main function is to analyze and control network elements, making it possible, for example:

- Traffic volume control
- Detection of unreachable destinations
- Route redirection
- Checking the status of remote hosts

It is the protocol used by the `ping` command, which will be studied in another subtopic.

## IPv6

So far we have studied version 4 of the IP protocol, i.e. IPv4. This has been the standard version

used in all network and Internet environments. However it has limitations especially in regards to the number of available addresses, and with an already current reality that all devices will be somehow connected to the Internet (see IoT), it is becoming increasingly common to use version 6 of the IP protocol, commonly written as IPv6.

IPv6 brings a series of changes, new implementations and features, as well as a new representation of the address itself.

Each IPv6 address has 128 bits, divided into 8 groups of 16 bits, represented by hexadecimal values.

For example:

```
2001:0db8:85a3:08d3:1319:8a2e:0370:7344
```

## Abbreviations

IPv6 defines ways to shorten addresses in some situations. Let's review the following address:

```
2001:0db8:85a3:0000:0000:0000:0000:7344
```

The first possibility is to reduce strings from `0000` to just `0`, resulting in:

```
2001:0db8:85a3:0:0:0:0:7344
```

In addition, in case of group strings with a value of `0`, they can be omitted, as follows:

```
2001:0db8:85a3::7344
```

However, this last abbreviation can only be done once in the address. See the example:

```
2001:0db8:85a3:0000:0000:1319:0000:7344
```

```
2001:0db8:85a3:0:0:1319:0:7344
```

```
2001:0db8:85a3::1319:0:7344
```

## IPv6 Address Types

IPv6 classifies addresses into 3 types:

### Unicast

Identifies a single network interface. By default, the 64 bits on the left identify the network, and the 64 bits on the right identify the interface.

### Multicast

Identifies a set of network interfaces. A packet sent to a multicast address will be sent to all interfaces that belong to that group. Although similar, it should not be confused with broadcast, which does not exist in the IPv6 protocol.

### Anycast

This also identifies a set of interfaces on the network, but the packet forwarded to an *anycast* address will be delivered to only one address in that set, not everyone.

## Differences between IPv4 and IPv6

In addition to the address several other differences can be pointed out between versions 4 and 6 of the IP. Here are some of them:

- Service ports follow the same standards and protocols (TCP, UDP), the difference is only in the representation of the IP and port set. In IPv6 the IP address must be protected with [] (brackets):

### IPv4

```
200.216.10.15:443
```

### IPv6

```
[2001:0db8:85a3:08d3:1319:8a2e:0370:7344]:443
```

- IPv6 does not implement the broadcast feature exactly as it exists in IPv4. However the same result can be achieved by sending the packet to the address `ff02::1`, reaching all hosts on the local network. Something similar to using `224.0.0.1` on IPv4 for multicasting as a destination.
- Through the SLAAC (*Stateless Address Autoconfiguration*) feature, IPv6 hosts are able to self-configure.
- The TTL (*Time to Live*) field of IPv4 has been replaced by the “Hop Limit” in the IPv6 header.
- All IPv6 interfaces have a local address, called link-local address, prefixed with `fe80::/10`.
- IPv6 implements the *Neighbor Discovery Protocol* (NDP), which is similar to the ARP used by

IPv4, but with much more functionality.

## Guided Exercises

1. Which port is the default for the SMTP protocol?

2. How many different ports are available in a system?

3. Which transport protocol ensures that all packets are delivered properly, verifying the integrity and the order of the packets?

4. Which type of IPv6 address is used to send a packet to all interfaces that belong to group of hosts?

## Explorational Exercises

1. Mention 4 examples of services that use the TCP protocol by default.


2. What is the name of the field on IPv6 header package that implement the same resource of TTL on IPv4?

--

3. What kind of information Neighbor Discovery Protocol (NDP) is able to discover?

--



## Summary

This lesson covered the main transport protocols and services used on TCP/IP stack.

Another important topic was the version 6 of IP Protocol, including the IPv6 addresses and the main differences with IPv4.

The following subjects were addressed:

- The correlation between Port numbers and Services
- TCP (Transmission Control Protocol)
- UDP (User Datagram Protocol)
- ICMP (Internet Control Message Protocol)
- The IPv6 address and how it can be abbreviated
- IPv6 address types
- Main differences between IPv4 and IPv6

## Answers to Guided Exercises

1. Which port is the default for the SMTP protocol?

25

2. How many different ports are available in a system?

65535

3. Which transport protocol ensures that all packets are delivered properly, verifying the integrity and the order of the packets?

TCP

4. Which type of IPv6 address is used to sent a packet to all interfaces that belong to group of hosts?

Multicast

# Answers to Explorational Exercises

1. Mention 4 examples of services that use the TCP protocol by default.

FTP, SMTP, HTTP, POP3, IMAP, SSH

2. What is the name of the field on IPv6 header package that implement the same resource of TTL on IPv4?

Hop Limit

3. What kind of information Neighbor Discovery Protocol (NDP) is able to discover?

NDP is able to obtain various information from the network, including other nodes, duplicate addresses, routes, DNS servers, gateways, etc.



## 109.2 Persistent network configuration

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 109.2](#)

### Weight

4

### Key knowledge areas

- Understand basic TCP/IP host configuration
- Configure ethernet and wi-fi network configuration using NetworkManager
- Awareness of systemd-networkd

### Partial list of the used files, terms and utilities

- `/etc/hostname`
- `/etc/hosts`
- `/etc/nsswitch.conf`
- `/etc/resolv.conf`
- `nmcli`
- `hostnamectl`
- `ifup`
- `ifdown`



# 109.2 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	109 Networking Fundamentals
<b>Objective:</b>	109.2 Persistent network configuration
<b>Lesson:</b>	1 of 2

## Introduction

In any TCP/IP network, every node must configure its network adapter to match the network requirements, otherwise they will not be able to communicate with each other. Therefore, the system administrator must provide the basic configuration so the operating system will be able to setup the appropriate network interface, as well as to identify itself and the basic features of the network every time it boots.

Network settings are agnostic in regard to operating systems, but the latter have their own methods to store and apply these settings. Linux systems rely on configurations stored in plain text files under the `/etc` directory to bring up network connectivity during boot time. It is worth knowing how these files are used to avoid connectivity loss due to local misconfiguration.

## The Network Interface

*Network interface* is the term by which the operating system refers to the communication channel configured to work with the network hardware attached to the system, such as an ethernet or wi-fi device. The exception to this is the *loopback* interface, which the operating system uses when it needs to establish a connection with itself, but the main purpose of a network interface is to

provide a route through which local data can be sent and remote data can be received. Unless the network interface is properly configured, the operating system will not be able to communicate with other machines in the network.

For most cases, the correct interface settings are either defined by default or customized during the installation of the operating system. Nevertheless, these settings often need to be inspected or even modified when the communication isn't working properly or when the interface's behavior requires customization.

There are many Linux commands to list which network interfaces are present on the system, but not all of them are available in all distributions. Command `ip`, however, is part of the basic set of networking tools bundled with all Linux distributions and can be used to list the network interfaces. The complete command to show the interfaces is `ip link show`:

```
$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp3s5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT
group default qlen 1000
    link/ether 00:16:3e:8d:2b:5b brd ff:ff:ff:ff:ff:ff
```

If available, command `nmcli device` can also be used:

```
$ nmcli device
DEVICE      TYPE      STATE      CONNECTION
enp3s5      ethernet  connected  Gigabit Powerline Adapter
lo          loopback  unmanaged  --
```

The commands shown in the examples do not modify any settings in the system, so they can be executed by an unprivileged user. Both commands list two network interfaces: `lo` (the loopback interface) and `enp3s5` (an ethernet interface).

Desktops and laptops running Linux usually have two or three predefined network interfaces, one for the loopback virtual interface and the others assigned to the network hardware found by the system. Servers and network appliances running Linux, on the other hand, may have tens of network interfaces, but the same principles apply to all of them. The abstraction provided by the operating system allows for the setup of network interfaces using the same methods, regardless of the underlying hardware.

However, knowing the details about the underlying hardware of an interface can be useful to

better understand what is going on when the communication is not working as expected. In a system where many network interfaces are available, it could not be obvious which one corresponds to the wi-fi and which one corresponds to the ethernet, for example. For this reason, Linux uses an interface naming convention that helps identify which network interface corresponds to which device and port.

## Interface Names

Older Linux distributions named ethernet network interfaces as `eth0`, `eth1`, etc., numbered according to the order in which the kernel identifies the devices. The wireless interfaces were named `wlan0`, `wlan1`, etc. This naming convention, however, does not clarify which specific ethernet port matches with the interface `eth0`, for example. Depending on how the hardware was detected, it was even possible for two network interfaces to swap names after a reboot.

To overcome this ambiguity, more recent Linux systems employ a predictable naming convention for network interfaces, making up a closer relationship between the interface name and the underlying hardware connection.

In Linux distributions that use the systemd naming scheme, all interface names start with a two-character prefix that signifies the interface type:

### **en**

Ethernet

### **ib**

InfiniBand

### **sl**

Serial line IP (slip)

### **wl**

Wireless local area network (WLAN)

### **ww**

Wireless wide area network (WWAN)

From higher to lower priority, the following rules are used by the operating system to name and number the network interfaces:

1. Name the interface after the index provided by the BIOS or by the firmware of embedded devices, e.g. `eno1`.

2. Name the interface after the PCI express slot index, as given by the BIOS or firmware, e.g. `ens1`.
3. Name the interface after its address at the corresponding bus, e.g. `enp3s5`.
4. Name the interface after the interface's MAC address, e.g. `enx78e7d1ea46da`.
5. Name the interface using the legacy convention, e.g. `eth0`.

It is correct to assume, for example, that the network interface `enp3s5` was so named because it did not fit the first two naming methods, so its address in the corresponding bus and slot was used instead. The device address `03:05.0`, found in the output of the `lspci` command, reveals the associate device:

```
$ lspci | fgrep Ethernet
03:05.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8110SC/8169SC Gigabit
Ethernet (rev 10)
```

Network interfaces are created by the Linux kernel itself, but there are many commands that can be used to interact with them. Normally, the configuration happens automatically and there is no need to change the settings manually. Nonetheless, with the name of the interface, it is possible to tell the kernel how to proceed in configuring it if necessary.

## Interface Management

Over the years, several programs have been developed to interact with the networking features provided by the Linux kernel. Although the old `ifconfig` command can still be used to do simple interface configurations and queries, it is now deprecated due to its limited support of non-ethernet interfaces. The `ifconfig` command was superseded by the command `ip`, which is capable of managing many other aspects of TCP/IP interfaces, like routes and tunnels.

The many capabilities of the `ip` command can be overkill for most ordinary tasks, so there are auxiliary commands to facilitate the activation and configuration of the network interfaces. Commands `ifup` and `ifdown` may be used to configure network interfaces based on interface definitions found in the file `/etc/network/interfaces`. Although they can be invoked manually, these commands are normally executed automatically during system boot.

All network interfaces managed by `ifup` and `ifdown` should be listed in the `/etc/network/interfaces` file. The format used in the file is straightforward: lines beginning with the word `auto` are used to identify the physical interfaces to be brought up when `ifup` is executed with the `-a` option. The interface name should follow the word `auto` on the same line. All interfaces marked `auto` are brought up at boot time, in the order they are listed.



**WARNING**

Network configuration methods used by `ifup` and `ifdown` are not standardized throughout all Linux distributions. CentOS, for example, keeps the interface settings in individual files in the `/etc/sysconfig/network-scripts/` directory and the configuration format used in them is slightly different from the format used in `/etc/network/interfaces`.

The actual interface configuration is written in another line, starting with the word `iface`, followed by the interface name, the name of the address family that the interface uses and the name of the method used to configure the interface. The following example shows a basic configuration file for interfaces `lo` (loopback) and `enp3s5`:

```
auto lo
iface lo inet loopback

auto enp3s5
iface enp3s5 inet dhcp
```

The address family should be `inet` for TCP/IP networking, but there is also support for IPX networking (`ipx`), and IPv6 networking (`inet6`). Loopback interfaces use the `loopback` configuration method. With the `dhcp` method, the interface will use the IP settings provided by the network's DHCP server. The settings from the example configuration allow the execution of command `ifup` using interface name `enp3s5` as its argument:

```
# ifup enp3s5
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp3s5/00:16:3e:8d:2b:5b
Sending on   LPF/enp3s5/00:16:3e:8d:2b:5b
Sending on   Socket/fallback
DHCPDISCOVER on enp3s5 to 255.255.255.255 port 67 interval 4
DHCPOFFER of 10.90.170.158 from 10.90.170.1
DHCPREQUEST for 10.90.170.158 on enp3s5 to 255.255.255.255 port 67
DHCPACK of 10.90.170.158 from 10.90.170.1
bound to 10.90.170.158 -- renewal in 1616 seconds.
```

In this example, the method chosen for the `enp3s5` interface was `dhcp`, so the command `ifup` called a DHCP client program to obtain the IP settings from the DHCP server. Likewise, command `ifdown enp3s5` can be used to turn the interface off.

In networks without a DHCP server, the `static` method could be used instead and the IP settings provided manually in `/etc/network/interfaces`. For example:

```
iface enp3s5 inet static
    address 192.168.1.2/24
    gateway 192.168.1.1
```

Interfaces using the `static` method do not need a corresponding `auto` directive, as they are brought up whenever the network hardware is detected.

If the same interface has more than one `iface` entry, then all of the configured addresses and options will be applied when bringing up that interface. This is useful to configure both IPv4 and IPv6 addresses on the same interface, as well as to configure multiple addresses of the same type on a single interface.

## Local and Remote Names

A working TCP/IP setup is just the first step towards full network usability. In addition to being able to identify nodes on the network by their IP numbers, the system must be able to identify them with names more easily understood by human beings.

The name by which the system identifies itself is customizable and it is good practice to define it, even if the machine is not intended to join a network. The local name often matches the network name of the machine, but this isn't necessarily always true. If the file `/etc/hostname` exists, the operating system will use the contents of the first line as its local name, thereafter simply called the *hostname*. Lines starting with `#` inside `/etc/hostname` are ignored.

The `/etc/hostname` file can be edited directly, but the machine's hostname can also be defined with the `hostnamectl` command. When supplied with sub-command `set-hostname`, command `hostnamectl` will take the name given as an argument and write it in `/etc/hostname`:

```
# hostnamectl set-hostname storage
# cat /etc/hostname
storage
```

The hostname defined in `/etc/hostname` is the *static* hostname, that is, the name which is used to initialize the system's hostname at boot. The static hostname may be a free-form string up to 64 characters in length. However, it is recommended that it consists only of ASCII lower-case characters and no spaces or dots. It should also limit itself to the format allowed for DNS domain name labels, even though this is not a strict requirement.

Command `hostnamectl` can set two other types of hostnames in addition to the static hostname:

### Pretty hostname

Unlike the static hostname, the pretty hostname may include all kinds of special characters. It can be used to set a more descriptive name for the machine, e.g. “LAN Shared Storage”:

```
# hostnamectl --pretty set-hostname "LAN Shared Storage"
```

### Transient hostname

Used when the static hostname is not set or when it is the default `localhost` name. The transient hostname is normally the name set together with other automatic configurations, but it can also be modified by the command `hostnamectl`, e.g.

```
# hostnamectl --transient set-hostname generic-host
```

If neither the `--pretty` nor `--transient` option is used, then all three hostname types will be set to the given name. To set the static hostname, but not the pretty and transient names, the option `--static` should be used instead. In all cases, only the static hostname is stored in the `/etc/hostname` file. Command `hostnamectl` can also be used to display various descriptive and identity bits of information about the running system:

```
$ hostnamectl status
  Static hostname: storage
  Pretty hostname: LAN Shared Storage
  Transient hostname: generic-host
    Icon name: computer-server
    Chassis: server
  Machine ID: d91962a957f749bbaf16da3c9c86e093
  Boot ID: 8c11dcab9c3d4f5aa53f4f4e8fdc6318
  Operating System: Debian GNU/Linux 10 (buster)
    Kernel: Linux 4.19.0-8-amd64
  Architecture: x86-64
```

This is the default action of the `hostnamectl` command, so the `status` sub-command can be omitted.

Regarding the name of the remote network nodes, there are two basic ways the operating system can implement to match names and IP numbers: to use a local source or to use a remote server to translate names into IP numbers and vice versa. The methods can be complementary to each

other and their priority order is defined in the *Name Service Switch* configuration file: `/etc/nsswitch.conf`. This file is used by the system and applications to determine not only the sources for name-IP matches, but also the sources from which to obtain name-service information in a range of categories, called *databases*.

The *hosts* database keeps track of the mapping between host names and host numbers. The line inside `/etc/nsswitch.conf` beginning with `hosts` defines the services accountable for providing the associations for it:

```
hosts: files dns
```

In this example entry, `files` and `dns` are the service names that specify how the lookup process for host names will work. First, the system will look for matches in local files, then it will ask the DNS service for matches.

The local file for the hosts database is `/etc/hosts`, a simple text file that associates IP addresses with hostnames, one line per IP address, e.g.:

```
127.0.0.1 localhost
```

The IP number `127.0.0.1` is the default address for the loopback interface, hence its association with the *localhost* name.

It is also possible to bind optional aliases to the same IP. Aliases can provide alternate spellings, shorter hostnames and should be added at the end of the line, for example:

```
192.168.1.10 foo.mydomain.org foo
```

The formatting rules for the `/etc/hosts` file are:

- Fields of the entry are separated by any number of blanks and/or tab characters.
- Text from a `#` character until the end of the line is a comment and is ignored.
- Host names may contain only alphanumeric characters, minus signs and periods.
- Host names must begin with an alphabetic character and end with an alphanumeric character.

IPv6 addresses may also be added to `/etc/hosts`. The following entry refers to the IPv6 loopback address:

```
:::1 localhost ip6-localhost ip6-loopback
```

Following the `files` service specification, the `dns` specification tells the system to ask a DNS service for the desired name/IP association. The set of routines responsible for this method is called the *resolver* and its configuration file is `/etc/resolv.conf`. The following example shows a generic `/etc/resolv.conf` containing entries for Google's public DNS servers:

```
nameserver 8.8.4.4
nameserver 8.8.8.8
```

As shown in the example, the `nameserver` keyword indicates the IP address of the DNS server. Only one `nameserver` is required, but up to three `nameservers` can be given. The supplementary ones will be used as a fallback. If no `nameserver` entries are present, the default behaviour is to use the name server on the local machine.

The resolver can be configured to automatically add the domain to names before consulting them on the name server. For example:

```
nameserver 8.8.4.4
nameserver 8.8.8.8
domain mydomain.org
search mydomain.net mydomain.com
```

The `domain` entry sets `mydomain.org` as the local domain name, so queries for names within this domain will be allowed to use short names relative to the local domain. The `search` entry has a similar purpose, but it accepts a list of domains to try when a short name is provided. By default, it contains only the local domain name.

## Guided Exercises

1. What commands can be used to list the network adapters present in the system?

2. What is the type of network adapter whose interface name is `wlo1`?

3. What role does the file `/etc/network/interfaces` play during boot time?

4. What entry in `/etc/network/interfaces` configures interface `eno1` to obtain its IP settings with DHCP?

## Explorational Exercises

1. How could the `hostnamectl` command be used to change only the *static* hostname of the local machine to `firewall`?

2. What details other than hostnames can be modified by command `hostnamectl`?

3. What entry in `/etc/hosts` associates both names `firewall` and `router` with IP `10.8.0.1`?

4. How could the `/etc/resolv.conf` file be modified in order to send all DNS requests to `1.1.1.1`?

## Summary

This lesson covers how to make persistent changes to the local network configuration using standard Linux files and commands. Linux expects the TCP/IP settings to be in specific places and it may be necessary to change them when the default settings are not appropriate. The lesson goes through the following topics:

- How Linux identifies network interfaces.
- Interface activation during boot and basic IP configuration.
- How the operating system associates names with hosts.

The concepts, commands and procedures addressed were:

- Interface naming conventions.
- Listing network interfaces with `ip` and `nmcli`.
- Interface activation with `ifup` and `ifdown`.
- Command `hostnamectl` and the `/etc/hostname` file.
- Files `/etc/nsswitch.conf`, `/etc/hosts` and `/etc/resolv.conf`.



## Answers to Guided Exercises

1. What commands can be used to list the network adapters present in the system?

Commands `ip link show`, `nmcli device` and the legacy `ifconfig`.

2. What is the type of a network adapter whose interface name is `wl01`?

The name starts with `wl`, so it is a wireless LAN adapter.

3. What role does the file `/etc/network/interfaces` play during boot time?

It has the configurations used by command `ifup` to activate the corresponding interfaces during boot time.

4. What entry in `/etc/network/interfaces` configures interface `eno1` to obtain its IP settings with DHCP?

The line `iface eno1 inet dhcp`.

## Answers to Explorational Exercises

1. How could the `hostnamectl` command be used to change only the *static* hostname of the local machine to `firewall`?

With the `--static` option: `hostnamectl --static set-hostname firewall`.

2. What details other than hostnames can be modified by command `hostnamectl`?

`hostnamectl` can also set the default icon for the local machine, its chassis type, the location and the deployment environment.

3. What entry in `/etc/hosts` associates both names `firewall` and `router` with IP `10.8.0.1`?

The line `10.8.0.1 firewall router`.

4. How could the `/etc/resolv.conf` file be modified in order to send all DNS requests to `1.1.1.1`?

Using `nameserver 1.1.1.1` as its only nameserver entry.



## 109.2 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	109 Networking Fundamentals
<b>Objective:</b>	109.2 Persistent network configuration
<b>Lesson:</b>	2 of 2

### Introduction

Linux supports virtually every network technology used to connect servers, containers, virtual machines, desktops and mobile devices. The connections between all these network nodes can be dynamic and heterogeneous, thus requiring appropriate management by the operating system running in them.

In the past, distributions developed their own customized solutions for managing dynamic network infrastructure. Today, tools like *NetworkManager* and *systemd* provide more comprehensive and integrated features to meet all the specific demands.

### NetworkManager

Most Linux distributions adopt the *NetworkManager* service daemon to configure and control the system's network connections. NetworkManager's purpose is to make the network configuration as simple and automatic as possible. When using DHCP, for example, NetworkManager arranges route changes, IP address fetching and updates to the local list of DNS servers, if necessary. When both wired and wireless connections are available, NetworkManager prioritizes the wired connection by default. NetworkManager will try to keep at least one connection active all the time,

whenever it is possible.

**NOTE** A request using DHCP (*Dynamic Host Configuration Protocol*) is usually sent through the network adapter as soon as the link to the network is established. The DHCP server that is active on the network then responds with the settings (IP address, network mask, default route, etc.) which the requester must use to communicate via IP protocol.

By default, the NetworkManager daemon controls the network interfaces not mentioned in the `/etc/network/interfaces` file. It does so to not interfere with other configuration methods that may be present as well, thus modifying the unattended interfaces only.

The NetworkManager service runs in the background with root privileges and triggers the necessary actions to keep the system online. Ordinary users can create and modify network connections with client applications that, albeit not having root privileges themselves, are capable of communicating with the underlying service in order to perform the requested actions.

Client applications for NetworkManager are available for both the command line and the graphical environment. For the latter, the client application comes as an accessory of the desktop environment (under names like, *nm-tray*, *network-manager-gnome*, *nm-applet* or *plasma-nm*) and it is usually accessible through an indicator icon at the corner of the desktop bar or from the system configuration utility.

In the command line, NetworkManager itself provides two client programs: `nmcli` and `nmtui`. Both programs have the same basic features, but `nmtui` has a curses-based interface while `nmcli` is a more comprehensive command that can also be used in scripts. Command `nmcli` separates all network related properties controlled by NetworkManager in categories called *objects*:

### **general**

NetworkManager's general status and operations.

### **networking**

Overall networking control.

### **radio**

NetworkManager radio switches.

### **connection**

NetworkManager's connections.

**device**

Devices managed by NetworkManager.

**agent**

NetworkManager secret agent or polkit agent.

**monitor**

Monitor NetworkManager changes.

The object name is the main argument to command `nmcli`. To show the overall connectivity status of the system, for example, the object `general` should be given as the argument:

```
$ nmcli general
STATE      CONNECTIVITY  WIFI-HW  WIFI      WWAN-HW  WWAN
connected  full          enabled  enabled   enabled  enabled
```





Column `STATE` tells whether the system is connected to a network or not. If the connection is limited due to external misconfiguration or access restrictions, then the `CONNECTIVITY` column will not report the `full` connectivity status. If `Portal` appears in the `CONNECTIVITY` column, it means that extra authentication steps (usually through the web browser) are required to complete the connection process. The remaining columns report the status of the wireless connections (if any), either `WIFI` or `WWAN` (Wide Wireless Area Network, i.e. cellular networks). The `HW` suffix indicates that the status corresponds to the network device rather than the system network connection, that is, it tells if the hardware is enabled or disabled to save power.

In addition to the object argument, `nmcli` also needs a command argument to execute. The `status` command is used by default if no command argument is present, so the command `nmcli general` is actually interpreted as `nmcli general status`.

It is hardly necessary to take any action when the network adapter is connected directly to the access point through cables, but wireless networks require further interaction to accept new members. `nmcli` facilitates the connection process and saves the settings to connect automatically in the future, hence it is very helpful for laptops or any other mobile appliances.

Before connecting to wi-fi, it is convenient to first list the available networks in the local area. If the system has a working wi-fi adapter, then the `device` object will use it to scan the available networks with command `nmcli device wifi list`:

```
$ nmcli device wifi list
IN-USE  BSSID          SSID      MODE  CHAN  RATE      SIGNAL  BARS  SECURITY
      90:F6:52:C5:FA:12  Hypnotoad  Infra  11    130 Mbit/s  67      ████  WPA2
```

10:72:23:C7:27:AC	Jumbao	Infra	1	130 Mbit/s	55		WPA2
00:1F:33:33:E9:BE	NETGEAR	Infra	1	54 Mbit/s	35		WPA1 WPA2
A4:33:D7:85:6D:B0	AP53	Infra	11	130 Mbit/s	32		WPA1 WPA2
98:1E:19:1D:CC:3A	Bruma	Infra	1	195 Mbit/s	22		WPA1 WPA2

Most users will probably use the name in the SSID column to identify the network of interest. For example, command `nmcli` can connect to the network named `Hypnotoad` using the device object again:

```
$ nmcli device wifi connect Hypnotoad
```

If the command is executed inside a terminal emulator in the graphical environment, then a dialog box will appear asking for the network's passphrase. When executed in a text only console, the password may be provided together with the other arguments:

```
$ nmcli device wifi connect Hypnotoad password MyPassword
```

If the wi-fi network hides its SSID name, `nmcli` can still connect to it with the extra hidden `yes` arguments:

```
$ nmcli device wifi connect Hypnotoad password MyPassword hidden yes
```

If the system has more than one wi-fi adapter, the one to be used may be indicated with `ifname`. For example, to connect using the adapter named `wlo1`:

```
$ nmcli device wifi connect Hypnotoad password MyPassword ifname wlo1
```

After the connection succeeds, NetworkManager will name it after the corresponding SSID (if it is a wi-fi connection) and will keep it for future connections. The connections names and their UUIDs are listed by command `nmcli connection show`:

```
$ nmcli connection show
NAME                UUID                                TYPE      DEVICE
Ethernet            53440255-567e-300d-9922-b28f0786f56e ethernet enp3s5
tun0                cae685e1-b0c4-405a-8ece-6d424e1fb5f8 tun       tun0
Hypnotoad           6fdec048-bcc5-490a-832b-da83d8cb7915 wifi      wlo1
4G                  a2cf4460-0cb7-42e3-8df3-ccb927f2fd88 gsm       --
```

The type of each connection is shown — which can be `ethernet`, `wifi`, `tun`, `gsm`, `bridge`, etc. — as well as the device to which they are associated with. To perform actions on a specific connection, its name or UUID must be supplied. To deactivate the `Hypnotoad` connection, for example:

```
$ nmcli connection down Hypnotoad
Connection 'Hypnotoad' successfully deactivated
```

Likewise, the command `nmcli connection up Hypnotoad` can be used to bring the connection up, as it is now saved by `NetworkManager`. The interface name can also be used to reconnect, but in this case the `device` object should be used instead:

```
$ nmcli device disconnect wlo2
Device 'wlo1' successfully disconnected.
```

The interface name can also be used to reestablish the connection:

```
$ nmcli device connect wlo2
Device 'wlo1' successfully activated with '833692de-377e-4f91-a3dc-d9a2b1fcf6cb'.
```

Note that the connection UUID changes every time the connection is brought up, so it is preferable to use its name for consistency.

If the wireless adapter is available but it is not being used, then it can be turned off to save power. This time, the object `radio` should be passed to `nmcli`:

```
$ nmcli radio wifi off
```

Of course, the wireless device can be turned on again with command `nmcli radio wifi on`.

Once the connections are established no manual interaction will be required in the future, as `NetworkManager` identifies available known networks and automatically connects to them. If necessary, `NetworkManager` has plugins that can extend its functionalities, like the plugin to support VPN connections.

## systemd-networkd

Systems running `systemd` can optionally use its built-in daemons to manage network connectivity: `systemd-networkd` to control network interfaces and `systemd-resolved` to manage the local name resolution. These services are backwards compatible with legacy Linux configuration

methods, but the configuration of network interfaces in particular has features that are worth knowing.

The configuration files used by `systemd-networkd` to setup network interfaces can be found in any of the following three directories:

### **`/lib/systemd/network`**

The system network directory.

### **`/run/systemd/network`**

The volatile runtime network directory.

### **`/etc/systemd/network`**

The local administration network directory.

The files are processed in lexicographic order, so it is recommended to start their names with numbers to make the ordering easier to read and set.

Files in `/etc` have the highest priority, whilst files in `/run` take precedence over files with the same name in `/lib`. This means that if configuration files in different directories have the same name, then `systemd-networkd` will ignore the files with lesser priority. Separating files like that is a way to change the interface settings without having to modify the original files: modifications can be placed in `/etc/systemd/network` to override those in `/lib/systemd/network`.

The purpose of each configuration file depends on its suffix. File names ending in `.netdev` are used by `systemd-networkd` to create virtual network devices, such as *bridge* or *tun* devices. Files ending in `.link` set low-level configurations for the corresponding network interface. `systemd-networkd` detects and configures network devices automatically as they appear — as well as ignore devices already configured by other means — so there is little need to add these files in most situations.

The most important suffix is `.network`. Files using this suffix can be used to setup network addresses and routes. As with the other configuration file types, the name of the file defines the order in which the file will be processed. The network interface to which the configuration file refers to is defined in the `[Match]` section inside the file.

For example, the ethernet network interface `enp3s5` can be selected within the file `/etc/systemd/network/30-lan.network` by using the `Name=enp3s5` entry in the `[Match]` section:

```
[Match]
```



```
Name=enp3s5
```

A list of whitespace-separated names is also accepted to match many network interfaces with this same file at once. The names can contain shell-style globs, like `en*`. Other entries provide various matching rules, like selecting a network device by its MAC address:

```
[Match]
MACAddress=00:16:3e:8d:2b:5b
```

The settings for the device are in the `[Network]` section of the file. A simple static network configuration only requires the `Address` and `Gateway` entries:

```
[Match]
MACAddress=00:16:3e:8d:2b:5b

[Network]
Address=192.168.0.100/24
Gateway=192.168.0.1
```

To use the DHCP protocol instead of static IP addresses, the `DHCP` entry should be used instead:

```
[Match]
MACAddress=00:16:3e:8d:2b:5b

[Network]
DHCP=yes
```

The `systemd-networkd` service will try to fetch both IPv4 and IPv6 addresses for the network interface. To use IPv4 only, `DHCP=ipv4` should be used. Likewise, `DHCP=ipv6` will ignore IPv4 settings and use the provided IPv6 address only.

Password-protected wireless networks can also be configured by `systemd-networkd`, but the network adapter must be already authenticated in the network before `systemd-networkd` can configure it. Authentication is performed by *WPA supplicant*, a program dedicated to configure network adapters for password protected networks.

The first step is to create the credentials file with command `wpa_passphrase`:

```
# wpa_passphrase MyWifi > /etc/wpa_supplicant/wpa_supplicant-wlo1.conf
```

This command will take the passphrase for the MyWifi wireless network from the standard input and store its hash in the `/etc/wpa_supplicant/wpa_supplicant-wlo1.conf`. Note that the filename should contain the appropriate name of the wireless interface, hence the `wlo1` in the file name.

The systemd manager reads the WPA passphrase files in `/etc/wpa_supplicant/` and creates the corresponding service to run WPA supplicant and bring the interface up. The passphrase file created in the example will then have a corresponding service unit called `wpa_supplicant@wlo1.service`. Command `systemctl start wpa_supplicant@wlo1.service` will associate the wireless adapter with the remote access point. Command `systemctl enable wpa_supplicant@wlo1.service` makes the association automatic during boot time.

Finally, a `.network` file matching the `wlo1` interface must be present in `/etc/systemd/network/`, as `systemd-networkd` will use it to configure the interface as soon as WPA supplicant finishes the association with the access point.

## Guided Exercises

1. What is the meaning of the word `Portal` in the `CONNECTIVITY` column in the output of command `nmcli general status`?

2. In a console terminal, how can an ordinary user use the command `nmcli` to connect to the `MyWifi` wireless network protected by the password `MyPassword`?

3. What command can turn the wireless adapter on if it was previously disabled by the operating system?

4. Custom configuration files should be placed in what directory when `systemd-networkd` is managing the network interfaces?

## Explorational Exercises

1. How can a user run the command `nmcli` to delete an unused connection named `Hotel Internet`?

2. NetworkManager scans wi-fi networks periodically and command `nmcli device wifi list` only lists the access points found in the last scan. How should the `nmcli` command be used to ask NetworkManager to immediately re-scan all available access points?

3. What name entry should be used in the `[Match]` section of a `systemd-networkd` configuration file to match all ethernet interfaces?

4. How should the `wpa_passphrase` command be executed to use the passphrase given as an argument and not from the standard input?

## Summary

This lesson covers the common tools used in Linux to manage heterogeneous and dynamic network connections. Although most configuration methods do not require user intervention, sometimes that is necessary and tools like *NetworkManager* and *systemd-networkd* can reduce the hassle to a minimum. The lesson goes through the following topics:

- How NetworkManager and systemd-networkd integrate with the system.
- How the user can interact with NetworkManager and systemd-networkd.
- Basic interface configuration with both NetworkManager and systemd-networkd.

The concepts, commands and procedures addressed were:

- NetworkManager's client commands: `nmtui` and `nmcli`.
- Scanning and connecting to wireless networks using `nmcli` appropriate commands.
- Persistent wi-fi network connections using `systemd-networkd`.

## Answers to Guided Exercises

1. What is the meaning of the word `Portal` in the `CONNECTIVITY` column in the output of command `nmcli general status`?

It means that extra authentication steps (usually through the web browser) are required to complete the connection process.

2. In a console terminal, how can an ordinary user use the command `nmcli` to connect to the `MyWifi` wireless network protected by the password `MyPassword`?

In a text-only terminal, the command would be

```
$ nmcli device wifi connect MyWifi password MyPassword
```

3. What command can turn the wireless adapter on if it was previously disabled by the operating system?

```
$ nmcli radio wifi on
```

4. Custom configuration files should be placed in what directory when `systemd-networkd` is managing the network interfaces?

In the local administration network directory: `/etc/systemd/network`.

## Answers to Explorational Exercises

1. How can a user run the command `nmcli` to delete an unused connection named `Hotel Internet`?

```
$ nmcli connection delete "Hotel Internet"
```

2. NetworkManager scans wi-fi networks periodically and command `nmcli device wifi list` only lists the access points found in the last scan. How should the `nmcli` command be used to ask NetworkManager to immediately re-scan all available access points?

The root user can run `nmcli device wifi rescan` to make NetworkManager re-scan available access points.

3. What name entry should be used in the `[Match]` section of a `systemd-networkd` configuration file to match all ethernet interfaces?

The entry `name=en*`, as `en` is the prefix for ethernet interfaces in Linux and `systemd-networkd` accepts shell-like globs.

4. How should the `wpa_passphrase` command be executed to use the passphrase given as an argument and not from the standard input?

The password should be given just after the SSID, as in `wpa_passphrase MyWifi MyPassword`.



## 109.3 Basic network troubleshooting

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 109.3](#)

### Weight

4

### Key knowledge areas

- Manually configure network interfaces, including viewing and changing the configuration of network interfaces using `iproute2`.
- Manually configure routing, including viewing and changing routing tables and setting the default route using `iproute2`.
- Debug problems associated with the network configuration.
- Awareness of legacy net-tools commands.

### Partial list of the used files, terms and utilities

- `ip`
- `hostname`
- `ss`
- `ping`
- `ping6`
- `traceroute`
- `traceroute6`
- `tracpath`
- `tracpath6`



- netcat
- ifconfig
- netstat
- route



## 109.3 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	109 Networking Fundamentals
<b>Objective:</b>	109.3 Basic network troubleshooting
<b>Lesson:</b>	1 of 2

### Introduction

Linux has very flexible and powerful network capabilities. In fact, Linux based operating systems are often used on common network devices, including expensive commercial equipment. Linux networking could be a certification in itself. With this mind, this lesson is only going to cover a few basic configuration and troubleshooting tools.

Be sure to review the lessons on internet protocols and persistent network configuration prior to this lesson. Within this lesson, we will be covering tools to configure and troubleshoot IPv4 and IPv6 networking.

While not an official objective, *packet sniffers* such as `tcpdump` are useful troubleshooting tools. Packet sniffers allow you to view and record packets coming into or out of a network interface. Tools such as *hex viewers* and *protocol analyzers* can be used to view these packets in more detail than a packet sniffer will typically allow. It wouldn't hurt to at least be aware of such programs.

### About the `ip` Command

The `ip` command is a fairly recent utility used to view and configure just about anything relating

to network configurations. This lesson covers some of the most used subcommands of `ip`, but it barely scratches the surface of what is available. Learning to read the documentation will help you be much more efficient with it.

Each subcommand of `ip` has its own man page. The `SEE ALSO` section of the `ip` man page has a list of them:

```
$ man ip
...
SEE ALSO
    ip-address(8), ip-addrlabel(8), ip-l2tp(8), ip-link(8), ip-maddress(8),
    ip-monitor(8), ip-mroute(8), ip-neighbour(8), ip-netns(8), ip-
    ntable(8), ip-route(8), ip-rule(8), ip-tcp_metrics(8), ip-token(8), ip-
    tunnel(8), ip-xfrm(8)
    IP Command reference ip-cref.ps
...
```

Instead of looking at this every time you need the man page, simply add `-` and the name of the subcommand to `ip`, e.g. `man ip-route`.

Another source of information is the help function. To view the built-in help, add `help` after the subcommand:

```
$ ip address help
Usage: ip address {add|change|replace} IFADDR dev IFNAME [ LIFETIME ]
                                     [ CONFFLAG-LIST ]

    ip address del IFADDR dev IFNAME [mngtmpaddr]
    ip address {save|flush} [ dev IFNAME ] [ scope SCOPE-ID ]
                                     [ to PREFIX ] [ FLAG-LIST ] [ label LABEL ] [up]
    ip address [ show [ dev IFNAME ] [ scope SCOPE-ID ] [ master DEVICE ]
                                     [ type TYPE ] [ to PREFIX ] [ FLAG-LIST ]
                                     [ label LABEL ] [up] [ vrf NAME ] ]
    ip address {showdump|restore}
IFADDR := PREFIX | ADDR peer PREFIX
...
```

## Netmask and Routing Review

IPv4 and IPv6 are what are known as routed or routable protocols. This means they are designed in a way that make it possible for network designers to control traffic flow. Ethernet is not a routable protocol. This means that if you were to connect a bunch of devices together using

nothing but Ethernet, there is very little you can do to control the flow of network traffic. Any measures to control traffic would end up similar to current routable and routing protocols.

Routable protocols allow network designers to segment networks to reduce the processing requirements of connectivity devices, provide redundancy, and manage traffic.

IPv4 and IPv6 addresses have two sections. The first set of bits make up the network section while the second set make up the host portion. The number of bits that make up the network portion are determined by the *netmask* (also called *subnet mask*). Sometimes it will also be referred to as the *prefix length*. Regardless of what it is called, it is the number of bits that the machine treats as the network portion of the address. With IPv4, sometimes this is specified in dotted decimal notation.

Below is an example using IPv4. Notice how the binary digits maintain their place value in the octets even when it is divided by the netmask.

```
192.168.130.5/20

      192      168      130      5
      11000000 10101000 10000010 00000101

20 bits = 11111111 11111111 11110000 00000000

Network = 192.168.128.0
Host    = 2.5
```

The network portion of an address is used by an IPv4 or IPv6 machines to lookup which interface a packet should be sent out on in its routing table. When an IPv4 or IPv6 host with routing enabled receives a packet that is not for the host itself, it attempts to match the network portion of the destination to a network in the routing table. If a matching entry is found, it sends the packet to the destination specified in the routing table. If no entries are found and a default route is configured, it is sent to the default route. If no entry is found and no default route are configured, the packet is discarded.

## Configuring an Interface

There are two tools we will be covering that you can use to configure a network interface: `ifconfig` and `ip`. The `ifconfig` program, while still widely used, is considered a legacy tool and may not be available on newer systems.

### TIP

On newer Linux distributions, installation of the `net-tools` package will provide you with the legacy networking commands.

Before configuring an interface, you must first know what interfaces are available. There are a few ways to do this. One way is to use the `-a` option of `ifconfig`:

```
$ ifconfig -a
```

Another way is with `ip`. Sometimes you will see examples with `ip addr`, `ip a`, and some with `ip address`. They are synonymous. Officially, the subcommand is `ip address`. This means that if you wish to view the man page, you must use `man ip-address` and not `man ip-addr`.

The `link` subcommand for `ip` will list the interface links available for configuration:

```
$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT
group default qlen 1000
    link/ether 08:00:27:54:18:57 brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT
group default qlen 1000
    link/ether 08:00:27:ab:11:3e brd ff:ff:ff:ff:ff:ff
```

Assuming the `sys` filesystem is mounted, you can also list the contents of `/sys/class/net`:

```
$ ls /sys/class/net
enp0s3 enp0s8 lo
```

To configure an interface with `ifconfig`, you must be logged in as root or use a utility such as `sudo` to run the command with root privilege. Follow the example below:

```
# ifconfig enp1s0 192.168.50.50/24
```

The Linux version of `ifconfig` is flexible with how you specify the subnet mask:

```
# ifconfig eth2 192.168.50.50 netmask 255.255.255.0
# ifconfig eth2 192.168.50.50 netmask 0xffffffff00
# ifconfig enp0s8 add 2001:db8::10/64
```

Notice how with IPv6 the keyword `add` was used. If you don't precede an IPv6 address with `add`,

you will get an error message.

The following command configures an interface with `ip`:

```
# ip addr add 192.168.5.5/24 dev enp0s8
# ip addr add 2001:db8::10/64 dev enp0s8
```

With `ip`, the same command is used for both IPv4 and IPv6.

## Configuring Low Level Options

The `ip link` command is used to configure low level interface or protocol settings such as VLANs, ARP, or MTUs, or disabling an interface.

A common task for `ip link` is to disable or enable an interface. This can be done with `ifconfig` as well:

```
# ip link set dev enp0s8 down
# ip link show dev enp0s8
3: enp0s8: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN mode DEFAULT group
default qlen 1000
    link/ether 08:00:27:ab:11:3e brd ff:ff:ff:ff:ff:ff
# ifconfig enp0s8 up
# ip link show dev enp0s8
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT
group default qlen 1000
    link/ether 08:00:27:ab:11:3e brd ff:ff:ff:ff:ff:ff
```

Sometimes you may need to adjust an interface's MTU. As with enabling/disabling interfaces, this can be done with either `ifconfig` or `ip link`:

```
# ip link set enp0s3 mtu 2000
# ip link show dev enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 2000 qdisc pfifo_fast state UP mode DEFAULT
group default qlen 1000
    link/ether 08:00:27:54:53:59 brd ff:ff:ff:ff:ff:ff
# ifconfig enp0s3 mtu 1500
# ip link show dev enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT
group default qlen 1000
    link/ether 08:00:27:54:53:59 brd ff:ff:ff:ff:ff:ff
```

## The Routing Table

The commands `route`, `netstat -r`, and `ip route` can all be used to view your routing table. If you wish to modify your routes, you need to use `route` or `ip route`. Below are examples of viewing a routing table:

```
$ netstat -r
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window  irtt Iface
default          10.0.2.2        0.0.0.0         UG      0 0        0 enp0s3
10.0.2.0         0.0.0.0         255.255.255.0   U       0 0        0 enp0s3
192.168.150.0   0.0.0.0         255.255.255.0   U       0 0        0 enp0s8

$ ip route
default via 10.0.2.2 dev enp0s3 proto dhcp metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100
192.168.150.0/24 dev enp0s8 proto kernel scope link src 192.168.150.200

$ route
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
default          10.0.2.2        0.0.0.0         UG    100    0      0 enp0s3
10.0.2.0         0.0.0.0         255.255.255.0   U     100    0      0 enp0s3
192.168.150.0   0.0.0.0         255.255.255.0   U      0      0      0 enp0s8
```

Notice how there is no output regarding IPv6. If you wish to view your routing table for IPv6, you must use `route -6`, `netstat -6r`, and `ip -6 route`.

```
$ route -6
Kernel IPv6 routing table
Destination      Next Hop          Flag Met Ref Use If
2001:db8::/64    [::]              U    256 0    0 enp0s8
fe80::/64        [::]              U    100 0    0 enp0s3
2002:a00::/24    [::]              !n   1024 0    0 lo
[::]/0           2001:db8::1      UG   1    0    0 enp0s8
localhost/128    [::]              Un   0    2    84 lo
2001:db8::10/128 [::]              Un   0    1    0 lo
fe80::a00:27ff:fe54:5359/128 [::]              Un   0    1    0 lo
ff00::/8         [::]              U    256 1    3 enp0s3
ff00::/8         [::]              U    256 1    6 enp0s8
```

An example of `netstat -r6` has been omitted because its output is identical to `route -6`. Some of the output of the above `route` command is self explanatory. The `Flag` column provides some

information about the route. The `U` flag indicates that a route is up. A `!` means reject route i.e. a route with a `!` won't be used. The `n` flag means the route hasn't been cached. The kernel maintains a cache of routes for faster lookups separately from all known routes. The `G` flag indicates a gateway. The `Metric` or `Met` column isn't used by the kernel. It refers to the administrative distance to the target. This administrative distance is used by routing protocols to determine dynamic routes. The `Ref` column is the reference count, or number of uses of a route. Like `Metric`, it is not used by the Linux kernel. The `Use` column shows the number of lookups for a route.

In the output of `netstat -r`, `MSS` indicates the maximum segment size for TCP connections over that route. The `Window` column shows you the default TCP window size. The `irrtt` shows the round trip time for packets on this route.

The output of `ip route` and `ip -6 route` reads as follows:

1. Destination.
2. Optional address followed by interface.
3. The routing protocol used to add the route.
4. The scope of the route. If this is omitted, it is global scope, or a gateway.
5. The route's metric. This is used by dynamic routing protocols to determine the cost of the route. This isn't used by most systems.
6. If it is an IPv6 route, the RFC4191 route preference.

Working through a few examples should clarify this:

### IPv4 Example

```
default via 10.0.2.2 dev enp0s3 proto dhcp metric 100
```

1. The destination is the default route.
2. The gateway address is `10.0.2.2` reachable through interface `enp0s3`.
3. It was added to the routing table by DHCP.
4. The scope was omitted, so it is global.
5. The route has a cost value of `100`.
6. No IPv6 route preference.

### IPv6 Example



```
fc0::/64 dev enp0s8 proto kernel metric 256 pref medium
```

1. The destination is `fc0::/64`.
2. It is reachable through interface `enp0s8`.
3. It was added automatically by the kernel.
4. The scope was omitted, so it is global.
5. The route has a cost value of 256.
6. It has an IPv6 preference of `medium`.

## Managing Routes

Routes can be managed by using `route` or `ip route`. Below is an example of adding and removing a route using the `route` command. With `route`, you must use the `-6` option for IPv6:

```
# ping6 -c 2 2001:db8:1::20
connect: Network is unreachable
# route -6 add 2001:db8:1::/64 gw 2001:db8::3
# ping6 -c 2 2001:db8:1::20
PING 2001:db8:1::20(2001:db8:1::20) 56 data bytes
64 bytes from 2001:db8:1::20: icmp_seq=1 ttl=64 time=0.451 ms
64 bytes from 2001:db8:1::20: icmp_seq=2 ttl=64 time=0.438 ms
# route -6 del 2001:db8:1::/64 gw 2001:db8::3
# ping6 -c 2 2001:db8:1::20
connect: Network is unreachable
```

Below is the same example using the `ip route` command:

```
# ping6 -c 2 2001:db8:1:20
connect: Network is unreachable
# ip route add 2001:db8:1::/64 via 2001:db8::3
# ping6 -c 2 2001:db8:1:20
PING 2001:db8:1::20(2001:db8:1::20) 56 data bytes
64 bytes from 2001:db8:1::20: icmp_seq=2 ttl=64 time=0.529 ms
64 bytes from 2001:db8:1::20: icmp_seq=2 ttl=64 time=0.438 ms
# ip route del 2001:db8:1::/64 via 2001:db8::3
# ping6 -c 2 2001:db8:1::20
connect: Network is unreachable
```

## Guided Exercises

1. Which commands can be used to list network interfaces?

2. How would you temporarily disable an interface? How would you re-enable it?

3. Which of the following is a reasonable subnet mask for IPv4?

0.0.0.255	
255.0.255.0	
255.252.0.0	
/24	

4. Which commands can you use to verify your default route?

5. How would add a second IP address to an interface?

## Explorational Exercises

1. Which subcommand of `ip` can be used to configure vlan tagging?

2. How would you configure a default route?

3. How would you get detailed information about the `ip neighbour` command? What happens if you run it by itself?

4. How would you backup your routing table? How would you restore from it?

5. Which `ip` subcommand can be used to configure spanning tree options?

## Summary

Networking is usually configured by a system's startup scripts or a helper such as NetworkManager. Most distributions have tools that will edit the startup script configuration files for you. Consult your distribution's documentation for details.

Being able to manually configure networking allows you to troubleshoot more effectively. It is useful in minimal environments used for things like restoring from backups or migrating to new hardware.

The utilities covered in this section have more functionality than covered in this lesson. It would be worthwhile to skim through the man page of each to familiarize yourself with the options available. The `ss` and `ip` commands are the modern way of doing things, while the rest that are covered, while still in common use, are considered legacy tools.

The best way to get familiar with the tools covered is practice. Using a computer with a modest amount of RAM, it is possible to setup a virtual network lab using virtual machines that you can practice with. Three virtual machines are enough to get comfortable with the tools listed.

Commands used in this lesson include:

### **ifconfig**

Legacy utility used to configure network interfaces and review their states.

### **ip**

Modern and versatile utility used to configure network interfaces and review their states.

### **netstat**

Legacy command used to view current network connections and route information.

### **route**

Legacy command used to view or modify a system's routing table.

## Answers to Guided Exercises

1. Which commands can be used to list network interfaces?

Any of the commands below:

```
ip link, ifconfig -a, or ls /sys/class/net
```

2. How would you temporarily disable an interface? How would you re-enable it?

You could use `ifconfig` or `ip link`:

Using `ifconfig`:

```
$ ifconfig wlan1 down
$ ifconfig wlan1 up
```

Using `ip link`:

```
$ ip link set wlan1 down
$ ip link set wlan1 up
```

3. Which of the following is a reasonable subnet mask for IPv4?

- 255.252.0.0
- /24

The other masks listed are invalid because they don't separate the address cleanly into two sections, the first part defining the network, and the second the host. The left most bits of a mask will always be 1 and the right bits will always be 0.

4. Which commands can you use to verify your default route?

You can use `route`, `netstat -r`, or `ip route`:

```
$ route
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
default        server          0.0.0.0         UG    600   0      0 wlan1
192.168.1.0    0.0.0.0        255.255.255.0  U    600   0      0 wlan1
$ netstat -r
```

```
Kernel IP routing table
Destination      Gateway          Genmask          Flags   MSS Window  irtt Iface
default          server           0.0.0.0          UG      0 0        0 wlan1
192.168.1.0      0.0.0.0         255.255.255.0   U       0 0        0 wlan1
$ ip route
default via 192.168.1.20 dev wlan1 proto static metric 600
192.168.1.0/24 dev wlan1 proto kernel scope link src 192.168.1.24 metric 600
```

## 5. How would add a second IP address to an interface?

You would use `ip address` or `ifconfig`. Keep in mind that `ifconfig` is a legacy tool:

```
$ ip addr add 172.16.15.16/16 dev enp0s9 label enp0s9:sub1
```

The portion of the command `label enp0s9:sub1` adds an alias to `enp0s9`. If you don't use the legacy `ifconfig` you can omit this. If you do, the command will still work, but the address you just added won't show up in the output of `ifconfig`.

You can also use `ifconfig`:

```
$ ifconfig enp0s9:sub1 172.16.15.16
```

## Answers to Explorational Exercises

1. Which subcommand of `ip` can be used to configure vlan tagging?

`ip link` has a `vlan` option that can be used. Below is an example of tagging a sub interface with vlan 20.

```
# ip link add link enp0s9 name enp0s9.20 type vlan id 20
```

2. How would you configure a default route?

Using `route` or `ip route`:

```
# route add default gw 192.168.1.1
# ip route add default via 192.168.1.1
```

3. How would you get detailed information about the `ip neighbour` command? What happens if you run it by itself?

By reading the man page:

```
$ man ip-neighbour
```

It displays your ARP cache:

```
$ ip neighbour
10.0.2.2 dev enp0s3 lladdr 52:54:00:12:35:02 REACHABLE
```

4. How would you backup your routing table? How would you restore from it?

The example below demonstrates backing up and restoring a routing table:

```
# ip route save > /root/routes/route_backup
# ip route restore < /root/routes/route_backup
```

5. Which `ip` subcommand can be used to configure spanning tree options?

Similar to managing vlan settings, `ip link` can configure spanning tree by using the `bridge` type. The example shows adding a virtual interface with a STP priority of 50:

```
# ip link add link enp0s9 name enp0s9.50 type bridge priority 50
```





## 109.3 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	109 Networking Fundamentals
<b>Objective:</b>	109.3 Basic network troubleshooting
<b>Lesson:</b>	2 of 2

### Introduction

Linux based operating systems have a variety of tools to troubleshoot network problems with. This lesson is going to cover some of the more common ones. At this point you should have a grasp of the OSI or other layered models of networking, IPv4 or IPv6 addressing, and the basics of routing and switching.

The best way to test a network connection is to try to use your application. When that doesn't work, there are plenty of tools available to help diagnose the problem.

### Testing Connections With ping

The `ping` and `ping6` commands can be used to send an ICMP echo request to an IPv4 or IPv6 address, respectively. An ICMP echo request sends a small amount of data to the destination address. If the destination address is reachable, it will send an ICMP echo reply message back to the sender with the same data that was sent to it:

```
$ ping -c 3 192.168.50.2
PING 192.168.50.2 (192.168.50.2) 56(84) bytes of data.
```

```
64 bytes from 192.168.50.2: icmp_seq=1 ttl=64 time=0.525 ms
64 bytes from 192.168.50.2: icmp_seq=2 ttl=64 time=0.419 ms
64 bytes from 192.168.50.2: icmp_seq=3 ttl=64 time=0.449 ms

--- 192.168.50.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 0.419/0.464/0.525/0.047 ms
```

```
$ ping6 -c 3 2001:db8::10
PING 2001:db8::10(2001:db8::10) 56 data bytes
64 bytes from 2001:db8::10: icmp_seq=1 ttl=64 time=0.425 ms
64 bytes from 2001:db8::10: icmp_seq=2 ttl=64 time=0.480 ms
64 bytes from 2001:db8::10: icmp_seq=3 ttl=64 time=0.725 ms

--- 2001:db8::10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.425/0.543/0.725/0.131 ms
```

The `-c` option is used to specify the number of packets to send. If you omit this option, `ping` and `ping6` will continue to send packets until you stop it, typically with the `Ctrl + C` keyboard combination.

Just because you can't ping a host, doesn't mean you can't connect to it. Many organizations have firewalls or router access control lists that block everything but the bare minimum needed for their systems to function. This includes ICMP echo request and replies. Since these packets can include arbitrary data, a clever attacker could use them to exfiltrate data.

## Tracing Routes

The `traceroute` and `traceroute6` programs can be used to show you the route a packet takes to get to its destination. They do this by sending multiple packets to the destination, incrementing the *Time-To-Live* (TTL) field of the IP header with each subsequent packet. Each router along the way will respond with a TTL exceeded ICMP message:

```
$ traceroute 192.168.1.20
traceroute to 192.168.1.20 (192.168.1.20), 30 hops max, 60 byte packets
 1  10.0.2.2 (10.0.2.2)  0.396 ms  0.171 ms  0.132 ms
 2  192.168.1.20 (192.168.1.20)  2.665 ms  2.573 ms  2.573 ms
$ traceroute 192.168.50.2
traceroute to 192.168.50.2 (192.168.50.2), 30 hops max, 60 byte packets
 1  192.168.50.2 (192.168.50.2)  0.433 ms  0.273 ms  0.171 ms
```

```

$ traceroute6 2001:db8::11
traceroute to 2001:db8::11 (2001:db8::11), 30 hops max, 80 byte packets
 1 2001:db8::11 (2001:db8::11) 0.716 ms 0.550 ms 0.641 ms
$ traceroute 2001:db8::11
traceroute to 2001:db8::11 (2001:db8::11), 30 hops max, 80 byte packets
 1 2001:db8::10 (2001:db8::11) 0.617 ms 0.461 ms 0.387 ms
$ traceroute net2.example.net
traceroute to net2.example.net (192.168.50.2), 30 hops max, 60 byte packets
 1 net2.example.net (192.168.50.2) 0.533 ms 0.529 ms 0.504 ms
$ traceroute6 net2.example.net
traceroute to net2.example.net (2001:db8::11), 30 hops max, 80 byte packets
 1 net2.example.net (2001:db8::11) 0.738 ms 0.607 ms 0.304 ms

```

By default, `traceroute` sends 3 UDP packets with junk data to port 33434, incrementing it each time it sends a packet. Each line in the command's output is a router interface the packet traverses through. The times shown in each line of the output is the round trip time for each packet. The IP address is the address of the router interface in question. If `traceroute` is able to, it uses the DNS name of the router interface. Sometimes you will see `*` in place of a time. When this happens, it means that `traceroute` never received the TTL exceeded message for this packet. When you start seeing this, this often indicates that the last response is the last hop on the route.

If you have access to `root`, the `-I` option will set `traceroute` to use ICMP echo requests instead of UDP packets. This is often more effective than UDP because the destination host is more likely to respond to an ICMP echo request than the UDP packet:

```

# traceroute -I learning.lpi.org
traceroute to learning.lpi.org (208.94.166.201), 30 hops max, 60 byte packets
 1 047-132-144-001.res.spectrum.com (47.132.144.1) 9.764 ms 9.702 ms 9.693 ms
 2 096-034-094-106.biz.spectrum.com (96.34.94.106) 8.389 ms 8.481 ms 8.480 ms
 3 dtr01h1rgnc-gbe-4-15.h1rg.nc.charter.com (96.34.64.172) 8.763 ms 8.775 ms 8.770 ms
 4 acr01mgtnnc-vln-492.mgtn.nc.charter.com (96.34.67.202) 27.080 ms 27.154 ms 27.151 ms
 5 bbr01gnvlsc-bue-3.gnvl.sc.charter.com (96.34.2.112) 31.339 ms 31.398 ms 31.395 ms
 6 bbr01aldlmi-tge-0-0-0-13.aldl.mi.charter.com (96.34.0.161) 39.092 ms 38.794 ms 38.821
ms
 7 prr01ashbva-bue-3.ashb.va.charter.com (96.34.3.51) 34.208 ms 36.474 ms 36.544 ms
 8 bx2-ashburn.bell.ca (206.126.236.203) 53.973 ms 35.975 ms 38.250 ms
 9 tcore4-ashburnbk_0-12-0-0.net.bell.ca (64.230.125.190) 66.315 ms 65.319 ms 65.345 ms
10 tcore4-toronto47_2-8-0-3.net.bell.ca (64.230.51.22) 67.427 ms 67.502 ms 67.498 ms
11 agg1-toronto47_xe-7-0-0_core.net.bell.ca (64.230.161.114) 61.270 ms 61.299 ms 61.291
ms
12 dis4-clarkson16_5-0.net.bell.ca (64.230.131.98) 61.101 ms 61.177 ms 61.168 ms
13 207.35.12.142 (207.35.12.142) 70.009 ms 70.069 ms 59.893 ms

```

```

14 unassigned-117.001.centrilogic.com (66.135.117.1) 61.778 ms 61.950 ms 63.041 ms
15 unassigned-116.122.akn.ca (66.135.116.122) 62.702 ms 62.759 ms 62.755 ms
16 208.94.166.201 (208.94.166.201) 62.936 ms 62.932 ms 62.921 ms

```

Some organizations block ICMP echo requests and replies. To get around this, you can use TCP. By using a known open TCP port, you can guarantee the destination host will respond. To use TCP, use the `-T` option along with `-p` to specify the port. As with ICMP echo requests, you must have access to `root` to do this:

```

# traceroute -m 60 -T -p 80 learning.lpi.org
traceroute to learning.lpi.org (208.94.166.201), 60 hops max, 60 byte packets
 1 * * *
 2 096-034-094-106.biz.spectrum.com (96.34.94.106) 12.178 ms 12.229 ms 12.175 ms
 3 dtr01h1rgnc-gbe-4-15.h1rg.nc.charter.com (96.34.64.172) 12.134 ms 12.093 ms 12.062 ms
 4 acr01mgtnnc-vln-492.mgtn.nc.charter.com (96.34.67.202) 31.146 ms 31.192 ms 31.828 ms
 5 bbr01gnvlsc-bue-3.gnvl.sc.charter.com (96.34.2.112) 39.057 ms 46.706 ms 39.745 ms
 6 bbr01aldlmi-tge-0-0-0-13.aldl.mi.charter.com (96.34.0.161) 50.590 ms 58.852 ms 58.841
ms
 7 prr01ashbva-bue-3.ashb.va.charter.com (96.34.3.51) 34.556 ms 37.892 ms 38.274 ms
 8 bx2-ashburn.bell.ca (206.126.236.203) 38.249 ms 36.991 ms 36.270 ms
 9 tcore4-ashburnbk_0-12-0-0.net.bell.ca (64.230.125.190) 66.779 ms 63.218 ms tcore3-
ashburnbk_100ge0-12-0-0.net.bell.ca (64.230.125.188) 60.441 ms
10 tcore4-toronto47_2-8-0-3.net.bell.ca (64.230.51.22) 63.932 ms 63.733 ms 68.847 ms
11 agg2-toronto47_xe-7-0-0_core.net.bell.ca (64.230.161.118) 60.144 ms 60.443 ms agg1-
toronto47_xe-7-0-0_core.net.bell.ca (64.230.161.114) 60.851 ms
12 dis4-clarkson16_5-0.net.bell.ca (64.230.131.98) 67.246 ms dis4-clarkson16_7-
0.net.bell.ca (64.230.131.102) 68.404 ms dis4-clarkson16_5-0.net.bell.ca (64.230.131.98)
67.403 ms
13 207.35.12.142 (207.35.12.142) 66.138 ms 60.608 ms 64.656 ms
14 unassigned-117.001.centrilogic.com (66.135.117.1) 70.690 ms 62.190 ms 61.787 ms
15 unassigned-116.122.akn.ca (66.135.116.122) 62.692 ms 69.470 ms 68.815 ms
16 208.94.166.201 (208.94.166.201) 61.433 ms 65.421 ms 65.247 ms
17 208.94.166.201 (208.94.166.201) 64.023 ms 62.181 ms 61.899 ms

```

Like `ping`, `traceroute` has its limitations. It is possible for firewalls and routers to block the packets sent from or returned to `traceroute`. If you have `root` access, there are options that can help you get accurate results.

## Finding MTUs With `tracepath`

The `tracepath` command is similar to `traceroute`. The difference is it tracks *Maximum*

*Transmission Unit* (MTU) sizes along the path. The MTU is either a configured setting on a network interface or hardware limitation of the largest protocol data unit that it can transmit or receive. The `tracepath` program works the same way as `traceroute` in that it increments the TTL with each packet. It differs by sending a very large UDP datagram. It is almost inevitable for the datagram to be larger than the device with the smallest MTU along the route. When the packet reaches this device, the device will typically respond with a destination unreachable packet. The ICMP destination unreachable packet has a field for the MTU of the link it would send the packet on if it were able. `tracepath` then sends all subsequent packets with this size:

```
$ tracepath 192.168.1.20
1?: [LOCALHOST]                pmtu 1500
1:  10.0.2.2                    0.321ms
1:  10.0.2.2                    0.110ms
2:  192.168.1.20                2.714ms reached
Resume: pmtu 1500 hops 2 back 64
```

Unlike `traceroute`, you must explicitly use `tracepath6` for IPv6:

```
$ tracepath 2001:db8::11
tracepath: 2001:db8::11: Address family for hostname not supported
$ tracepath6 2001:db8::11
1?: [LOCALHOST]                0.027ms pmtu 1500
1:  net2.example.net            0.917ms reached
1:  net2.example.net            0.527ms reached
Resume: pmtu 1500 hops 1 back 1
```

The output is similar to `traceroute`. The advantage of `tracepath` is on the last line it outputs the smallest MTU on the entire link. This can be useful for troubleshooting connections that can't handle fragments.

As with the previous troubleshooting tools, there is the potential for equipment to block your packets.

## Creating Arbitrary Connections

The `nc` program, known as netcat, can send or receive arbitrary data over a TCP or UDP network connection. The following examples should make its functionality clear.

Here is an example of setting up a listener on port 1234:

```
$ nc -l 1234
LPI Example
```

The output of `LPI Example` appears after the example below, which is setting up a netcat sender to send packets to `net2.example.net` on port `1234`. The `-l` option is used to specify that you wish for `nc` to receive data instead of send it:

```
$ nc net2.example.net 1234
LPI Example
```

Press `Ctrl` + `C` on either system to stop the connection.

Netcat works with both IPv4 and IPv6 addresses. It works with both TCP and UDP. It can even be used to setup a crude remote shell.

#### WARNING

Note that not every installation of `nc` supports the `-e` switch. Be sure to review the man pages for your installation for security information about this option as well as alternative methods to execute commands on a remote system.

```
$ hostname
net2
$ nc -u -e /bin/bash -l 1234
```

The `-u` option is for UDP. `-e` instructs netcat to send everything it receives to standard input of the executable following it. In this example, `/bin/bash`.

```
$ hostname
net1
$ nc -u net2.example.net 1234
hostname
net2
pwd
/home/emma
```

Notice how the `hostname` command output matched that of the listening host and the `pwd` command output a directory?

## Viewing Current Connections and Listeners

The `netstat` and `ss` programs can be used to view the status of your current listeners and connections. As with `ifconfig`, `netstat` is a legacy tool. Both `netstat` and `ss` have similar output and options. Here some options available to both programs:

**-a**

Shows all sockets.

**-l**

Shows listening sockets.

**-p**

Shows the process associated with the connection.

**-n**

Prevents name lookups for both ports and addresses.

**-t**

Shows TCP connections.

**-u**

Shows UDP connections.

The examples below show the output of a commonly used set of options for both programs:

```
# netstat -tulnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22             0.0.0.0:*               LISTEN      892/sshd
tcp        0      0 127.0.0.1:25           0.0.0.0:*               LISTEN      1141/master
tcp6       0      0 :::22                  :::*                     LISTEN      892/sshd
tcp6       0      0 :::1:25                 :::*                     LISTEN      1141/master
udp        0      0 0.0.0.0:68             0.0.0.0:*               *          692/dhclient

# ss -tulnp
# ss -tulnp
Netid State      Recv-Q Send-Q   Local Address:Port      Peer Address:Port
udp  UNCONN    0      0           *:68                    *:*
users: (("dhclient",pid=693,fd=6))
tcp  LISTEN    0      128          *:22                    *:*
```

```
users: ("sshd",pid=892,fd=3)
tcp  LISTEN    0      100      127.0.0.1:25      :
users: ("master",pid=1099,fd=13)
tcp  LISTEN    0      128      [::]:22          [::]:*
users: ("sshd",pid=892,fd=4)
tcp  LISTEN    0      100      [::1]:25         [::]:*
users: ("master",pid=1099,fd=14)
```

The `Recv-Q` column is the number of packets a socket has received but not passed off to its program. The `Send-Q` column is the number of packets a socket has sent that have not been acknowledged by the receiver. The rest of the columns are self explanatory.



## Guided Exercises

1. What command(s) would you use to send an ICMP echo to `learning.lpi.org`?

2. How could you determine the route to `8.8.8.8`?

3. What command would show you if any processes are listening on TCP port 80?

4. How could you find which process is listening on a port?

5. How could you determine the max MTU of a network path?

## Explorational Exercises

1. How could you use netcat to send an HTTP request to a web server?

2. What are a few reasons pinging a host can fail?

3. Name a tool you could use to see network packets reaching or leaving a Linux host?

4. How can you force traceroute to use a different interface?

5. Is it possible for traceroute to report MTUs?

## Summary

Networking is usually configured by a system's startup scripts or a helper such as NetworkManager. Most distributions have tools that will edit the startup script configuration files for you. Consult your distribution's documentation for details.

Being able to manually configure networking allows you to troubleshoot more effectively. It is useful in minimal environments used for things like restoring from backups or migrating to new hardware.

The utilities covered in this section have more functionality than covered in this lesson. It would be worthwhile to skim through the man page of each to familiarize yourself with the options available. The `ss` and `ip` commands are the modern way of doing things, while the rest that are covered, while still in common use, are considered legacy tools.

The best way to get familiar with the tools covered is practice. Using a computer with a modest amount of RAM, it is possible to setup a virtual network lab using virtual machines that you can practice with. Three virtual machines are enough to get comfortable with the tools listed.

The commands covered in this lesson include:

### **ping and ping6**

Used to transmit ICMP packets to a remote host to test a network connection's availability.

### **traceroute and traceroute6**

Used to trace a path through a network to determine a network's connectivity.

### **tracepath and tracepath6**

Used to trace a path through a network as well as determine MTU sizes along a route.

### **nc**

Used to set up arbitrary connections on a network for testing connectivity, as well as querying a network for available services and devices.

### **netstat**

Legacy command used to determine a system's open network connections and statistics.

### **ss**

Modern command used to determine a system's open network connections and statistics.

## Answers to Guided Exercises

1. What command(s) would you use to send an ICMP echo to `learning.lpi.org`?

You would use `ping` or `ping6`:

```
$ ping learning.lpi.org
```

OR

```
$ ping6 learning.lpi.org
```

2. How could you determine the route to `8.8.8.8`?

By using the `tracpath` or `traceroute` commands.

```
$ tracpath 8.8.8.8
```

OR

```
$ traceroute 8.8.8.8
```

3. What command would show you if any processes are listening on TCP port 80?

With `ss`:

```
$ ss -ln | grep ":80"
```

With `netstat`:

```
$ netstat -ln | grep ":80"
```

While not listed as a requirement for the exam, you can also use `lsof`:

```
# lsof -Pi:80
```

4. How could you find which process is listening on a port?

Again, there are multiple ways to do this. You could use `lsof` in the same manner as the previous answer, replacing the port number. You could also use `netstat` or `ss` with the `-p` option. Remember, `netstat` is considered a legacy tool.

```
# netstat -lnp | grep ":22"
```

The same options that work with `netstat` also work with `ss`:

```
# ss -lnp | grep ":22"
```

5. How could you determine the max MTU of a network path?

By using the `tracert` command:

```
$ tracert somehost.example.com
```

# Answers to Explorational Exercises

1. How could you use netcat to send an HTTP request to a web server?

By entering the HTTP request line, any headers, and a blank line into the terminal:

```
$ nc learning.lpi.org 80
GET /index.html HTTP/1.1
HOST: learning.lpi.org

HTTP/1.1 302 Found
Location: https://learning.lpi.org:443/index.html
Date: Wed, 27 May 2020 22:54:46 GMT
Content-Length: 5
Content-Type: text/plain; charset=utf-8

Found
```

2. What are a few reasons pinging a host can fail?

There are a number of possible reasons. Here are some:

- The remote host is down.
- A router ACL is blocking your ping.
- The remote host's firewall is blocking your ping.
- You may be using an incorrect host name or address.
- Your name resolution is returning an incorrect address.
- Your machine's network configuration is incorrect.
- Your machine's firewall is blocking it.
- The remote host's network configuration is incorrect.
- Your machine's interface(s) are disconnected.
- The remote machine's interface(s) are disconnected.
- A network component such as a switch, cable, or router between your machine and the remote's is no longer functioning.

3. Name a tool you could use to see network packets reaching or leaving a Linux host?

Both `tcpdump` and `wireshark` can be used.

#### 4. How can you force `traceroute` to use a different interface?

By using the `-i` option:

```
$ traceroute -i eth2 learning.lpi.org
traceroute -i eth2 learning.lpi.org
traceroute to learning.lpi.org (208.94.166.201), 30 hops max, 60 byte packets
...
```

#### 5. Is it possible for `traceroute` to report MTUs?

Yes, with the `--mtu` option:

```
# traceroute -I --mtu learning.lpi.org
traceroute to learning.lpi.org (208.94.166.201), 30 hops max, 65000 byte packets
 1  047-132-144-001.res.spectrum.com (47.132.144.1)  9.974 ms F=1500  10.476 ms  4.743 ms
 2  096-034-094-106.biz.spectrum.com (96.34.94.106)  8.697 ms  9.963 ms  10.321 ms
...
```



## 109.4 Configure client side DNS

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 109.4](#)

### Weight

2

### Key knowledge areas

- Query remote DNS servers.
- Configure local name resolution and use remote DNS servers.
- Modify the order in which name resolution is done.
- Debug errors related to name resolution.
- Awareness of `systemd-resolved`

### Partial list of the used files, terms and utilities

- `/etc/hosts`
- `/etc/resolv.conf`
- `/etc/nsswitch.conf`
- `host`
- `dig`
- `getent`





# 109.4 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	109 Networking Fundamentals
<b>Objective:</b>	109.4 Configure client side DNS
<b>Lesson:</b>	1 of 1

## Introduction

This lesson covers configuring client side name resolution and how to use a few CLI name resolution tools.

Remembering and maintaining IP addresses, UIDs, and GIDs, and other numbers for everything isn't feasible. Name resolution services translate easy to remember names to numbers and vice versa. This lesson focuses on host name resolution, but a similar process happens for user names, group names, port numbers, and several others.

## Name Resolution Process

Programs that resolve names to numbers almost always use functions provided by the standard C library, which on Linux systems is the GNU project's `glibc`. The first things these functions do is read the file `/etc/nsswitch.conf` for instructions on how to resolve that type of name. This lesson is focused on host name resolution, but the same process applies to other types of name resolution as well. Once the process reads `/etc/nsswitch.conf`, it looks up the name in the manner specified. Since `/etc/nsswitch.conf` supports plugins, what comes next could be anything. After the function is done looking up the name or number, it returns the result to the

calling process.

## DNS Classes

DNS has three record classes, IN, HS, and CH. In this lesson, all DNS queries will be of type IN. The IN class is for internet addresses using the TCP/IP stack. CH is for ChaosNet, which is a network technology that was short lived and is no longer in use. The HS class is for Hesiod. Hesiod is a way to store things like passwd and group entries in DNS. Hesiod is beyond the scope of this lesson.

## Understanding `/etc/nsswitch.conf`

The best way to learn about this file is to read the man page that is part of the Linux man-pages project. It is available on most systems. It can be accessed with the command `man nsswitch.conf`. Alternatively, it can be found at [https://man7.org/linux/man-pages/dir\\_section\\_5.html](https://man7.org/linux/man-pages/dir_section_5.html)

Below is a simple example of `/etc/nsswitch.conf` from its man page:

```
passwd:          compat
group:           compat
shadow:          compat

hosts:           dns [!UNAVAIL=return] files
networks:        nis [NOTFOUND=return] files
ethers:          nis [NOTFOUND=return] files
protocols:       nis [NOTFOUND=return] files
rpc:             nis [NOTFOUND=return] files
services:        nis [NOTFOUND=return] files
# This is a comment. It is ignored by the resolution functions.
```

The file is organized into columns. The far left column is the type of name database. The rest of the columns are the methods the resolution functions should use to lookup a name. The methods are followed by the functions from left to right. Columns with `[]` are used to provide some limited conditional logic to the column immediately to the left of it.

Suppose a process is trying to resolve the host name `learning.lpi.org`. It would make an appropriate C library call (most likely `gethostbyname`). This function will then read `/etc/nsswitch.conf`. Since the process is looking up a host name, it will find the line starting with `hosts`. It would then attempt to use DNS to resolve the name. The next column, `[!UNAVAIL=return]` means that if the service is *not* unavailable, then do not try the next source, i.e., if DNS is available, stop trying to resolve the host name even if the name servers are unable to.

If DNS is unavailable, then continue on to the next source. In this case, the next source is `files`.

When you see a column in the format `[result=action]`, it means that when a resolver lookup of the column to the left of it is `result`, then `action` is performed. If `result` is preceded with a `!`, it means if the result is not `result`, then perform `action`. For descriptions of the possible results and actions, see the man page.

Now suppose a process is trying to resolve a port number to a service name. It would read the `services` line. The first source listed is `NIS`. `NIS` stands for *Network Information Service* (it is sometimes referred to as yellow pages). It is an old service that allowed central management of things such as users. It is rarely used anymore due to its weak security. The next column `[NOTFOUND=return]` means that if the lookup succeeded but the service was not found to stop looking. If the aforementioned condition does not apply, use local files.

Anything to the right of `#` is a comment and ignored.

## The `/etc/resolv.conf` File

The file `/etc/resolv.conf` is used to configure host resolution via DNS. Some distributions have startup scripts, daemons, and other tools that write to this file. Keep this in mind when manually editing this file. Check your distribution and any network configuration tools documentation if this is the case. Some tools, such as Network Manager will leave a comment in the file letting you know that manual changes will be overwritten.

As with `/etc/nsswitch.conf`, there is a man page associated with the file. It can be accessed with the command `man resolv.conf` or at <https://man7.org/linux/man-pages/man5/resolv.conf.5.html>.

The file format is rather straight forward. In the far left column, you have the option name. The rest of the columns on the same line are the option's value.

The most common option is the `nameserver` option. It is used to specify the IPv4 or IPv6 address of a DNS server. As of the date of this writing, you can specify up to three name servers. If your `/etc/resolv.conf` does not have a `nameserver` option, your system will by default use the name server on the local machine.

Below is a simple example file that is representative of common configurations:

```
search lpi.org
nameserver 10.0.0.53
nameserver fd00:ffff::2:53
```

The `search` option is used to allow short form searches. In the example, a single search domain of `lpi.org` is configured. This means that any attempt to resolve a host name without a domain portion will have `.lpi.org` appended before the search. For example, if you were to try to search for a host named `learning`, the resolver would search for `learning.lpi.org`. You can have up to six search domains configured.

Another common option is the `domain` option. This is used to set your local domain name. If this option is missing, this defaults to everything after the first `.` in the machine's host name. If the host name does not contain a `.`, it is assumed that the machine is part of the root domain. Like `search`, `domain` can be used for short name searches.

Keep in mind that `domain` and `search` are mutually exclusive. If both are present, the last instance in the file is used.

There are several options that can be set to affect the behavior of the resolver. To set these, use the `options` keyword, followed by the name of the option to set, and if applicable, a `:` followed by the value. Below is an example of setting the `timeout` option, which is the length of time in seconds the resolver will wait for a name server before giving up:

```
option timeout:3
```

There are other options to `resolv.conf`, but these are the most common.

## The `/etc/hosts` File

The file `/etc/hosts` is used to resolve names to IP addresses and vice versa. Both IPv4 and IPv6 are supported. The left column is the IP address, the rest are names associated with that address. The most common use for `/etc/hosts` is for hosts and addresses where DNS is not possible, such as loop back addresses. In the example below, IP addresses of critical infrastructure components are defined.

Here is a realistic example of an `/etc/hosts` file:

```
127.0.0.1    localhost
127.0.1.1    proxy
::1         localhost ip6-localhost ip6-loopback
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters

10.0.0.1    gateway.lpi.org gateway gw
fd00:ffff::1 gateway.lpi.org gateway gw
```

```
10.0.1.53      dns1.lpi.org
fd00:ffff::1:53 dns1.lpi.org
10.0.2.53      dns2.lpi.org
fd00:ffff::2:53 dns2.lpi.org
```

## systemd-resolved

Systemd provides a service called `systemd-resolved`. It provides mDNS, DNS, and LLMNR. When it is running, it listens for DNS requests on `127.0.0.53`. It does *not* provide a full fledged DNS server. Any DNS requests it receives are looked up by querying servers configured in `/etc/systemd/resolv.conf` or `/etc/resolv.conf`. If you wish to use this, use `resolve` for hosts in `/etc/nsswitch.conf`. Keep in mind that the OS package that has the `systemd-resolved` library may not be installed by default.

## Name Resolution Tools

There are many tools available to Linux users for name resolution. This lesson covers three. One, `getent`, is useful for seeing how real world requests will resolve. Another command is `host`, which is useful for simple DNS queries. A program called `dig` is useful for complex DNS operations that can aid with troubleshooting DNS server problems.

### The `getent` Command

The `getent` utility is used to display entries from name service databases. It can retrieve records from any source configurable by `/etc/nsswitch.conf`.

To use `getent`, follow the command with the type of name you wish to resolve and optionally a specific entry to lookup. If you only specify the type of name, `getent` will attempt to display all entries of that data type:

```
$ getent hosts
127.0.0.1      localhost
127.0.1.1      proxy
10.0.1.53      dns1.lpi.org
10.0.2.53      dns2.lpi.org
127.0.0.1      localhost ip6-localhost ip6-loopback
$ getent hosts dns1.lpi.org
fd00:ffff::1:53 dns1.lpi.org
```

Starting with glibc version 2.2.5, you can force `getent` to use a specific data source with the `-s` option. The example below demonstrates this:

```
$ getent -s files hosts learning.lpi.org
::1          learning.lpi.org
$ getent -s dns hosts learning.lpi.org
208.94.166.198 learning.lpi.org
```

## The `host` Command

`host` is a simple program for looking up DNS entries. With no options, if `host` is given a name, it returns the A, AAAA, and MX record sets. If given an IPv4 or IPv6 address, it outputs the PTR record if one is available:

```
$ host wikipedia.org
wikipedia.org has address 208.80.154.224
wikipedia.org has IPv6 address 2620:0:861:ed1a::1
wikipedia.org mail is handled by 10 mx1001.wikimedia.org.
wikipedia.org mail is handled by 50 mx2001.wikimedia.org.
$ host 208.80.154.224
224.154.80.208.in-addr.arpa domain name pointer text-lb.eqiad.wikimedia.org.
```

If you are looking for a specific record type, you can use `host -t`:

```
$ host -t NS lpi.org
lpi.org name server dns1.easydns.com.
lpi.org name server dns3.easydns.ca.
lpi.org name server dns2.easydns.net.
$ host -t SOA lpi.org
lpi.org has SOA record dns1.easydns.com. zone.easydns.com. 1593109612 3600 600 1209600 300
```

`host` can also be used to query a specific name server if you do not wish to use the ones in `/etc/resolv.conf`. Simply add the IP address or host name of the server you wish to use as the last argument:

```
$ host -t MX lpi.org dns1.easydns.com
Using domain server:
Name: dns1.easydns.com
Address: 64.68.192.10#53
Aliases:
```

```

lpi.org mail is handled by 10 aspmx4.googlemail.com.
lpi.org mail is handled by 10 aspmx2.googlemail.com.
lpi.org mail is handled by 5 alt1.aspmx.l.google.com.
lpi.org mail is handled by 0 aspmx.l.google.com.
lpi.org mail is handled by 10 aspmx5.googlemail.com.
lpi.org mail is handled by 10 aspmx3.googlemail.com.
lpi.org mail is handled by 5 alt2.aspmx.l.google.com.

```

## The dig Command

Another tool for querying DNS servers is `dig`. This command is much more verbose than `host`. By default, `dig` queries for A records. It is probably too verbose for simply looking up an IP address or host name. `dig` will work for simple lookups, but it is more suited for troubleshooting DNS server configuration:

```

$ dig learning.lpi.org

; <<>> DiG 9.11.5-P4-5.1+deb10u1-Debian <<>> learning.lpi.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63004
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:;, udp: 4096
; COOKIE: ca7a415be1cec45592b082665ef87f3483b81ddd61063c30 (good)
;; QUESTION SECTION:
;learning.lpi.org.      IN  A

;; ANSWER SECTION:
learning.lpi.org.     600 IN  A    208.94.166.198

;; AUTHORITY SECTION:
lpi.org.              86400 IN  NS   dns2.easydns.net.
lpi.org.              86400 IN  NS   dns1.easydns.com.
lpi.org.              86400 IN  NS   dns3.easydns.ca.

;; ADDITIONAL SECTION:
dns1.easydns.com.    172682 IN  A    64.68.192.10
dns2.easydns.net.    170226 IN  A    198.41.222.254
dns1.easydns.com.    172682 IN  AAAA 2400:cb00:2049:1::a29f:1835
dns2.easydns.net.    170226 IN  AAAA 2400:cb00:2049:1::c629:defe

```

```
;; Query time: 135 msec
;; SERVER: 192.168.1.20#53(192.168.1.20)
;; WHEN: Sun Jun 28 07:29:56 EDT 2020
;; MSG SIZE rcvd: 266
```

As you can see, `dig` provides a lot of information. The output is divided into sections. The first section displays information about the version of `dig` installed and the query sent, along with any options used for the command. Next it shows information about the query and the response.

The next section shows information about EDNS extensions used and the query. In the example, the cookie extension is used. `dig` is looking for an A record for `learning.lpi.org`.

The next section shows the result of the query. The number in the second column is the TTL of the resource in seconds.

The rest of the output provides information about the domain's name servers, including the NS records for the server along with the A and AAAA records of the servers in the domain's NS record.

Like `host`, you can specify a record type with the `-t` option:

```
$ dig -t SOA lpi.org
```

```
; <<>> DiG 9.11.5-P4-5.1+deb10u1-Debian <<>> -t SOA lpi.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16695
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 6

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 185c67140a63baf46c4493215ef8906f7bfbe15bdca3b01a (good)
;; QUESTION SECTION:
;lpi.org.                IN      SOA

;; ANSWER SECTION:
lpi.org.                600 IN    SOA dns1.easydns.com. zone.easydns.com. 1593109612 3600 600 1209600
300

;; AUTHORITY SECTION:
lpi.org.                81989 IN    NS   dns1.easydns.com.
lpi.org.                81989 IN    NS   dns2.easydns.net.
```



```
lpi.org.          81989  IN  NS  dns3.easydns.ca.

;; ADDITIONAL SECTION:
dns1.easydns.com. 168271 IN  A   64.68.192.10
dns2.easydns.net. 165815 IN  A   198.41.222.254
dns3.easydns.ca.  107 IN  A   64.68.196.10
dns1.easydns.com. 168271 IN  AAAA 2400:cb00:2049:1::a29f:1835
dns2.easydns.net. 165815 IN  AAAA 2400:cb00:2049:1::c629:defe

;; Query time: 94 msec
;; SERVER: 192.168.1.20#53(192.168.1.20)
;; WHEN: Sun Jun 28 08:43:27 EDT 2020
;; MSG SIZE rcvd: 298
```

Dig has many options to fine tune both the output and query sent to the server. These options begin with +. One option is the `short` option, which suppresses all output except the result:

```
$ dig +short lpi.org
65.39.134.165
$ dig +short -t SOA lpi.org
dns1.easydns.com. zone.easydns.com. 1593109612 3600 600 1209600 300
```

Here is an example of turning off the cookie EDNS extension:

```
$ dig +nocookie -t MX lpi.org

; <<>> DiG 9.11.5-P4-5.1+deb10u1-Debian <<>> +nocookie -t MX lpi.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47774
;; flags: qr rd ra; QUERY: 1, ANSWER: 7, AUTHORITY: 3, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;lpi.org.          IN  MX

;; ANSWER SECTION:
lpi.org.          468 IN  MX  0 aspmx.l.google.com.
lpi.org.          468 IN  MX  10 aspmx4.googlemail.com.
lpi.org.          468 IN  MX  10 aspmx5.googlemail.com.
lpi.org.          468 IN  MX  10 aspmx2.googlemail.com.
```

```
lpi.org.      468 IN  MX  10 aspmx3.googlemail.com.
lpi.org.      468 IN  MX  5  alt2.aspmx.l.google.com.
lpi.org.      468 IN  MX  5  alt1.aspmx.l.google.com.

;; AUTHORITY SECTION:
lpi.org.      77130 IN  NS  dns2.easydns.net.
lpi.org.      77130 IN  NS  dns3.easydns.ca.
lpi.org.      77130 IN  NS  dns1.easydns.com.

;; ADDITIONAL SECTION:
dns1.easydns.com.  76140 IN  A   64.68.192.10
dns2.easydns.net.  73684 IN  A   198.41.222.254
dns1.easydns.com.  76140 IN  AAAA 2400:cb00:2049:1::a29f:1835
dns2.easydns.net.  73684 IN  AAAA 2400:cb00:2049:1::c629:de

;; Query time: 2 msec
;; SERVER: 192.168.1.20#53(192.168.1.20)
;; WHEN: Mon Jun 29 10:18:58 EDT 2020
;; MSG SIZE rcvd: 389
```

## Guided Exercises

1. What will the command below do?

```
getent group openldap
```

2. What is the biggest difference between `getent` and the other tools covered, `host` and `dig`?

3. Which option to `dig` and `host` is used to specify the type of record you wish to retrieve?

4. Which of the following is a proper `/etc/hosts` entry?

```
:::1 localhost
```

```
localhost 127.0.0.1
```

5. Which option to `getent` is used to specify which data source should be used to perform a lookup?

## Explorational Exercises

1. If you were to edit the `/etc/resolv.conf` below with a text editor, what is likely to happen?

```
# Generated by NetworkManager
nameserver 192.168.1.20
```

The changes will be overwritten by NetworkManager.	
NetworkManager will update its configuration with your changes.	
Your changes won't affect the system.	
NetworkManager will be disabled.	

2. What does the following line in `/etc/nsswitch.conf` mean:

```
hosts: files [SUCCESS=continue] dns
```

3. Considering the following `/etc/resolv.conf` why isn't the system resolving names through DNS?

```
search lpi.org
#nameserver fd00:ffff::1:53
#nameserver 10.0.1.53
```

4. What does the command `dig +noall +answer +question lpi.org` do?

5. How can you override the defaults of `dig` without specifying them on the command line?

## Summary

The `getent` command is a great tool to see the results of resolver calls. For simple DNS queries, `host` is easy to use and produces straightforward output. If you need detailed information or need to fine tune a DNS query, `dig` is most likely your best bet.

Due to the ability to add shared library plugins and configure the resolver behavior, Linux has excellent support for name and number resolution of various types. The `getent` program can be used to resolve names using the resolver libraries. `host` and `dig` can be used to query DNS servers.

The file `/etc/nsswitch.conf` is used to configure the resolver behavior. You are able to change data sources and add some simple conditional logic for name types with multiple sources.

DNS is configured by editing `/etc/resolv.conf`. Many distributions have tools that manage this file for you, so make sure to check your system's documentation if manual changes aren't persisting.

The `/etc/hosts` file is used to resolve host names to IPs and vice versa. It is typically used to define names, such as `localhost`, that aren't available through DNS.

It is possible to leave comments in the configuration files covered in this lesson. Any text to the right of `#` is ignored by the system.

## Answers to Guided Exercises

1. What will the command below do?

```
getent group openldap
```

It will read `/etc/nsswitch.conf`, lookup the `openldap` group from the sources listed, and display information about it if it is found.

2. What is the biggest difference between `getent` and the other tools covered, `host` and `dig`?

`getent` looks up names using the resolver libraries, the others just query DNS. `getent` can be used to troubleshoot your `/etc/nsswitch.conf` and the configuration of name resolution libraries your system is configured to use. `host` and `dig` are used to lookup DNS records.

3. Which option to `dig` and `host` is used to specify the type of record you wish to retrieve?

Both programs use `-t` to specify the type of record you wish to look up.

4. Which of the following is a proper `/etc/hosts` entry?

<code>::1 localhost</code>	X
<code>localhost 127.0.0.1</code>	

`::1 localhost` is the correct line. The left column is always an IPv4 or IPv6 address.

5. Which option to `getent` is used to specify which data source should be used to perform a lookup?

The `-s` option is used to specify the data source. For example:

```
$ getent -s files hosts learning.lpi.org
192.168.10.25 learning.lpi.org
$ getent -s dns hosts learning.lpi.org
208.94.166.198 learning.lpi.org
```

## Answers to Explorational Exercises

1. If you were to edit the `/etc/resolv.conf` below with a text editor, what is likely to happen?

```
# Generated by NetworkManager
nameserver 192.168.1.20
```

The changes will be overwritten by NetworkManager.	X
NetworkManager will update its configuration with your changes.	
Your changes won't affect the system.	
NetworkManager will be disabled.	

2. What does the following line in `/etc/nsswitch.conf` mean:

```
hosts: files [SUCCESS=continue] dns
```

Lookups for host names will check your `/etc/hosts` files first and then DNS. If an entry found is found in files and DNS, the entry in DNS will be used.

3. Considering the following `/etc/resolv.conf` why isn't the system resolving names through DNS?

```
search lpi.org
#nameserver fd00:ffff::1:53
#nameserver 10.0.1.53
```

Both DNS servers are commented out and there is no DNS server running on the local host.

4. What does the command `dig +noall +answer +question lpi.org` do?

It looks up the A record for `lpi.org` and displays only the query and response.

5. How can you override the defaults of `dig` without specifying them on the command line?

You create a `.digrc` file in your home directory.



**Linux  
Professional  
Institute**

## **Topic 110: Security**





**Linux  
Professional  
Institute**

## 110.1 Perform security administration tasks

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 110.1](#)

### Weight

3

### Key knowledge areas

- Audit a system to find files with the suid/sgid bit set.
- Set or change user passwords and password aging information.
- Being able to use nmap and netstat to discover open ports on a system.
- Set up limits on user logins, processes and memory usage.
- Determine which users have logged in to the system or are currently logged in.
- Basic sudo configuration and usage.

### Partial list of the used files, terms and utilities

- `find`
- `passwd`
- `fuser`
- `lsof`
- `nmap`
- `chage`
- `netstat`
- `sudo`

- `/etc/sudoers`
- `su`
- `usermod`
- `ulimit`
- `who, w, last`



# 110.1 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	110 Security
<b>Objective:</b>	110.1 Perform security administration tasks
<b>Lesson:</b>	1 of 1

## Introduction

Security is a must in system administration. As a good Linux sysadmin you have to keep an eye on a number of things such as special permissions on files, user password aging, open ports and sockets, limiting the use of system resources, dealing with logged-in users, and privilege escalation through `su` and `sudo`. In this lesson we will be reviewing each one of these topics.

## Checking for Files with the SUID and SGID Set

Aside from the traditional permission set of *read*, *write* and *execute*, files in a Linux system may also have some special permissions set such as the *SUID* or the *SGID* bits.

The SUID bit will allow the file to be executed with the privileges of the file's owner. It is numerically represented by `4000` and symbolically represented by either `s` or `S` on the owner's *execute* permission bit. A classic example of an executable file with the SUID permission set is `passwd`:

```
carol@debian:~$ ls -l /usr/bin/passwd
```

```
-rwsr-xr-x 1 root root 63736 jul 27 2018 /usr/bin/passwd
```

The lowercase `s` in `rws` indicates the presence of the SUID on the file — together with the *execute* permission. An uppercase `S` instead (`rwS`) would mean the underlying *execute* permission is not set.

**NOTE**

You will learn about `passwd` in the next section. The utility is mostly used by `root` to set/change users' passwords (e.g.: `passwd carol`). However, regular users can use it to change their own passwords, too. Therefore it comes with the SUID set.

On the other hand, the SGID bit can be set both on files and directories. With files, its behaviour is equivalent to that of SUID but the privileges are those of the group owner. When set on a directory, however, it will allow all files created therein to inherit the ownership of the directory's group. Like SUID, SGID is symbolically represented by either `s` or `S` on the group's *execute* permission bit. Numerically, it is represented by `2000`. You can set the SGID on a directory by using `chmod`. You have to add 2 (SGID) to the traditional permissions (755 in our case):

```
carol@debian:~$ ls -ld shared_directory
drwxr-xr-x 2 carol carol 4096 may 30 23:55 shared_directory
carol@debian:~$ sudo chmod 2755 shared_directory/
carol@debian:~$ ls -ld shared_directory
drwxr-sr-x 2 carol carol 4096 may 30 23:55 shared_directory
```

To find files with either or both the SUID and SGID set you can use the `find` command and make use of the `-perm` option. You can use both numeric and symbolic values. The values—in turn—can be passed on their own or preceded by a dash (`-`) or a forward slash (`/`). The meaning is as follows:

**`-perm numeric-value` or `-perm symbolic-value`**

find files having the special permission *exclusively*

**`-perm -numeric-value` or `-perm -symbolic-value`**

find files having the special permission and other permissions

**`-perm /numeric-value` or `-perm /symbolic-value`**

find files having either of the special permission (and other permissions)

For example, to find files with *only* SUID set in the present working directory, you will use the following command:

```
carol@debian:~$ find . -perm 4000
carol@debian:~$ touch file
carol@debian:~$ chmod 4000 file
carol@debian:~$ find . -perm 4000
./file
```

Note how — as there were not any files having exclusively the SUID — we created one to show some output. You can run the same command in symbolic notation:

```
carol@debian:~$ find . -perm u+s
./file
```

To find files matching SUID (irrespective of any other permissions) in the `/usr/bin/` directory, you can use either of the following commands:

```
carol@debian:~$ sudo find /usr/bin -perm -4000
/usr/bin/umount
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/mount
/usr/bin/passwd
/usr/bin/chsh
/usr/bin/sudo
/usr/bin/su
carol@debian:~$ sudo find /usr/bin -perm -u+s
/usr/bin/umount
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/mount
/usr/bin/passwd
/usr/bin/chsh
/usr/bin/sudo
/usr/bin/su
```

If you are looking for files in the same directory with the SGID bit set, you can execute `find /usr/bin/ -perm -2000` or `find /usr/bin/ -perm -g+s`.

Finally, to find files with either of the two special permissions set, add 4 and 2 and use `/`:

```
carol@debian:~$ sudo find /usr/bin -perm /6000
/usr/bin/dotlock.mailutils
/usr/bin/umount
/usr/bin/newgrp
/usr/bin/wall
/usr/bin/ssh-agent
/usr/bin/chage
/usr/bin/dotlockfile
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/mount
/usr/bin/passwd
/usr/bin/chsh
/usr/bin/expiry
/usr/bin/sudo
/usr/bin/bsd-write
/usr/bin/crontab
/usr/bin/su
```

## Password Management and Aging

As noted above you can use the `passwd` utility to change your own password as a regular user. Additionally, you can pass the `-S` or `--status` switch to get status information about your account:

```
carol@debian:~$ passwd -S
carol P 12/07/2019 0 99999 7 -1
```

Here is a breakdown of the seven fields you get in the output:

### **carol**

User's login name.

### **P**

It indicates that the user has a valid password (P); other possible values are L for a locked password and NP for no password.

### **12/07/2019**

Date of the last password change.

**0**

Minimum age in days (the minimum number of days between password changes). A value of 0 means the password may be changed at any time.

**99999**

Maximum age in days (the maximum number of days the password is valid for). A value of 99999 will disable password expiration.

**7**

Warning period in days (the number of days prior to password expiration that a user will be warned).

**-1**

Password inactivity period in days (the number of inactive days after password expiration before the account is locked). A value of -1 will remove an account's inactivity.

Apart from reporting on account status, you will use the `passwd` command as root to carry out some basic account maintenance. You can lock and unlock accounts, force a user to change their password on the next login and delete a user's password with the `-l`, `-u`, `-e` and `-d` options, respectively.

To test these options it is convenient to introduce the `su` command at this point. Through `su` you can change users during a login session. So, for example, let us use `passwd` as root to lock `carol`'s password. Then we will switch to `carol` and check our account status to verify that the password has—in fact—been locked (L) and cannot be changed. Finally, going back to the root user, we will unlock `carol`'s password:

```
root@debian:~# passwd -l carol
passwd: password expiry information changed.
root@debian:~# su - carol
carol@debian:~$ passwd -S
carol L 05/31/2020 0 99999 7 -1
carol@debian:~$ passwd
Changing password for carol.
Current password:
passwd: Authentication token manipulation error
passwd: password unchanged
carol@debian:~$ exit
logout
root@debian:~# passwd -u carol
passwd: password expiry information changed.
```

Alternatively, you can also lock and unlock a user's password with the `usermod` command:

### Lock user `carol`'s password

```
usermod -L carol or usermod --lock carol.
```

### Unlock user `carol`'s password

```
usermod -U carol or usermod --unlock carol.
```

#### NOTE

With the `-f` or `--inactive` switches, `usermod` can also be used to set the number of days before an account with an expired password is disabled (e.g.: `usermod -f 3 carol`).

Aside from `passwd` and `usermod`, the most direct command to deal with password and account aging is `chage` (“change age”). As root, you can pass `chage` the `-l` (or `--list`) switch followed by a username to have that user's current password and account expiry information printed on the screen; as a regular user, you can view your own information:

```
carol@debian:~$ chage -l carol
Last password change           : Aug 06, 2019
Password expires                : never
Password inactive              : never
Account expires                : never
Minimum number of days between password change : 0
Maximum number of days between password change  : 99999
Number of days of warning before password expires : 7
```

Run without options and only followed by a username, `chage` will behave interactively:

```
root@debian:~# chage carol
Changing the aging information for carol
Enter the new value, or press ENTER for the default

Minimum Password Age [0]:
Maximum Password Age [99999]:
Last Password Change (YYYY-MM-DD) [2020-06-01]:
Password Expiration Warning [7]:
Password Inactive [-1]:
Account Expiration Date (YYYY-MM-DD) [-1]:
```

The options to modify the different `chage` settings are as follows:



**-m days username or --mindays days username**

Specify minimum number of days between password changes (e.g.: `chage -m 5 carol`). A value of `0` will enable the user to change his/her password at any time.

**-M days username or --maxdays days username**

Specify maximum number of days the password will be valid for (e.g.: `chage -M 30 carol`). To disable password expiration, it is customary to give this option a value of `99999`.

**-d days username or --lastday days username**

Specify number of days since the password was last changed (e.g.: `chage -d 10 carol`). A value of `0` will force the user to change their password on the next login.

**-W days username or --warndays days username**

Specify number of days the user will be reminded of their password being expired.

**-I days username or --inactive days username**

Specify number of inactive days after password expiration (e.g.: `chage -I 10 carol`)—the same as `usermod -f` or `usermod --inactive`. Once that number of days has gone by, the account will be locked. With a value of `0`, the account will not be locked, though.

**-E date username or --expiredate date username**

Specify date (or number of days since *the epoch*—January, 1st 1970) on which the account will be locked. It is normally expressed in the format `YYYY-MM-DD`(e.g.: `chage -E 2050-12-13 carol`).

**NOTE** You can learn more about `passwd`, `usermod` and `chage`—and their options—by consulting their respective manual pages.

## Discovering Open Ports

When it comes to keeping an eye on open ports, four powerful utilities are present on most Linux systems: `lsof`, `fuser`, `netstat` and `nmap`. We will cover them in this section.

`lsof` stands for “list open files”, which is no small thing considering that—for Linux—everything is a file. In fact, if you type `lsof` into the terminal, you will get a large listing of regular files, device files, sockets, etc. However, for the sake of this lesson, we will mainly focus on ports. To print the listing of all “Internet” network files, run `lsof` with the `-i` option:

```
root@debian:~# lsof -i
COMMAND  PID    USER  FD   TYPE DEVICE SIZE/OFF NODE NAME
```

```

dhclient 357    root    7u    IPv4  13493    0t0  UDP  *:bootpc
sshd     389     root    3u    IPv4  13689    0t0  TCP  *:ssh (LISTEN)
sshd     389     root    4u    IPv6  13700    0t0  TCP  *:ssh (LISTEN)
apache2  399     root    3u    IPv6  13826    0t0  TCP  *:http (LISTEN)
apache2  401    www-data 3u    IPv6  13826    0t0  TCP  *:http (LISTEN)
apache2  402    www-data 3u    IPv6  13826    0t0  TCP  *:http (LISTEN)
sshd     557     root    3u    IPv4  14701    0t0  TCP  192.168.1.7:ssh->192.168.1.4:60510
(ESTABLISHED)
sshd     569     carol   3u    IPv4  14701    0t0  TCP  192.168.1.7:ssh->192.168.1.4:60510
(ESTABLISHED)

```

Apart from the `bootpc` service — which is used by DHCP — the output shows two services listening for connections — `ssh` and the Apache web server (`http`) — as well as two established SSH connections. You can specify a particular host with the `@ip-address` notation to check for its connections:

```

root@debian:~# lsof -i@192.168.1.7
COMMAND PID  USER  FD  TYPE DEVICE SIZE/OFF NODE NAME
sshd    557  root   3u  IPv4  14701      0t0  TCP  192.168.1.7:ssh->192.168.1.4:60510
(ESTABLISHED)
sshd    569  carol  3u  IPv4  14701      0t0  TCP  192.168.1.7:ssh->192.168.1.4:60510
(ESTABLISHED)

```

**NOTE** To print only IPv4 and IPv6 network files, use the `-i4` and `-i6` options respectively.

Likewise, you can filter by port by passing the `-i` (or `-i@ip-address`) option the `:port` argument:

```

root@debian:~# lsof -i :22
COMMAND PID  USER  FD  TYPE DEVICE SIZE/OFF NODE NAME
sshd    389  root   3u  IPv4  13689      0t0  TCP  *:ssh (LISTEN)
sshd    389  root   4u  IPv6  13700      0t0  TCP  *:ssh (LISTEN)
sshd    557  root   3u  IPv4  14701      0t0  TCP  192.168.1.7:ssh->192.168.1.4:60510
(ESTABLISHED)
sshd    569  carol  3u  IPv4  14701      0t0  TCP  192.168.1.7:ssh->192.168.1.4:60510
(ESTABLISHED)

```

Multiple ports are separated by commas (and ranges are specified using a dash):

```

root@debian:~# lsof -i@192.168.1.7:22,80
COMMAND PID  USER  FD  TYPE DEVICE SIZE/OFF NODE NAME

```

```
sshd    705  root    3u  IPv4  13960    0t0  TCP  192.168.1.7:ssh->192.168.1.4:44766
(ESTABLISHED)
sshd    718  carol   3u  IPv4  13960    0t0  TCP  192.168.1.7:ssh->192.168.1.4:44766
(ESTABLISHED)
```

**NOTE**

The amount of options available to `lsof` is quite impressive. To find out more, consult its manual page.

Next on the list of network commands is `fuser`. Its main purpose is to find a “file’s user” — which involves knowing what processes are accessing what files; it also gives you some other information such as the type of access. For example, to check on the current working directory, it is sufficient to run `fuser .` However, to get a little more information, it is convenient to use the verbose (`-v` or `--verbose`) option:

```
root@debian:~# fuser .
/root:                    580c
root@debian:~# fuser -v .
                USER      PID ACCESS COMMAND
/root:          root      580  ..c..  bash
```

Let us break down the output:

**File**

The file we are getting information about (`/root`).

**USER Column**

The owner of the file (`root`).

**PID Column**

The process identifier (`580`).

**ACCESS Column**

Type of access (`..c..`). One of:

**c**

Current directory.

**e**

Executable being run.

**f**

Open file (omitted in default display mode).

**F**

Open file for writing (omitted in default display mode).

**r**

root directory.

**m**

mmap'ed file or shared library.

**.**

Placeholder (omitted in default display mode).

**COMMAND Column**The command affiliated with the file (`bash`).

With the `-n` (or `--namespace`) option, you can find information about network ports/sockets. You must also supply the network protocol and port number. Thus, to get information about the Apache web server you will run the following command:

```
root@debian:~# fuser -vn tcp 80
80/tcp:          USER      PID ACCESS COMMAND
              root        402 F.... apache2
              www-data  404 F.... apache2
              www-data  405 F.... apache2
```

**NOTE**

`fuser` can be also be used to kill the processes accessing the file with the `-k` or `--kill` switches (e.g.: `fuser -k 80/tcp`). Refer to the manual page for more detailed information.

Let us turn to `netstat` now. `netstat` is a very versatile network tool that is mostly used to print “network statistics”.

Run without options, `netstat` will display both active Internet connections and Unix sockets. Because of the size of the listing, you may want to pipe its output through `less`:

```
carol@debian:~$ netstat |less
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
```

```

tcp      0      0 192.168.1.7:ssh      192.168.1.4:55444    ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags      Type       State          I-Node  Path
unix  2      [ ]       DGRAM          10509    /run/systemd/journal/syslog
unix  3      [ ]       DGRAM          10123    /run/systemd/notify
(...)

```

To list only “listening” ports and sockets, you will use the `-l` or `--listening` options. The `-t/--tcp` and `-u/--udp` options can be added to filter by TCP and UDP protocol, respectively (they can also be combined in the same command). Likewise, `-e/--extend` will display additional information:

```

carol@debian:~$ netstat -lu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp      0      0 0.0.0.0:bootpc        0.0.0.0:*
carol@debian:~$ netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:ssh           0.0.0.0:*              LISTEN
tcp      0      0 localhost:smtp        0.0.0.0:*              LISTEN
tcp6     0      0 [::]:http            [::]:*                 LISTEN
tcp6     0      0 [::]:ssh             [::]:*                 LISTEN
tcp6     0      0 localhost:smtp       [::]:*                 LISTEN
carol@debian:~$ netstat -lute
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State      User
Inode
tcp      0      0 0.0.0.0:ssh           0.0.0.0:*              LISTEN    root
13729
tcp      0      0 localhost:smtp        0.0.0.0:*              LISTEN    root
14372
tcp6     0      0 [::]:http            [::]:*                 LISTEN    root
14159
tcp6     0      0 [::]:ssh             [::]:*                 LISTEN    root
13740
tcp6     0      0 localhost:smtp       [::]:*                 LISTEN    root
14374
udp      0      0 0.0.0.0:bootpc        0.0.0.0:*              LISTEN    root
13604

```

If you omit `-l` option, *only* established connections will be shown:

```
carol@debian:~$ netstat -ute
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       User
Inode
tcp        0      0 192.168.1.7:ssh        192.168.1.4:39144     ESTABLISHED root
15103
```

If you are only interested in numerical information concerning ports and hosts, you can use the `-n` or `--numeric` option to print only port numbers and IP addresses. Note how `ssh` turns into to `22` when adding `-n` to the command above:

```
carol@debian:~$ netstat -uten
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       User
Inode
tcp        0      0 192.168.1.7:22        192.168.1.4:39144     ESTABLISHED 0
15103
```

As you can see, you can make very useful and productive `netstat` commands by combining some of its options. Browse through the man pages to learn more and find the combinations that best suit your needs.

Finally, we will introduce `nmap`—or the “network mapper”. Another very powerful utility, this port scanner is executed by specifying an IP address or hostname:

```
root@debian:~# nmap localhost
Starting Nmap 7.70 ( https://nmap.org ) at 2020-06-04 19:29 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000040s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http

Nmap done: 1 IP address (1 host up) scanned in 1.58 seconds
```

Aside from a single host, `nmap` allows you to scan:

## multiple hosts

by separating them by spaces (e.g.: `nmap localhost 192.168.1.7`).

## host ranges

by using a dash (e.g.: `nmap 192.168.1.3-20`).

## subnets

by using a wildcard or CIDR notation (e.g.: `nmap 192.168.1.*` or `nmap 192.168.1.0/24`).

You can exclude particular hosts (e.g.: `nmap 192.168.1.0/24 --exclude 192.168.1.7`).

To scan a particular port, use the `-p` switch followed by the port number or service name (`nmap -p 22` and `nmap -p ssh` will give you the same output):

```

root@debian:~# nmap -p 22 localhost
Starting Nmap 7.70 ( https://nmap.org ) at 2020-06-04 19:54 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000024s latency).
Other addresses for localhost (not scanned): ::1

PORT      STATE SERVICE
22/tcp    open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds

```

You can also scan multiple ports or port ranges by using commas and dashes, respectively:

```

root@debian:~# nmap -p ssh,80 localhost
Starting Nmap 7.70 ( https://nmap.org ) at 2020-06-04 19:58 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000051s latency).
Other addresses for localhost (not scanned): ::1

PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http

Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds

```

```

root@debian:~# nmap -p 22-80 localhost
Starting Nmap 7.70 ( https://nmap.org ) at 2020-06-04 19:58 CEST
Nmap scan report for localhost (127.0.0.1)

```

```
Host is up (0.000011s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 57 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http

Nmap done: 1 IP address (1 host up) scanned in 1.47 seconds
```

Two other important and handy `nmap` options are:

### **-F**

Run a fast scan on the 100 most common ports.

### **-v**

Get verbose output (`-vv` will print even more verbose output).

### **NOTE**

`nmap` can run quite complex commands by making use of scan types. However, that topic is outside of the scope of this lesson.

## Limits on Users Logins, Processes and Memory Usage

Resources on a Linux system are not unlimited so — as a sysadmin — you should ensure a good balance between *user limits* on resources and the proper functioning of the operating system. `ulimit` can help you out in that regard.

`ulimit` deals with *soft* and *hard* limits — specified by the `-S` and `-H` options, respectively. Run without options or arguments, `ulimit` will display the soft file blocks of the current user:

```
carol@debian:~$ ulimit
unlimited
```

With the `-a` option, `ulimit` will show all current soft limits (the same as `-Sa`); to display all current hard limits, use `-Ha`:

```
carol@debian:~$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
(...)
carol@debian:~$ ulimit -Ha
```



```

core file size      (blocks, -c) unlimited
data seg size      (kbytes, -d) unlimited
scheduling priority      (-e) 0
(...)

```

The available shell resources are specified by options such as:

**-b**

maximum socket buffer size

**-f**

maximum size of files written by the shell and its children

**-l**

maximum size that may be locked into memory

**-m**

maximum resident set size (RSS)—the current portion of memory held by a process in main memory (RAM)

**-v**

maximum amount of virtual memory

**-u**

maximum number of processes available to a single user

Thus, to display limits you will use `ulimit` followed by either `-S` (soft) or `-H` (hard) and the resource option; if neither `-S` or `-H` is supplied, soft limits will be shown:

```

carol@debian:~$ ulimit -u
10000
carol@debian:~$ ulimit -Su
10000
carol@debian:~$ ulimit -Hu
15672

```

Likewise, to set new limits on a particular resource you will specify either `-S` or `-H`, followed by the corresponding resource option and the new value. This value can be a number or the special words `soft` (current soft limit), `hard` (current hard limit) or `unlimited` (no limit). If neither `-S` or `-H` is specified, both limits will be set. For example, let us first read the current maximum size value for files written by the shell and its children:

```
root@debian:~# ulimit -Sf
unlimited
root@debian:~# ulimit -Hf
unlimited
```

Now, let us change the value from `unlimited` to `500` blocks without specifying either `-S` or `-H`. Note how both the soft and hard limits are changed:

```
root@debian:~# ulimit -f 500
root@debian:~# ulimit -Sf
500
root@debian:~# ulimit -Hf
500
```

Finally, we will decrease only the soft limit to `200` blocks:

```
root@debian:~# ulimit -Sf 200
root@debian:~# ulimit -Sf
200
root@debian:~# ulimit -Hf
500
```

Hard limits can only be increased by the root user. On the other hand, regular users can decrease hard limits and increase soft limits up to the value of hard limits. To make new limit values persistent across reboots, you must write them to the `/etc/security/limits.conf` file. This is also the file used by the administrator to apply restrictions on particular users.

#### NOTE

Be warned that there is no `ulimit` man page as such. It is a bash builtin so you have to refer to bash man page to learn about it.

## Dealing with Logged in Users

Another of your jobs as a sysadmin entails keeping track of logged-in users. There are three utilities that can help you out with those tasks: `last`, `who` and `w`.

`last` prints a listing of the last logged in users with the most recent information on top:

```
root@debian:~# last
carol pts/0 192.168.1.4 Sat Jun 6 14:25 still logged in
reboot system boot 4.19.0-9-amd64 Sat Jun 6 14:24 still running
```

```
mimi pts/0 192.168.1.4 Sat Jun 6 12:07 - 14:24 (02:16)
reboot system boot 4.19.0-9-amd64 Sat Jun 6 12:07 - 14:24 (02:17)
(...)
wtmp begins Sun May 31 14:14:58 2020
```

Considering the truncated listing, we get information about the last two users on the system. The first two lines tell us about user `carol`; the next two lines, about user `mimi`. The information is as follows:

1. User `carol` at terminal `pts/0` from host `192.168.1.4` started her session on `Sat Jun 6` at `14:25` and is still logged in. The system—using kernel `4.19.0-9-amd64`—was started (`reboot system boot`) on `Sat Jun 6` at `14:24` and is still running.
2. User `mimi` at terminal `pts/0` from host `192.168.1.4` started her session on `Sat Jun 6` at `12:07` and logged out at `14:24` (the session lasted a total of `(02:16)` hours). The system—using kernel `4.19.0-9-amd64`—was started (`reboot system boot`) on `Sat Jun 6` at `12:07` and was powered off at `14:24` (it was up and running for `(02:17)` hours).

**NOTE** The line `wtmp begins Sun May 31 14:14:58 2020` refers to `/var/log/wtmp`, which is the special log file from which `last` gets the information.

You can pass `last` a username to have only entries for that user displayed:

```
root@debian:~# last carol
carol pts/0 192.168.1.4 Sat Jun 6 14:25 still logged in
carol pts/0 192.168.1.4 Sat Jun 6 12:07 - 14:24 (02:16)
carol pts/0 192.168.1.4 Fri Jun 5 00:48 - 01:28 (00:39)
(...)
```

Regarding the second column (terminal), `pts` stands for *Pseudo Terminal Slave*—as opposed to a proper *TeleTYpewriter* or `tty` terminal; `0` refers to the first one (the count starts at zero).

**NOTE** To check for bad login attempts, run `lastb` instead of `last`.

The `who` and `w` utilities focus on currently logged in users and are quite similar. The former displays who is logged on, while the latter also shows information on what they are doing.

When executed with no options, `who` will display four columns corresponding to logged-in user, terminal, date and time, and hostname:

```
root@debian:~# who
carol pts/0 2020-06-06 17:16 (192.168.1.4)
```

```
mimi pts/1 2020-06-06 17:28 (192.168.1.4)
```

who accepts a series of options, among which we can highlight the following:

### **-b, --boot**

Display time of last system boot.

### **-r, --runlevel**

Display current runlevel.

### **-H, --heading**

Print column headings.

Compared to who, w gives a little more verbose output:

```
root@debian:~# w
17:56:12 up 40 min, 2 users, load average: 0.04, 0.12, 0.09
USER      TTY      FROM          LOGIN@   IDLE   JCPU   PCPU WHAT
carol     pts/0    192.168.1.4   17:16    1.00s  0.15s  0.05s sshd: carol [priv]
mimi     pts/1    192.168.1.4   17:28    15:08  0.05s  0.05s -bash
```

The top line gives you information about the current time (17:56:12), how long the system has been up and running (up 40 min), the number of currently logged in users (2 users) and the load average numbers (load average: 0.04, 0.12, 0.09). These values refer to the number of jobs in the run queue averaged over the last 1, 5 and 15 minutes, respectively.

Then you find eight columns; let us break them down:

### **USER**

Login name of user.

### **TTY**

Name of terminal the user is on.

### **FROM**

Remote host from which the user has logged on.

### **LOGIN@**

Login time.

**IDLE**

Idle time.

**JCPU**

Time used by all processes attached to the tty (including currently running background jobs).

**PCPU**

Time used by current process (the one showing under **WHAT**).

**WHAT**

Command line of current process.

Just like with **who**, you can pass **w** usernames:

```
root@debian:~# w mimi
18:23:15 up 1:07, 2 users, load average: 0.00, 0.02, 0.05
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
mimi     pts/1    192.168.1.4     17:28   9:23   0.06s  0.06s -bash
```

## Basic sudo Configuration and Usage

As already noted in this lesson, **su** lets you switch to any other user in the system as long as you provide the target user's password. In the case of the root user, having its password distributed or known to (many) users puts the system at risk and is a very bad security practice. The basic usage of **su** is **su - target-username**. When changing to root—though—the target username is optional:

```
carol@debian:~$ su - root
Password:
root@debian:~# exit
logout
carol@debian:~$ su -
Password:
root@debian:~#
```

The use of the dash (-) ensures that the target user's environment is loaded. Without it, the old user's environment will be kept:

```
carol@debian:~$ su
Password:
```

```
root@debian:/home/carol#
```

On the other hand, there is the `sudo` command. With it you can execute a command as the root user—or any other user for that matter. From a security perspective, `sudo` is a far better option than `su` as it presents two main advantages:

1. to run a command as root, you do not need the root user's password, but only that of the invoking user in compliance with a security policy. The default security policy is `sudoers` as specified in `/etc/sudoers` and `/etc/sudoers.d/*`.
2. `sudo` lets you run single commands with elevated privileges instead of launching a whole new subshell for root as `su` does.

The basic usage of `sudo` is `sudo -u target-username command`. However, to run a command as user root, the `-u target-username` option is not necessary:

```
carol@debian:~$ sudo -u mimi whoami
mimi
carol@debian:~$ sudo whoami
root
```

#### NOTE

`sudoers` will use a per-user (and per-terminal) timestamp for credential caching, so that you can use `sudo` without a password for a default period of fifteen minutes. This default value can be modified by adding the `timestamp_timeout` option as a `Defaults` setting in `/etc/sudoers` (e.g.: `Defaults timestamp_timeout=1` will set credential caching timeout to one minute).

## The `/etc/sudoers` File

`sudo`'s main configuration file is `/etc/sudoers` (there is also the `/etc/sudoers.d` directory). That is the place where users' `sudo` privileges are determined. In other words, here you will specify who can run what commands as what users on what machines—as well as other settings. The syntax used is as follows:

```
carol@debian:~$ sudo less /etc/sudoers
(...)
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
```

```
(...)
```

The privilege specification for the root user is `ALL=(ALL:ALL) ALL`. This translates as: user root (root) can log in from all hosts (ALL), as all users and all groups ((ALL:ALL)), and run all commands (ALL). The same is true for members of the `sudo` group—note how group names are identified by a preceding percent sign (%).

Thus, to have user `carol` be able to check `apache2` status from any host as any user or group, you will add the following line in the `sudoers` file:

```
carol ALL=(ALL:ALL) /usr/bin/systemctl status apache2
```

You may want to save `carol` the inconvenience of having to provide her password to run the `systemctl status apache2` command. For that, you will modify the line to look like this:

```
carol ALL=(ALL:ALL) NOPASSWD: /usr/bin/systemctl status apache2
```

Say that now you want to restrict your hosts to `192.168.1.7` and enable `carol` to run `systemctl status apache2` as user `mimi`. You would modify the line as follows:

```
carol 192.168.1.7=(mimi) /usr/bin/systemctl status apache2
```

Now you can check the status of the Apache web server as user `mimi`:

```
carol@debian:~$ sudo -u mimi systemctl status apache2
• apache2.service - The Apache HTTP Server
  Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
  Active: active (running) since Tue 2020-06-09 13:12:19 CEST; 29min ago
(...)
```

If `carol` was to be promoted to `sysadmin` and you wanted to give her all privileges, the easiest approach would be that of including her in the special `sudo` group with `usermod` and the `-G` option (you may also want to use the `-a` option, which ensures that the user is not removed from any other groups they might belong to):

```
root@debian:~# sudo useradd -aG sudo carol
```

**NOTE** In the Red Hat family of distributions the `wheel` group is the counterpart to the

special administrative `sudo` group of Debian systems.

Instead of editing `/etc/sudoers` directly, you should simply use the `visudo` command as root (e.g.: `visudo`), which will open `/etc/sudoers` using your predefined text editor. To change the default text editor, you can add the `editor` option as a Defaults setting in `/etc/sudoers`. For instance, to change the editor to `nano`, you will add the following line:

```
Defaults    editor=/usr/bin/nano
```

**NOTE** Alternatively, you can specify a text editor via the `EDITOR` environment variable when using `visudo` (e.g.: `EDITOR=/usr/bin/nano visudo`)

Aside from users and groups, you can also make use of aliases in `/etc/sudoers`. There are three main categories of aliases that you can define: *host aliases* (`Host_Alias`), *user aliases* (`User_Alias`) and *command aliases* (`Cmnd_Alias`). Here is an example:

```
# Host alias specification

Host_Alias SERVERS = 192.168.1.7, server1, server2

# User alias specification

User_Alias REGULAR_USERS = john, mary, alex

User_Alias PRIVILEGED_USERS = mimi

User_Alias ADMINS = carol, %sudo, PRIVILEGED_USERS, !REGULAR_USERS

# Cmnd alias specification

Cmnd_Alias SERVICES = /usr/bin/systemctl *

# User privilege specification
root    ALL=(ALL:ALL) ALL
ADMINS  SERVERS=SERVICES

# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
```

Considering this sample `sudoers` file, let us explain the three types of aliases in a bit more detail:



## Host aliases

They include a comma-separated list of hostnames, IP addresses, as well as networks and netgroups (preceded by +). Netmasks can be also specified. The `SERVERS` host alias includes an IP address and two hostnames:

```
Host_Alias SERVERS = 192.168.1.7, server1, server2
```

## User aliases

They include a comma-separated list of users specified as usernames, groups (preceded by %) and netgroups (preceded by +). You can exclude particular users with !. The `ADMINS` user alias—for example—includes user `carol`, the members of the `sudo` group and those members of the `PRIVILEGE_USERS` user alias that do not belong in the `REGULAR_USERS` user alias:

```
User_Alias ADMINS = carol, %sudo, PRIVILEGED_USERS, !REGULAR_USERS
```

## Command aliases

They include a comma-separated list of commands and directories. If a directory is specified, any file in that directory will be included—subdirectories will be ignored, though. The `SERVICES` command alias includes a single command with all its subcommands—as specified by the asterisk (\*):

```
Cmnd_Alias SERVICES = /usr/bin/systemctl *
```

As a result of the alias specifications, the line `ADMINS SERVERS=SERVICES` under the `User privilege` specification section translates as: all users belonging in `ADMINS` can use `sudo` to run any command in `SERVICES` on any server in `SERVERS`.

### NOTE

There is a fourth type of alias that you can include in `/etc/sudoers`: *runas aliases* (`Runas_Alias`). They are very similar to user aliases, but allow you to specify users by their *user ID* (UID). This feature might be convenient in some scenarios.

## Guided Exercises

1. Complete the following table regarding special permissions:

Special Permission	Numeric Representation	Symbolic Representation	Find files with <i>only</i> that Permission set
SUID			
SGID			

2. Displaying files with *only* the SUID or SGID bit set is normally not very practical. Perform the following tasks to prove that your searches can be more productive:

- Find all files with the SUID (and other permissions) set in `/usr/bin`:

- Find all files with the SGID (and other permissions) set in `/usr/bin`:

- Find all files with either the SUID or the SGID set in `/usr/bin`:

3. `chage` lets you change a user's password expiry information. As root, complete the following table by providing the correct commands on user `mary`:

Meaning	<code>chage</code> Commands
Make password will be valid for 365 days.	
Make user change password on next login.	
Set minimum number of days between password changes to 1.	
Disable password expiration.	
Enable user to change her password at any time.	
Set warning period to 7 days and account expiration date to August, 20th 2050.	
Print user's current password expiry information.	

4. Complete the following table with the appropriate network utility:

Action	Command(s)
Show network files for host 192.168.1.55 on port 22 using <code>lsof</code> .	
Show processes accessing the default port of the Apache web server on your machine with <code>fuser</code> .	
List all listening <code>udp</code> sockets on your machine using <code>netstat</code> .	
Scan ports 80 through 443 on host 192.168.1.55 using <code>nmap</code> .	

5. Perform the following tasks concerning *resident set size (RSS)* and `ulimit` as a regular user:

- Display *soft* limits on the *maximum RSS*:

- Display *hard* limits on the *maximum RSS*:

- Set the *soft* limits on the *maximum RSS* to 5,000 kilobytes:

- Set the *hard* limits on the *maximum RSS* to 10,000 kilobytes:

- Finally, try to increase the *hard* limit on the *maximum RSS* up to 15,000 kilobytes. Can you do it? Why?

6. Consider the following `last` command output line and answer the questions:

```
carol pts/0 192.168.1.4 Sun May 31 14:16 - 14:22 (00:06)
```

- Was `carol` connected from a remote host? Why?

- How long did `carol`'s session last?

- Was `carol` connected through a true classic text-based terminal? Why?

7. Consider the following excerpt from `/etc/sudoers` and answer the question below.

```
# Host alias specification

Host_Alias SERVERS = 192.168.1.7, server1, server2

# User alias specification

User_Alias REGULAR_USERS = john, mary, alex

User_Alias PRIVILEGED_USERS = mimi

User_Alias ADMINS = carol, %sudo, PRIVILEGED_USERS, !REGULAR_USERS

# Cmnd alias specification

Cmnd_Alias WEB_SERVER_STATUS = /usr/bin/systemctl status apache2

# User privilege specification
root    ALL=(ALL:ALL) ALL
ADMINS  SERVERS=WEB_SERVER_STATUS

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

Can `alex` check the status of the Apache Web Server on any host? Why?

## Explorational Exercises

1. Apart from the SUID and SGID, there is a third special permission: the *sticky bit*. At present it is mostly used on directories such as `/tmp` to prevent regular users from deleting or moving files other than their own. Do the following tasks:
  - Set the *sticky bit* on `~/temporal`:
  - Find directories with the *sticky bit* (and any other permissions) set on your home directory:
  - Unset the *sticky bit* on `~/temporal`:
2. When a user's password is locked via `passwd -l username` or `usermod -L username`, how can you tell by looking into `/etc/shadow`?
3. What is the `usermod` command counterpart to `chage -E date username` or `chage --expiredate date username`?
4. Provide two different `nmap` commands to scan all 65535 ports on localhost:

# Summary

In this lesson you have learned how to perform a number of security administration tasks. The following topics have been covered:

- Finding files with the special `SUID` and `SGID` permissions set.
- Setting and changing users' passwords and dealing with password aging information.
- Using a number of network utilities to discover open ports on hosts/networks.
- Setting up limits on system resources.
- Checking on users that have logged in to the system or who are currently logged in.
- Basic `sudo` usage and configuration (through the `/etc/sudoers` file).

Commands and files discussed in this lesson:

## `find`

Search for files in a directory hierarchy.

## `passwd`

Change user password.

## `chmod`

Change file mode bits.

## `chage`

Change user password expiry information.

## `lsof`

List open files.

## `fuser`

Identify processes using files or sockets.

## `netstat`

Print network connections.

## `nmap`

Network exploration tool and port scanner.

**ulimit**

Get and set user limits.

**/etc/security/limits.conf**

Configuration file to apply restrictions on users.

**last**

Print a listing of last logged in users.

**lastb**

Print a listing of bad login attempts.

**/var/log/wtmp**

Database of user logins.

**who**

Show who is logged on.

**w**

Show who is logged on and what they are doing.

**su**

Change user or become superuser.

**sudo**

Execute a command as another user (including the superuser).

**/etc/sudoers**

Default configuration file for `sudo` security policy.

## Answers to Guided Exercises

1. Complete the following table regarding special permissions:

Special Permission	Numeric Representation	Symbolic Representation	Find files with <i>only</i> that Permission set
SUID	4000	s,S	find -perm 4000, find -perm u+s
SGID	2000	s,S	find -perm 2000, find -perm g+s

2. Displaying files with *only* the SUID or SGID bit set is normally not very practical. Perform the following tasks to prove that your searches can be more productive:

- Find all files with the SUID (and other permissions) set in /usr/bin:

```
find /usr/bin -perm -4000 or find /usr/bin -perm -u+s
```

- Find all files with the SGID (and other permissions) set in /usr/bin:

```
find /usr/bin -perm -2000 or find /usr/bin -perm -g+s
```

- Find all files with either the SUID or the SGID set in /usr/bin:

```
find /usr/bin -perm /6000
```

3. chage lets you change a user's password expiry information. As root, complete the following table by providing the correct commands on user mary:

Meaning	chage Commands
Make password will be valid for 365 days.	chage -M 365 mary, chage --maxdays 365 mary
Make user change password on next login.	chage -d 0 mary, chage --lastday 0 mary
Set minimum number of days between password changes to 1.	chage -m 1 mary, chage --mindays 1 mary
Disable password expiration.	chage -M 99999 mary, chage --maxdays 99999 mary



Meaning	chage Commands
Enable user to change her password at any time.	<code>chage -m 0 mary, chage --mindays 0 mary</code>
Set warning period to 7 days and account expiration date to August, 20th 2050.	<code>chage -W 7 -E 2050-08-20 mary, chage --warndays 7 --expiredate 2050-08-20 mary</code>
Print user's current password expiry information.	<code>chage -l mary, chage --list mary</code>

4. Complete the following table with the appropriate network utility:

Action	Command(s)
Show network files for host 192.168.1.55 on port 22 using <code>lsof</code> .	<code>lsof -i@192.168.1.55:22</code>
Show processes accessing the default port of the Apache web server on your machine with <code>fuser</code> .	<code>fuser -vn tcp 80, fuser --verbose --namespace tcp 80</code>
List all listening <code>udp</code> sockets on your machine using <code>netstat</code> .	<code>netstat -lu, netstat --listening --udp</code>
Scan ports 80 through 443 on host 192.168.1.55 using <code>nmap</code> .	<code>nmap -p 80-443 192.168.1.55</code>

5. Perform the following tasks concerning *resident set size (RSS)* and `ulimit` as a regular user:

- Display *soft* limits on the *maximum RSS*:

```
ulimit -m, ulimit -Sm
```

- Display *hard* limits on the *maximum RSS*:

```
ulimit -Hm
```

- Set the *soft* limits on the *maximum RSS* to 5,000 kilobytes:

```
ulimit -Sm 5000
```

- Set the *hard* limits on the *maximum RSS* to 10,000 kilobytes:

```
ulimit -Hm 10000
```

- Finally, try to increase the *hard* limit on the *maximum RSS* up to 15,000 kilobytes. Can you do it? Why?

No. Once set, regular users cannot increase hard limits.

6. Consider the following `last` command output line and answer the questions:

```
carol pts/0 192.168.1.4 Sun May 31 14:16 - 14:22 (00:06)
```

- Was `carol` connected from a remote host? Why?

Yes, the IP address of the remote host is in the third column.

- How long did `carol`'s session last?

Six minutes (as shown in the last column).

- Was `carol` connected through a true classic text-based terminal? Why?

No, `pts/0` in the second column indicates the connection was made through a graphical terminal emulator (aka *Pseudo Terminal Slave*).

7. Consider the following excerpt from `/etc/sudoers` and answer the question below.

```
# Host alias specification

Host_Alias SERVERS = 192.168.1.7, server1, server2

# User alias specification

User_Alias REGULAR_USERS = john, mary, alex

User_Alias PRIVILEGED_USERS = mimi

User_Alias ADMINS = carol, %sudo, PRIVILEGED_USERS, !REGULAR_USERS

# Cmnd alias specification

Cmnd_Alias WEB_SERVER_STATUS = /usr/bin/systemctl status apache2

# User privilege specification
root    ALL=(ALL:ALL) ALL
ADMINS  SERVERS=WEB_SERVER_STATUS
```

```
# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
```

Can alex check the status of the Apache Web Server on any host? Why?

No, as he is a member of REGULAR\_USERS and that group of users is excluded from ADMINS; the only users (apart from carol, members of the sudo group and root) that can run systemctl status apache2 on the SERVERS.

## Answers to Explorational Exercises

1. Apart from the SUID and SGID, there is a third special permission: the *Sticky bit*. At present it is mostly used on directories such as `/tmp` to prevent regular users from deleting or moving files other than their own. Do the following tasks:

- Set the *sticky bit* on `~/temporal`:

```
chmod +t temporal, chmod 1755 temporal
```

- Find directories with the *sticky bit* (and any other permissions) set on your home directory:

```
find ~ -perm -1000, find ~ -perm /1000
```

- Unset the *sticky bit* on `~/temporal`:

```
chmod -t temporal, chmod 0755 temporal
```

2. When a user's password is locked via `passwd -l username` or `usermod -L username`, how can you tell by looking into `/etc/shadow`?

An exclamation mark will appear in the second field, right after the login name of the affected user (e.g.: `mary: !$6$g0g9xJgv...`).

3. What is the `usermod` command counterpart to `chage -E date username` or `chage --expiredate date username`?

```
usermod -e date username, usermod --expiredate date username
```

4. Provide two different `nmap` commands to scan all 65535 ports on localhost:

```
nmap -p 1-65535 localhost and nmap -p- localhost
```



## 110.2 Setup host security

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 110.2](#)

### Weight

3

### Key knowledge areas

- Awareness of shadow passwords and how they work.
- Turn off network services not in use.
- Understand the role of TCP wrappers.

### Partial list of the used files, terms and utilities

- `/etc/nologin`
- `/etc/passwd`
- `/etc/shadow`
- `/etc/xinetd.d/`
- `/etc/xinetd.conf`
- `systemd.socket`
- `/etc/inittab`
- `/etc/init.d/`
- `/etc/hosts.allow`
- `/etc/hosts.deny`



## 110.2 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	110 Security
<b>Objective:</b>	110.2 Setup host security
<b>Lesson:</b>	1 of 1

### Introduction

This chapter explains four basic ways to improve host security:

1. Some basic commands and configuration settings to improve authentication security with shadow passwords.
2. How to use superdaemons to listen for incoming network connections.
3. Checking network services for unnecessary daemons.
4. TCP wrappers as sort of a simple firewall.

### Improve Authentication Security with Shadow Passwords

The basic components of a user's account data are stored in the file `/etc/passwd`. This file contains seven fields: login name, userid, groupid, password, comment (aka GECOS), home directory location and finally the default shell. A simple way to remember the order of these fields is to think about the process of a user logging in: first you enter a login name, secondly the system will map this into a userid (uid) and thirdly into a groupid (gid). The fourth step asks for a password, the fifth looks up the comment, the sixth enters the users home directory and the

seventh step sets the default shell.

Although in modern systems the password is no longer stored in the file `/etc/passwd`. Instead the password field contains a lower-case `x` only. The file `/etc/passwd` has to be readable by all users. Therefore it is not a good idea to store passwords there. The `x` indicates the encrypted (hashed) password is actually stored in the `/etc/shadow` file instead. This file must not be readable to all users.

Password settings are configured with the commands `passwd` and `chage`. Both commands will change the entry for user `emma` in the file `/etc/shadow`. As a superuser you can set the password for user `emma` with the following command:

```
$ sudo passwd emma
New password:
Retype new password:
passwd: password updated successfully
```

You will then be prompted twice to confirm the new password.

To list the password expiration time and other password expiration settings for user `emma` use:

```
$ sudo chage -l emma
Last password change           : Apr 27, 2020
Password expires               : never
Password inactive              : never
Account expires                : never
Minimum number of days between password change : 0
Maximum number of days between password change  : 99999
Number of days of warning before password expires : 7
```

To prevent the user `emma` from logging into the system the superuser may set a password expiration date that precedes the current date. For example, if today's date were 2020-03-27, you could expire the user's password by using an older date:

```
$ sudo chage -E 2020-03-26 emma
```

Alternatively, the superuser may use:

```
$ sudo passwd -l emma
```

to lock the account temporarily via the `-l` option for `passwd`. To test the effects of these changes, attempt to login as the `emma` account:

```
$ sudo login emma
Password:
Your account has expired; please contact your system administrator

Authentication failure
```

To prevent all users except the root user from logging into the system temporarily, the superuser may create a file named `/etc/nologin`. This file may contain a message to the users notifying them as to why they can not login (for example, system maintenance notifications). For details see `man 5 nologin`. Note there is also a command `nologin` which can be used to prevent a login when set as the default shell for a user. For example:

```
$ sudo usermod -s /sbin/nologin emma
```

See `man 8 nologin` for more details.

## How to Use a Superdaemon to Listen for Incoming Network Connections

Network services such as web servers, email servers and print servers usually run as a standalone service listening on a dedicated network port. All of these standalone services are running side-by-side. On classic Sys-V-init based system each of these services can be controlled by the command `service`. On current `systemd` based systems you would use `systemctl` to manage the service.

In former times the availability of computer resources had been much smaller. To run many services in standalone mode in tandem was not a good option. Instead a so-called superdaemon had been used to listen for incoming network connections and start the appropriate service on demand. This method of building a network connection took a little more time. Well known superdaemons are `inetd` and `xinetd`. On current systems based on `systemd` the `systemd.socket` unit can be used in a similar way. In this section we will use `xinetd` to intercept connections to the `sshd` daemon and start this daemon on request to demonstrate how the superdaemon was used.

Before configuring the `xinetd` service some preparation is necessary. It does not matter whether you use a Debian or Red Hat based system. Though these explanations have been tested with



Debian/GNU Linux 9.9 they should work on any current Linux system featuring systemd, without any significant changes. First make sure the packages `openssh-server` and `xinetd` are installed. Now verify that the SSH service works with:

```
$ systemctl status sshd
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2020-04-27 09:33:48 EDT; 3h 11min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 430 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 460 (sshd)
    Tasks: 1 (limit: 1119)
   Memory: 5.3M
   CGroup: /system.slice/ssh.service
           └─460 /usr/sbin/sshd -D
```

Also check that the SSH service is listening on its standard network port of 22:

```
# lsof -i :22
COMMAND  PID USER  FD  TYPE  DEVICE  SIZE/OFF  NODE  NAME
sshd     1194 root   3u  IPv4 16053268      0t0  TCP  *:ssh (LISTEN)
sshd     1194 root   4u  IPv6 16053270      0t0  TCP  *:ssh (LISTEN)
```

Finally stop the SSH service with:

```
$ sudo systemctl stop sshd.service
```

In the case that you would want to make this change permanent and to survive reboot use `systemctl disable sshd.service`.

Now you can create the `xinetd` configuration file `/etc/xinetd.d/ssh` with some basic settings:

```
service ssh
{
    disable      = no
    socket_type  = stream
    protocol    = tcp
    wait        = no
    user        = root
```

```
server      = /usr/sbin/sshd
server_args = -i
flags       = IPv4
interface   = 192.168.178.1
}
```

Restart the xinetd service with:

```
$ sudo systemctl restart xinetd.service
```

Check which service is listening now for incoming SSH connections.

```
$ sudo lsof -i :22
COMMAND  PID USER  FD  TYPE  DEVICE  SIZE/OFF  NODE  NAME
xinetd   24098 root   5u   IPv4  7345141      0t0   TCP  192.168.178.1:ssh (LISTEN)
```

We can see that the xinetd service has taken over control for the access of port 22.

Here are some more details about the xinetd configuration. The main configuration file is `/etc/xinetd.conf`:

```
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/

defaults
{
    # Please note that you need a log_type line to be able to use log_on_success
    # and log_on_failure. The default is the following :
    # log_type = SYSLOG daemon info
}

includedir /etc/xinetd.d
```

Besides the default settings, there is only one directive to set an include directory. In this directory you can set up a single configuration file for each service you want to have handled by xinetd. We have done this above for the SSH service and named the file `/etc/xinetd.d/ssh`. The names of the configuration files can be chosen arbitrarily, except filenames containing a dot (.) or

ending with a tilde (~). But it is widespread practice to name the file after the service you want to configure.

Some configuration files in the `/etc/xinet.d/` directory are provided by the distribution already:

```
$ ls -l /etc/xinetd.d
total 52
-rw-r--r-- 1 root root 640 Feb  5  2018 chargen
-rw-r--r-- 1 root root 313 Feb  5  2018 chargen-udp
-rw-r--r-- 1 root root 502 Apr 11 10:18 daytime
-rw-r--r-- 1 root root 313 Feb  5  2018 daytime-udp
-rw-r--r-- 1 root root 391 Feb  5  2018 discard
-rw-r--r-- 1 root root 312 Feb  5  2018 discard-udp
-rw-r--r-- 1 root root 422 Feb  5  2018 echo
-rw-r--r-- 1 root root 304 Feb  5  2018 echo-udp
-rw-r--r-- 1 root root 312 Feb  5  2018 servers
-rw-r--r-- 1 root root 314 Feb  5  2018 services
-rw-r--r-- 1 root root 569 Feb  5  2018 time
-rw-r--r-- 1 root root 313 Feb  5  2018 time-udp
```

These files can be used as templates in the rare case you have to use some legacy services such as `daytime`, a very early implementation of a time server. All of these template file contain the directive `disable = yes`.

Here are some more details about the directives used in the `/etc/xinetd.d/ssh` example file for `ssh` above.

```
service ssh
{
    disable      = no
    socket_type  = stream
    protocol    = tcp
    wait        = no
    user        = root
    server      = /usr/sbin/sshd
    server_args = -i
    flags       = IPv4
    interface   = 192.168.178.1
}
```

**service**

Lists the service `xinetd` has to control. You may use either a port number, such as 22, or the name mapped to the port number in `/etc/services` e.g. `ssh`.

```
{
```

Detailed settings begin with an opening curly bracket.

**disable**

To activate these settings, set this to `no`. If you want to disable the setting temporarily you may set it to `yes`.

**socket\_type**

You can choose `stream` for TCP sockets or `dgram` for UDP sockets and more.

**protocol**

Choose either TCP or UDP.

**wait**

For TCP connections this is set to `no` usually.

**user**

The service started in this line will be owned by this user.

**server**

Full path to the service which should be started by `xinetd`.

**server\_args**

You can add options for the service here. If started by a super-server many services require a special option. For SSH this would be the `-i` option.

**flags**

You may choose IPv4, IPv6 and others.

**interface**

The network interface which `xinetd` has to control. Note: you may also choose the `bind` directive, which is just a synonym for `interface`.

```
}
```

Finish with a closing curly bracket.

The successors of services started by the super-server `xinetd` are `systemd` socket units. Setting up

a systemd socket unit is very straight forward and easy, because there is a predefined systemd socket unit for SSH available already. Make sure that the `xinetd` and SSH services are not running.

Now you just have to start the SSH socket unit:

```
$ sudo systemctl start ssh.socket
```

To check which service is now listening on port 22 we use `lsof` again. Note here the option `-P` has been used to show the port number instead of the service name in the output:

```
$ sudo lsof -i :22 -P
COMMAND PID USER  FD  TYPE  DEVICE  SIZE/OFF  NODE  NAME
systemd   1 root   57u IPv6  14730112      0t0  TCP  *:22 (LISTEN)
```

To make this session complete, you should try to login into your server with an SSH client of your choice.

#### TIP

In case `systemctl start ssh.socket` does not work with your distribution, try `systemctl start sshd.socket`.

## Checking Services for Unnecessary Daemons

For security reasons, as well as to control system resources, it is important to have an overview of what services are running. Unnecessary and unused services should be disabled. For example in case you do not need to distribute web pages there is no need to run a web server such as Apache or nginx.

On Sys-V-init based systems you may check the status of all services with the following:

```
$ sudo service --status-all
```

Verify whether each of the services listed from the output of the command are necessary and disable all unnecessary services with (for Debian-based systems):

```
$ sudo update-rc.d SERVICE-NAME remove
```

Or on Red Hat-based systems you would use:

```
$ sudo chkconfig SERVICE-NAME off
```

On modern systemd based systems we can use the following to list all running services:

```
$ systemctl list-units --state active --type service
```

You would then disable each unnecessary service unit with:

```
$ sudo systemctl disable UNIT --now
```

This command will stop the service and remove it from the list of services, so as to prevent it from starting up at next system boot.

Additionally to get a survey of listening network services you may use `netstat` on older systems (provided that you have the `net-tools` package installed):

```
$ netstat -ltu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:ssh             0.0.0.0:*                LISTEN
tcp      0      0 localhost:mysql        0.0.0.0:*                LISTEN
tcp6     0      0 [::]:http              [::]:*                  LISTEN
tcp6     0      0 [::]:ssh               [::]:*                  LISTEN
udp      0      0 0.0.0.0:bootpc        0.0.0.0:*
```

Or on modern systems you could use the equivalent command `ss` (for “socket services”):

```
$ ss -ltu
Netid      State      Recv-Q     Send-Q     Local Address:Port      Peer
Address:Port
udp        UNCONN     0           0           0.0.0.0:bootpc
0.0.0.0:*
tcp        LISTEN     0           128          0.0.0.0:ssh
0.0.0.0:*
tcp        LISTEN     0           80          127.0.0.1:mysql
0.0.0.0:*
tcp        LISTEN     0           128          *:http
*:*
tcp        LISTEN     0           128          [::]:ssh
```

```
[::]:*
```

## TCP Wrappers as Sort of a Simple Firewall

In times when no firewalls were available for Linux, TCP wrappers had been used to secure network connections into a host. Nowadays many programs do not obey TCP wrappers anymore. In recent Red Hat (e.g. Fedora 29) based distributions TCP wrappers support has been removed completely. In order to support legacy Linux systems that still use TCP wrappers, it is helpful to have some basic knowledge about this particular technology.

We will once again use the SSH service as a basic example. The service on our example host should be reachable from the local network only. First we check whether the SSH daemon uses the `libwrap` library which offers TCP wrappers support:

```
$ ldd /usr/sbin/sshd | grep "libwrap"
libwrap.so.0 => /lib/x86_64-linux-gnu/libwrap.so.0 (0x00007f91dbec0000)
```

Now we add the following line in the file `/etc/hosts.deny`:

```
sshd: ALL
```

Finally we configure an exception in the file `/etc/hosts.allow` for SSH connections from the local network:

```
sshd: LOCAL
```

The changes take effect immediately, there is no need to restart any service. You may check this with the `ssh` client.

## Guided Exercises

1. How can the previously locked account `emma` be unlocked?

2. Previously the account `emma` had an expiration date set. How can the expiration date get set to `never`?

3. Imagine the CUPS printing service handling print jobs is not needed on your server. How can you disable the service permanently? How can you check the appropriate port is not active anymore?

4. You have installed the `nginx` web server. How can you check whether `nginx` supports TCP wrappers?



## Explorational Exercises

1. Check out whether the existence of the `/etc/nologin` file prevents the login of the user `root`?
2. Does the existence of the `/etc/nologin` file prevent passwordless logins with SSH keys?
3. What happens on login, when the file `/etc/nologin` contains this line of text `login currently is not possible only`?
4. May an ordinary user `emma` obtain information about the user `root` contained in the file `/etc/passwd` e.g. with the command `grep root /etc/passwd`?
5. May an ordinary user `emma` retrieve information about her own hashed password contained in the file `/etc/shadow` e.g. with the command `grep emma /etc/shadow`?
6. What steps have to be taken to enable and check the ancient daytime service to be handled by `xinetd`? Note this is just an explorational exercise don't do this in a production environment.

# Summary

In this lesson you learned:

1. In which file passwords are stored as well as some password security settings e.g. expiration time.
2. The purpose of the superdaemon `xinetd` and how to get it running and start the `sshd` service on demand.
3. To check which network services are running and how to disable unnecessary services.
4. Use TCP wrappers as sort of a simple firewall.

Commands used in the lab and the exercises:

## **chage**

Change the age of a user's password.

## **chkconfig**

A classic command initially used on Red Hat based systems to set whether a service would start at boot time or not.

## **netstat**

A classic utility (now in the `net-tools` package) that will display daemons that access network ports on a system and their usage.

## **nologin**

A command that can be used in place of a user's shell to prevent them from logging in.

## **passwd**

Used to create or change a user's password.

## **service**

Older method of controlling a daemon's status, such as stopping or starting a service.

## **ss**

The modern equivalent to `netstat`, but also displays more information about various sockets in use on the system.

## **systemctl**

The system control command used to control various aspects of services and sockets on a

computer using systemd.

### **update-rc.d**

A classic command similar to `chkconfig` that enables or disables a system to start at boot time on Debian based distributions.

### **xinetd**

A superdaemon that can control access to a network service on demand, thus leaving a service inactive until it is actually called upon to perform some task.

## Answers to Guided Exercises

1. How can the previously locked account `emma` be unlocked?

The superuser can run `passwd -u emma` to unlock the account.

2. Previously the account `emma` had an expiration date set. How can the expiration date get set to never?

The superuser may use `chage -E -1 emma` to set the expiration date to never. This setting may be checked with `chage -l emma`.

3. Imagine the CUPS printing service handling print jobs is not needed on your server. How can you disable the service permanently? How can you check the appropriate port is not active anymore?

As superuser issue

```
systemctl disable cups.service --now
```

Now you can check

```
netstat -l | grep ":ipp "` or `ss -l | grep ":ipp "
```

4. You have installed the `nginx` web server. How can you check whether `nginx` supports TCP wrappers?

The command

```
ldd /usr/sbin/nginx | grep "libwrap"
```

will show an entry in case `nginx` supports TCP wrappers.

## Answers to Explorational Exercises

1. Check out whether the existence of the `/etc/nologin` file prevents the login of the user `root`?

User `root` is still able to login.

2. Does the existence of the `/etc/nologin` file prevent passwordless logins with SSH keys?

Yes, also passwordless logins are prevented.

3. What happens on login, when the file `/etc/nologin` contains this line of text `login currently is not possible only`?

The message `login currently is not possible` will be shown, and a login is prevented.

4. May an ordinary user `emma` obtain information about the user `root` contained in the file `/etc/passwd` e.g. with the command `grep root /etc/passwd`?

Yes, because all users have read permission for this file.

5. May an ordinary user `emma` retrieve information about her own hashed password contained in the file `/etc/shadow` e.g. with the command `grep -i emma /etc/shadow`?

No, because ordinary users have no read permission for this file.

6. What steps have to be taken to enable and check the ancient daytime service to be handled by `xinetd`? Note this is just an explorational exercise don't do this in a production environment.

First change the file `/etc/xinetd.d/daytime` and set the directive `disable = no`. Second restart the `xinetd` service `systemctl restart xinetd.service` (or `service xinetd restart` on systems with Sys-V-Init). Now you can check whether it works `nc localhost daytime`. Instead of `nc` you may also use `netcat`.



## 110.3 Securing data with encryption

### Reference to LPI objectives

[LPIC-1 version 5.0, Exam 102, Objective 110.3](#)

### Weight

4

### Key knowledge areas

- Perform basic OpenSSH 2 client configuration and usage.
- Understand the role of OpenSSH 2 server host keys.
- Perform basic GnuPG configuration, usage and revocation.
- Use GPG to encrypt, decrypt, sign and verify files.
- Understand SSH port tunnels (including X11 tunnels).

### Partial list of the used files, terms and utilities

- `ssh`
- `ssh-keygen`
- `ssh-agent`
- `ssh-add`
- `~/.ssh/id_rsa` and `id_rsa.pub`
- `~/.ssh/id_dsa` and `id_dsa.pub`
- `~/.ssh/id_ecdsa` and `id_ecdsa.pub`
- `~/.ssh/id_ed25519` and `id_ed25519.pub`
- `/etc/ssh/ssh_host_rsa_key` and `ssh_host_rsa_key.pub`

- `/etc/ssh/ssh_host_dsa_key` and `ssh_host_dsa_key.pub`
- `/etc/ssh/ssh_host_ecdsa_key` and `ssh_host_ecdsa_key.pub`
- `/etc/ssh/ssh_host_ed25519_key` and `ssh_host_ed25519_key.pub`
- `~/.ssh/authorized_keys`
- `ssh_known_hosts`
- `gpg`
- `gpg-agent`
- `~/.gnupg/`



## 110.3 Lesson 1

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	110 Security
<b>Objective:</b>	110.3 Securing data with encryption
<b>Lesson:</b>	1 of 2

### Introduction

Securing data with encryption is of paramount importance in many aspects of today's system administration—even more so when it comes to accessing systems remotely. As opposed to insecure solutions such as *telnet*, *rlogin* or *FTP*, the *SSH (Secure Shell)* protocol was designed with security in mind. Using public key cryptography, it authenticates both hosts and users and encrypts all subsequent information exchange. Furthermore, SSH can be used to establish *port tunnels*, which—amongst other things—allows for a non-encrypted protocol to transmit data over an encrypted SSH connection. The current, recommended version of the SSH protocol is 2.0. *OpenSSH* is a free and open source implementation of the SSH protocol.

This lesson will cover basic *OpenSSH* client configuration as well as the role of *OpenSSH* server host keys. The concept of SSH port tunnels will also be discussed. We will be using two machines with the following setup:



Machine Role	OS	IP Address	Hostname	User
Client	Debian GNU/Linux 10 (buster)	192.168.1.55	debian	carol
Server	openSUSE Leap 15.1	192.168.1.77	halof	ina

## Basic OpenSSH Client Configuration and Usage

Although the OpenSSH server and client come in separate packages, you can normally install a metapackage that will provide both at once. To establish a remote session with the SSH server you use the `ssh` command, specifying the user you want to connect as on the remote machine and the remote machine's IP address or hostname. The first time you connect to a remote host you will receive a message like this:

```
carol@debian:~$ ssh ina@192.168.1.77
The authenticity of host '192.168.1.77 (192.168.1.77)' can't be established.
ECDSA key fingerprint is SHA256:5JF7anupYipByCQm2BPvDHRVFJJixeslmp2NwATYI.
Are you sure you want to continue connecting (yes/no)?
```

After typing `yes` and hitting Enter, you will be asked for the remote user's password. If successfully entered, you will be shown a warning message and then logged in to the remote host:

```
Warning: Permanently added '192.168.1.77' (ECDSA) to the list of known hosts.
Password:
Last login: Sat Jun 20 10:52:45 2020 from 192.168.1.4
Have a lot of fun...
ina@halof:~>
```

The messages are quite self-explanatory: as it was the first time that you established a connection to the `192.168.1.77` remote server, its authenticity could not be checked against any database. Thus, the remote server provided an ECDSA key fingerprint of its public key (using the SHA256 hash function). Once you accepted the connection, the public key of the remote server was added to the *known hosts* database, thus enabling the authentication of the server for future connections. This list of *known hosts*' public keys is kept in the file `known_hosts` which lives in `~/ .ssh`:

```
ina@halof:~> exit
logout
```

```
Connection to 192.168.1.77 closed.
carol@debian:~$ ls .ssh/
known_hosts
```

Both `.ssh` and `known_hosts` were created after the first remote connection was established. `~/ .ssh` is the default directory for user-specific configuration and authentication information.

**NOTE** You can also use `ssh` to just execute a single command on the remote host and then come back to your local terminal (e.g.: running `ssh ina@halof ls`).

If you are using the same user on both the local and remote hosts, there is no need to specify the username when establishing the SSH connection. For instance, if you are logged in as user `carol` on `debian` and wanted to connect to `halof` also as user `carol`, you would simply type `ssh 192.168.1.77` or `ssh halof` (if the name can be resolved):

```
carol@debian:~$ ssh halof
Password:
Last login: Wed Jul 1 23:45:02 2020 from 192.168.1.55
Have a lot of fun...
carol@halof:~>
```

Now suppose you establish a new remote connection with a host that happens to have the same IP address as `halof` (a common thing if you use DHCP in your LAN). You will be warned about the possibility of a *man-in-the-middle* attack:

```
carol@debian:~$ ssh john@192.168.1.77
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:KH4q3vP6C7e0SEjyG8WlZ9fVlf+jmWJ5139RBxBh3TY.
Please contact your system administrator.
Add correct host key in /home/carol/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/carol/.ssh/known_hosts:1
  remove with:
  ssh-keygen -f "/home/carol/.ssh/known_hosts" -R "192.168.1.77"
ECDSA host key for 192.168.1.77 has changed and you have requested strict checking.
Host key verification failed.
```

Since you are not dealing with a *man-in-the-middle* attack, you can safely add the public key fingerprint of the new host to `~/.ssh/known_hosts`. As the message indicates, you can first use the command `ssh-keygen -f "/home/carol/.ssh/known_hosts" -R "192.168.1.77"` to remove the *offending* key (alternatively, you can go for `ssh-keygen -R 192.168.1.77` to delete all keys belonging to `192.168.1.77` from `~/.ssh/known_hosts`). Then, you will be able to establish a connection to the new host.

## Key-Based Logins

You can set up your SSH client to not provide any passwords at login but use public keys instead. This is the preferred method of connecting to a remote server via SSH, as it is far more secure. The first thing you have to do is create a key pair on the client machine. To do this, you will use `ssh-keygen` with the `-t` option specifying the type of encryption you want (*Elliptic Curve Digital Signature Algorithm* in our case). Then, you will be asked for the path to save the key pair (`~/.ssh/` is convenient as well as the default location) and a passphrase. While a passphrase is optional, it is highly recommended to always use one.

```
carol@debian:~/.ssh$ ssh-keygen -t ecdsa
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/carol/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/carol/.ssh/id_ecdsa.
Your public key has been saved in /home/carol/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:tlamD0SaTquPZYdNepwj8XN4xvqmHCbe8g5FKKUfMo8 carol@debian
The key's randomart image is:
+---[ECDSA 256]---+
|   .               |
|  o .              |
| = o o             |
|  B *              |
| E B S o           |
|  o & O             |
|   @ ^ =           |
|  * .@ @.          |
|  o.o+B+o          |
+----[SHA256]-----+
```

### NOTE

When creating the key pair, you can pass `ssh-keygen` the `-b` option to specify the key size in bits (e.g.: `ssh-keygen -t ecdsa -b 521`).

The previous command produced two more files in your `~/ .ssh` directory:

```
carol@debian:~/ .ssh$ ls
id_ecdsa id_ecdsa.pub known_hosts
```

### **id\_ecdsa**

This is your private key.

### **id\_ecdsa.pub**

This is your public key.

#### **NOTE**

In asymmetric cryptography (aka public-key cryptography), the public and private keys are mathematically related to one another in such a way that whatever is encrypted by one can only be decrypted by the other.

The next thing you need to do is add your public key to the `~/ .ssh/authorized_keys` file of the user you want to log in as on the remote host (if the `~/ .ssh` directory does not already exist, you will have to create it first). You can copy your public key into the remote server in a number of ways: using a USB flash drive, through the `scp` command — which will transfer the file over using SSH — or by *catting out* the content of your public key and piping it into `ssh` like so:

```
carol@debian:~/ .ssh$ cat id_ecdsa.pub |ssh ina@192.168.1.77 'cat >> .ssh/authorized_keys'
Password:
```

Once your public key has been added to the `authorized_keys` file on the remote host, you can face two scenarios when trying to establish a new connection:

- If you did not provide a passphrase when creating the key pair, you will be logged in automatically. Although convenient, this method can be insecure depending on the situation:

```
carol@debian:~$ ssh ina@192.168.1.77
Last login: Thu Jun 25 20:31:03 2020 from 192.168.1.55
Have a lot of fun...
ina@halof:~>
```

- If you provided a passphrase when creating the key pair, you will have to enter it on every connection much in the same way as if it was a password. Apart from the public key, this method adds an extra layer of security in the form of a passphrase and can — therefore — be considered more secure. As far as convenience goes — however — it is exactly the same as

having to enter a password every time you establish a connection. If you don't use a passphrase and someone manages to obtain your private SSH key file, they would have access to every server on which your public key is installed.

```
carol@debian:~/.ssh$ ssh ina@192.168.1.77
Enter passphrase for key '/home/carol/.ssh/id_ecdsa':
Last login: Thu Jun 25 20:39:30 2020 from 192.168.1.55
Have a lot of fun...
ina@halof:~>
```

There is a way which combines security and convenience, though: using the *SSH authentication agent* (`ssh-agent`). The authentication agent needs to spawn its own shell and will hold your private keys — for public key authentication — in memory for the remainder of the session. Let us see how it works in a little bit more detail:

1. Use `ssh-agent` to start a new Bash shell:

```
carol@debian:~/.ssh$ ssh-agent /bin/bash
carol@debian:~/.ssh$
```

2. Use the `ssh-add` command to add your private key to a secure area of memory. If you supplied a passphrase when generating the key pair — which is recommended for extra security — you will be asked for it:

```
carol@debian:~/.ssh$ ssh-add
Enter passphrase for /home/carol/.ssh/id_ecdsa:
Identity added: /home/carol/.ssh/id_ecdsa (carol@debian)
```

Once your identity has been added, you can login to any remote server on which your public key is present without having to type your passphrase again. It is common practice on modern desktops to perform this command upon booting your computer, as it will remain in memory until the computer is shutdown (or the key is unloaded manually).

Let us round this section off by listing the four types of public-key algorithms that can be specified with `ssh-keygen`:

## RSA

Named after its creators Ron Rivest, Adi Shamir and Leonard Adleman, it was published in 1977. It is considered secure and still widely used today. Its minimum key size is 1024 bits (default is 2048).

## DSA

The *Digital Signature Algorithm* has proven to be insecure and it was deprecated as of OpenSSH 7.0. DSA keys must be exactly 1024 bits in length.

## ecdsa

The *Elliptic Curve Digital Signature Algorithm* is an improvement on DSA and—therefore—considered more secure. It uses elliptic curve cryptography. ECDSA key length is determined by one of the three possible elliptic curve sizes in bits: 256, 384 or 521.

## ed25519

It is an implementation of EdDSA—*Edwards-curve Digital Signature Algorithm*—that uses the stronger 25519 curve. It is considered the most secure of all. All Ed25519 keys have a fixed length of 256 bits.

### NOTE

If invoked with no `-t` specification, `ssh-keygen` will generate an RSA key pair by default.

## The Role of OpenSSH Server Host Keys

The global configuration directory for OpenSSH lives in the `/etc` directory:

```
halof:~ # tree /etc/ssh
/etc/ssh
├── moduli
├── ssh_config
├── ssh_host_dsa_key
├── ssh_host_dsa_key.pub
├── ssh_host_ecdsa_key
├── ssh_host_ecdsa_key.pub
├── ssh_host_ed25519_key
├── ssh_host_ed25519_key.pub
├── ssh_host_rsa_key
├── ssh_host_rsa_key.pub
└── sshd_config

0 directories, 11 files
```

Apart from `moduli` and the configuration files for the client (`ssh_config`) and the server (`sshd_config`), you will find four key pairs—a key pair for each supported algorithm—that are created when the *OpenSSH* server is installed. As already noted, the server uses these *host keys* to identify itself to clients as required. Their name pattern is as follows:

## Private keys

`ssh_host_` prefix + *algorithm* + key suffix (e.g.: `ssh_host_rsa_key`)

## Public keys (or public key fingerprints)

`ssh_host_` prefix + *algorithm* + key .pub suffix (e.g.: `ssh_host_rsa_key.pub`)

### NOTE

A fingerprint is created by applying a cryptographic hash function to a public key. As fingerprints are shorter than the keys they refer to, they come in handy to simplify certain key management tasks.

The permissions on the files containing the private keys are `0600` or `-rw-----`: only readable and writable by the owner (root). On the other hand, all public key files are also readable by members in the owner group and everybody else (`0644` or `-rw-r--r--`):

```
halof:~ # ls -l /etc/ssh/ssh_host_*
-rw----- 1 root root 1381 Dec 21 20:35 /etc/ssh/ssh_host_dsa_key
-rw-r--r-- 1 root root 605 Dec 21 20:35 /etc/ssh/ssh_host_dsa_key.pub
-rw----- 1 root root 505 Dec 21 20:35 /etc/ssh/ssh_host_ecdsa_key
-rw-r--r-- 1 root root 177 Dec 21 20:35 /etc/ssh/ssh_host_ecdsa_key.pub
-rw----- 1 root root 411 Dec 21 20:35 /etc/ssh/ssh_host_ed25519_key
-rw-r--r-- 1 root root 97 Dec 21 20:35 /etc/ssh/ssh_host_ed25519_key.pub
-rw----- 1 root root 1823 Dec 21 20:35 /etc/ssh/ssh_host_rsa_key
-rw-r--r-- 1 root root 397 Dec 21 20:35 /etc/ssh/ssh_host_rsa_key.pub
```

You can view the fingerprints of the keys by passing `ssh-keygen` the `-l` switch. You must also provide the `-f` to specify the key file path:

```
halof:~ # ssh-keygen -l -f /etc/ssh/ssh_host_ed25519_key
256 SHA256:8cnPrinC49ZHc+/9Ai5pV+1JfZ4WBRZhd3rD0sc2z1A root@halof (ED25519)
halof:~ # ssh-keygen -l -f /etc/ssh/ssh_host_ed25519_key.pub
256 SHA256:8cnPrinC49ZHc+/9Ai5pV+1JfZ4WBRZhd3rD0sc2z1A root@halof (ED25519)
```

To view the key fingerprint as well as its random art, just add the `-v` switch like so:

```
halof:~ # ssh-keygen -lv -f /etc/ssh/ssh_host_ed25519_key.pub
256 SHA256:8cnPrinC49ZHc+/9Ai5pV+1JfZ4WBRZhd3rD0sc2z1A root@halof (ED25519)
+--[ED25519 256]--+
|                   +oo|
|                   .+o.|
|                   ..E.|
|                   + . +.o|
```

```
|      S + + *o|
|      ooo Oo=|
|      . . . =o+.==|
|      = o =oo o=o|
|      o.o +o+..o.+|
+-----[SHA256]-----+
```

## SSH Port Tunnels

OpenSSH features a very powerful forwarding facility whereby traffic on a source port is tunnelled—and encrypted—through an SSH process which then redirects it to a port on a destination host. This mechanism is known as *port tunnelling* or *port forwarding* and has important advantages like the following:

- It allows you to bypass firewalls to access ports on remote hosts.
- It allows access from the outside to a host on your private network.
- It provides encryption for all data exchange.

Roughly speaking, we can differentiate between local and remote port tunnelling.

### Local Port Tunnel

You define a port locally to forward traffic to the destination host through the SSH process which sits in between. The SSH process can run on the local host or on a remote server. For instance, if for some reason you wanted to tunnel a connection to `www.gnu.org` through SSH using port `8585` on your local machine, you would do something like this:

```
carol@debian:~$ ssh -L 8585:www.gnu.org:80 debian
carol@debian's password:
Linux debian 4.19.0-9-amd64 #1 SMP Debian 4.19.118-2 (2020-04-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
(...)
Last login: Sun Jun 28 13:47:27 2020 from 127.0.0.1
```

The explanation is as follows: with the `-L` switch, we specify the local port `8585` to connect to `http` port `80` on `www.gnu.org` using the SSH process running on `debian`—our *localhost*. We could have written `ssh -L 8585:www.gnu.org:80 localhost` with the same effect. If you now use a web browser to go to `http://localhost:8585`, you will be forwarded to `www.gnu.org`. For demonstration purposes, we will use `lynx` (the classic, text-mode web browser):



```

carol@debian:~$ lynx http://localhost:8585
(...)
* Back to Savannah Homepage
  * Not Logged in
  * Login
  * New User
  * This Page
  * Language
  * Clean Reload
  * Printer Version
  * Search
  * _
(...)

```

If you wanted to do the exact same thing but connecting through an SSH process running on halof, you would have proceeded like so:

```

carol@debian:~$ ssh -L 8585:www.gnu.org:80 -Nf ina@192.168.1.77
Enter passphrase for key '/home/carol/.ssh/id_ecdsa':
carol@debian:~$
carol@debian:~$ lynx http://localhost:8585
(...)
* Back to Savannah Homepage
  * Not Logged in
  * Login
  * New User
  * This Page
  * Language
  * Clean Reload
  * Printer Version
  * Search
  * _
(...)

```

It is important that you note three details in the command:

- Thanks to the `-N` option we did not login to halof but did the port forwarding instead.
- The `-f` option told SSH to run in the background.
- We specified user `ina` to do the forwarding: `ina@192.168.1.77`

## Remote Port Tunnel

In remote port tunnelling (or reverse port forwarding) the traffic coming on a port on the remote server is forwarded to the SSH process running on your local host, and from there to the specified port on the destination server (which may also be your local machine). For example, say you wanted to let someone from outside your network access the Apache web server running on your local host through port 8585 of the SSH server running on halof (192.168.1.77). You would proceed with the following command:

```
carol@debian:~$ ssh -R 8585:localhost:80 -Nf ina@192.168.1.77
Enter passphrase for key '/home/carol/.ssh/id_ecdsa':
carol@debian:~$
```

Now anyone who establishes a connection to halof on port 8585 will see Debian's Apache2 default homepage:

```
carol@debian:~$ lynx 192.168.1.77:8585
(...)
                                                                 Apache2 Debian Default
Page: It works (p1 of 3)
  Debian Logo Apache2 Debian Default Page
  It works!

  This is the default welcome page used to test the correct operation of the Apache2 server
after
  installation on Debian systems. If you can read this page, it means that the Apache HTTP
server
  installed at this site is working properly. You should replace this file (located at
/var/www/html/index.html) before continuing to operate your HTTP server.
(...)
```

### NOTE

There is a third, more complex type of port forwarding which is outside the scope of this lesson: *dynamic port forwarding*. Instead of interacting with a single port, this type of forwarding uses various TCP communications across a range of ports.

## X11 Tunnels

Now that you understand port tunnels, let us round this lesson off by discussing X11 tunnelling (also known as *X11 forwarding*). Through an X11 tunnel, the *X Window System* on the remote host is forwarded to your local machine. For that, you just need to pass ssh the `-X` option:

```
carol@debian:~$ ssh -X ina@halof
...
```

You can now launch a graphical application such as the `firefox` web browser with the following result: the app will be run on the remote server, but its display will be forwarded to your local host.

If you start a new SSH session with the `-x` option instead, *X11 forwarding* will be disabled. Try to start `firefox` now and you will get an error such as the following:

```
carol@debian:~$ ssh -x ina@halof
carol@192.168.0.106's password:
(...)
ina@halof:~$ firefox

(firefox-esr:1779): Gtk-WARNING **: 18:45:45.603: Locale not supported by C library.
  Using the fallback 'C' locale.
Error: no DISPLAY environment variable specified
```

**NOTE**

The three configuration directives related to local port forwarding, remote port forwarding and X11 forwarding are `AllowTcpForwarding`, `GatewayPorts` and `X11Forwarding`, respectively. For more information, type `man ssh_config` and/or `man sshd_config`.

## Guided Exercises

1. Logged in as user `sonya` on your client machine, carry out the following SSH tasks on the remote server `halof`:

- Execute the command to list the contents of `~/ .ssh` as user `serena` on the remote host; then return to your local terminal.

- Login as user `serena` on the remote host.

- Login as user `sonya` on the remote host.

- Delete all keys belonging to `halof` from your local `~/ .ssh/known_hosts` file.

- On your client machine, create an `ecdsa` key pair of 256 bits.

- On your client machine, create an `ed25519` key pair of 256 bits.

2. Put the following steps in the right order to establish an SSH connection using the *SSH authentication agent*:

- On the client, start a new Bash shell for the *authentication agent* with `ssh-agent /bin/bash`.
- On the client, create a key pair using `ssh-keygen`.
- On the client, add your private key to a secure area of memory with `ssh-add`.
- Add your client's public key to the `~/ .ssh/authorized_keys` file of the user you want to login as on the remote host.
- If it does not already exist, create `~/ .ssh` for the user you want to login as on the server.
- Connect to the remote server.

The correct order is:

**Step 1:**

<b>Step 2:</b>	
<b>Step 3:</b>	
<b>Step 4:</b>	
<b>Step 5:</b>	
<b>Step 6:</b>	

3. Regarding *port forwarding*, what option and directive is used for the following tunnel types:

<b>Tunnel Type</b>	<b>Option</b>	<b>Directive</b>
Local		
Remote or Reverse		
X		

4. Suppose you type the command `ssh -L 8888:localhost:80 -Nf ina@halof` into the terminal of your client machine. Still on the client machine, you point a browser to `http://localhost:8888`. What will you get?

# Explorational Exercises

## 1. Concerning SSH security directives:

- What directive is used in `/etc/ssh/sshd_config` to enable `root` logins:

- What directive would you use in `/etc/ssh/sshd_config` to specify only a local account to accept SSH connections:

## 2. When using the same user on both the client and the server, what `ssh` command can you use to transfer the client's public key over to the server so that you can login through public key authentication?

## 3. Create two local port tunnels in a single command forwarding local unprivileged ports 8080 and 8585 through remote server `halof` to the websites `www.gnu.org` and `www.melpa.org`, respectively. Use user `ina` on the remote server and do not forget to use the `-Nf` switches:

## Summary

In this lesson we have discussed *OpenSSH 2*, which uses the *Secure Shell* protocol to encrypt communications between server and client. You learned:

- how to login to a remote server.
- how to execute commands remotely.
- how to create key pairs.
- how to establish key-based logins.
- how to use the *authentication agent* for both extra security and convenience.
- the public-key algorithms supported by *OpenSSH*: `RSA`, `DSA`, `ecdsa`, `ed25519`.
- the role of *OpenSSH* host keys.
- how to create port tunnels: local, remote and X.

The following commands were discussed in this lesson:

### **ssh**

Log into or excute commands on a remote machine.

### **ssh-keygen**

Generate, manage and convert authentication keys.

### **ssh-agent**

OpenSSH authentication agent.

### **ssh-add**

Adds private key identities to the authentication agent.

## Answers to Guided Exercises

1. Logged in as user `sonya` on your client machine, carry out the following SSH tasks on the remote server `halof`:

- Execute the command to list the contents of `~/ .ssh` as user `serena` on the remote host; then return to your local terminal.

```
ssh serena@halof ls .ssh
```

- Login as user `serena` on the remote host.

```
ssh serena@halof
```

- Login as user `sonya` on the remote host.

```
ssh halof
```

- Delete all keys belonging to `halof` from your local `~/ .ssh/known_hosts` file.

```
ssh-keygen -R halof
```

- On your client machine, create an `ecdsa` key pair of 256 bits.

```
ssh-keygen -t ecdsa -b 256
```

- On your client machine, create an `ed25519` key pair of 256 bits.

```
ssh-keygen -t ed25519
```

2. Put the following steps in the right order to establish an SSH connection using the *SSH authentication agent*:

- On the client, start a new Bash shell for the *authentication agent* with `ssh-agent /bin/bash`.
- On the client, create a key pair using `ssh-keygen`.
- On the client, add your private key to a secure area of memory with `ssh-add`.



- Add your client's public key to the `~/.ssh/authorized_keys` file of the user you want to login as on the remote host.
- If it does not already exist, create `~/.ssh` for the user you want to login as on the server.
- Connect to the remote server.

The correct order is:

<b>Step 1:</b>	On the client, create a key pair using <code>ssh-keygen</code> .
<b>Step 2:</b>	If it does not already exist, create <code>~/.ssh</code> for the user you want to login as on the server.
<b>Step 3:</b>	Add your client's public key to the <code>~/.ssh/authorized_keys</code> file of the user you want to login as on the remote host.
<b>Step 4:</b>	On the client, start a new Bash shell for the <i>authentication agent</i> with <code>ssh-agent /bin/bash</code> .
<b>Step 5:</b>	On the client, add your private key to a secure area of memory with <code>ssh-add</code> .
<b>Step 6:</b>	Connect to the remote server.

3. Regarding *port forwarding*, what option and directive is used for the following tunnel types:

Tunnel Type	Option	Directive
Local	<code>-L</code>	<code>AllowTcpForwarding</code>
Remote or Reverse	<code>-R</code>	<code>GatewayPorts</code>
X	<code>-X</code>	<code>X11Forwarding</code>

4. Suppose you type the command `ssh -L 8888:localhost:80 -Nf ina@halof` into the terminal of your client machine. Still on the client machine, you point a browser to `http://localhost:8888`. What will you get?

The webserver's homepage of `halof`, as `localhost` is understood from the server's perspective.

# Answers to Explorational Exercises

## 1. Concerning SSH security directives:

- What directive is used in `/etc/ssh/sshd_config` to enable `root` logins:

```
PermitRootLogin
```

- What directive would you use in `/etc/ssh/sshd_config` to specify only a local account to accept SSH connections:

```
AllowUsers
```

## 2. When using the same user on both the client and the server, what `ssh` command can you use to transfer the client's public key over to the server so that you can login through public key authentication?

```
ssh-copy-id
```

## 3. Create two local port tunnels in a single command forwarding local unprivileged ports 8080 and 8585 through remote server `halof` to the websites `www.gnu.org` and `www.meltpa.org`, respectively. Use user `ina` on the remote server and do not forget to use the `-Nf` switches:

```
ssh -L 8080:www.gnu.org:80 -L 8585:www.meltpa.org:80 -Nf ina@halof
```



Linux  
Professional  
Institute

## 110.3 Lesson 2

<b>Certificate:</b>	LPIC-1
<b>Version:</b>	5.0
<b>Topic:</b>	110 Security
<b>Objective:</b>	110.3 Securing data with encryption
<b>Lesson:</b>	2 of 2

### Introduction

In the previous lesson we learned how to use *OpenSSH* to encrypt remote login sessions as well as any other subsequent exchange of information. There may be other scenarios where you may want to encrypt files or email so that they reach their recipient safely and free from prying eyes. You may also need to digitally sign those files or messages to prevent them from being tampered with.

A great tool for these types of uses is the *GNU Privacy Guard* (aka *GnuPG* or simply *GPG*), which is a free and open source implementation of the proprietary *Pretty Good Privacy (PGP)*. *GPG* uses the *OpenPGP* standard as defined by the *OpenPGP Working Group* of the *Internet Engineering Task Force (IETF)* in RFC 4880. In this lesson we will be reviewing the basics of the *GNU Privacy Guard*.

### Perform Basic GnuPG Configuration, Usage and Revocation

Just as with SSH, the underlying mechanism to GPG is that of *asymmetric cryptography* or *public-key cryptography*. A user generates a key pair which is made up of a *private key* and a *public key*. The keys are mathematically related in such a way that what is encrypted by one can only be decrypted by the other. For communication to take place successfully, the user has to send their

public key to the intended recipient.

## GnuPG Configuration and Usage

The command to work with GPG is `gpg`. You can pass it a number of options to carry out different tasks. Let us start by generating a key pair as user `carol`. For that, you will use the `gpg --gen-key` command:

```
carol@debian:~$ gpg --gen-key
gpg (GnuPG) 2.2.12; Copyright (C) 2018 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory '/home/carol/.gnupg' created
gpg: keybox '/home/carol/.gnupg/pubring.kbx' created
Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name:

(...)
```

After informing you—amongst other things—that the configuration directory `~/.gnupg` and your public keyring `~/.gnupg/pubring.kbx` have been created, `gpg` goes on to ask you to provide your real name and email address:

```
(...)
Real name: carol
Email address: carol@debian
You selected this USER-ID:
    "carol <carol@debian>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit?
```

If you are OK with the resulting USER-ID and press `o`, you will be then asked for a passphrase (it is recommended that it has enough complexity):

```
| Please enter the passphrase to |
| protect your new key          |
```



**private-keys-v1.d**

This is the directory that keeps your private keys, therefore permissions are restrictive.

**pubring.kbx**

This is your public keyring. It stores your own as well as any other imported public keys.

**trustdb.gpg**

The trust database. This has to do with the concept of *Web of Trust* (which is outside the scope of this lesson).

**NOTE**

The arrival of *GnuPG 2.1* brought along some significant changes, such as the disappearance of the `secring.gpg` and `pubring.gpg` files in favour of `private-keys-v1.d` and `pubring.kbx`, respectively.

Once your key pair has been created, you can view your public keys with `gpg --list-keys` — which will display the contents of your public keyring:

```
carol@debian:~/gnupg$ gpg --list-keys
/home/carol/.gnupg/pubring.kbx
-----
pub  rsa3072 2020-07-03 [SC] [expires: 2022-07-03]
     D18FA0021F644CDAF57FD0F919BBEFD16813034E
uid          [ultimate] carol <carol@debian>
sub  rsa3072 2020-07-03 [E] [expires: 2022-07-03]
```

The hexadecimal string `D18FA0021F644CDAF57FD0F919BBEFD16813034E` is your *public key fingerprint*.

**NOTE**

Apart from the `USER-ID` (`carol` in the example), there is also the `KEY-ID`. The `KEY-ID` consists of the last 8 hexadecimal digits in your public key fingerprint (`6813034E`). You can check your key fingerprint with the command `gpg --fingerprint USER-ID`.

**Key Distribution and Revocation**

Now that you have your public key, you should save it (i.e. *export* it) to a file so that you can make it available to your future recipients. They will then be able to use it to encrypt files or messages intended for you (since you are the only one in possession of the private key, you will also be the only one able to decrypt and read them). Likewise, your recipients will also use it to decrypt and verify your encrypted or signed messages/files. The command to use is `gpg --export` followed by the `USER-ID` and a redirection to the output file name of your choice:

```
carol@debian:~/gnupg$ gpg --export carol > carol.pub.key
carol@debian:~/gnupg$ ls
carol.pub.key  openpgp-revocs.d  private-keys-v1.d  pubring.kbx  trustdb.gpg
```

**NOTE**

Passing the `-a` or `--armor` option to `gpg --export` (e.g.: `gpg --export --armor carol > carol.pub.key`) will create ASCII armored output (instead of the default binary OpenPGP format) which can be safely emailed.

As already noted, you must now send your public key file (`carol.pub.key`) to the recipient with whom you want to exchange information. For instance, let us send the public key file to `ina` on remote server `halof` using `scp` (*secure copy*):

```
carol@debian:~/gnupg$ scp carol.pub.key ina@halof:/home/ina/
Enter passphrase for key '/home/carol/.ssh/id_ecdsa':
carol.pub.key
100% 1740  775.8KB/s  00:00
carol@debian:~/gnupg$
```

`ina` is now in the possession of `carol.pub.key`. She will use it to encrypt a file and send it to `carol` in the next section.

**NOTE**

Another means of public key distribution is through the use of *key servers*: you upload your public key to the server with the command `gpg --keyserver keyserver-name --send-keys KEY-ID` and other users will get (i.e. *import*) them with `gpg --keyserver keyserver-name --recv-keys KEY-ID`.

Let us close this section by discussing key revocation. Key revocation should be used when your private keys have been compromised or retired. The first step is to create a revocation certificate by passing `gpg` the option `--gen-revoke` followed by the `USER-ID`. You can precede `--gen-revoke` with the `--output` option followed by a destination file name specification to save the resulting certificate into a file (instead of having it printed on the terminal screen). The output messages throughout the revocation process are quite self-explanatory:

```
sonya@debian:~/gnupg$ gpg --output revocation_file.asc --gen-revoke sonya

sec  rsa3072/0989EB7E7F9F2066 2020-07-03 sonya <sonya@debian>

Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
 0 = No reason specified
```

```

1 = Key has been compromised
2 = Key is superseded
3 = Key is no longer used
Q = Cancel
(Probably you want to select 1 here)
Your decision? 1
Enter an optional description; end it with an empty line:
> My laptop was stolen.
>
Reason for revocation: Key has been compromised
My laptop was stolen.
Is this okay? (y/N) y
ASCII armored output forced.
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable. But have some caution: The print system of
your machine might store the data and make it available to others!

```

The revocation certificate has been saved to the file `revocation_file.asc` (asc for ASCII format):

```

sonya@debian:~/gnupg$ ls
openpgp-revocs.d private-keys-v1.d pubring.kbx revocation_file.asc trustdb.gpg
sonya@debian:~/gnupg$ cat revocation_file.asc
-----BEGIN PGP PUBLIC KEY BLOCK-----
Comment: This is a revocation certificate

iQHDBCABCgAtFiEEiIVjfDnnpieFi0wvnlcN6yLCeHEFA18ASx4PHQJzdG9sZW4g
bGFwdG9wAAoJEJ5XDesiwnhxT9YMAKkjQiMpo9Uyiy9hyvukPPSrLcmtAGLk4pKS
pLZfzA5kxa+HPQwBgIAEvfNRR6VMxqXUgUGYC/IAYQQM62oNACy2PCPrxyJNgVF7
8l4mMZKvW++5ikjZwyg6WVW0+w6oro9qruJFjcu752p4T+9gsHV2r+KRqcPQe
aZ65sAvsBJlcsUDZqfWUXg2kQp9mNPCdQuqvDaKRgNCHA1zbzNFzXWVd2X5RgFo5
nY+tUP8ZQA9DTQPBLPcggICmfLopMPZYB2bft5geb2mMi2oNpf9CNPdQkdcimNV
aRjqdUP9C89PwTafBQkQi0NlsR/dWTFcqrG5KOWQPA7xjeMV8wretdEgSyTxqHp
v1iRzWjshiJCKBXXvz7wSmQrJ40fiMDHeS4ipR0AYd08QCzm0zmcFQKikGSHGMy1
z/YRltd6NZIKjf1TD0nTrFnRvPdsZ0lKYSArbfqNrHRBQkgirOD4JPI1tYKTffq
i0eZFx25K+fj2+0AJjvrbe4HDo5m+Q==
=umI8
-----END PGP PUBLIC KEY BLOCK-----

```



To effectively revoke your private key, you now need to merge the certificate with the key, which is done by importing the revocation certificate file to your keyring:

```
sonya@debian:~/gnupg$ gpg --import revocation_file.asc
gpg: key 9E570DEB22C27871: "sonya <sonya@debian>" revocation certificate imported
gpg: Total number processed: 1
gpg:    new key revocations: 1
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid: 1  signed: 0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2022-07-04
```

List your keys now and you will be informed about your revoked key:

```
sonya@debian:~/gnupg$ gpg --list-keys
/home/sonya/.gnupg/pubring.kbx
pub  rsa3072 2020-07-04 [SC] [revoked: 2020-07-04]
    8885637C39E7A627858B4C2F9E570DEB22C27871
uid  [revoked] sonya <sonya@debian>
```

Last — but not least — make sure you make the revoked key available to any party that has public keys associated with it (including keyservers).

## Use GPG to Encrypt, Decrypt, Sign and Verify Files

In the previous section, `carol` sent her public key to `ina`. We will use it now to discuss how GPG can encrypt, decrypt, sign and verify files.

### Encrypting and Decrypting Files

First, `ina` must import `carol`'s public key (`carol.pub.key`) into her keyring so that she can start working with it:

```
ina@halof:~> gpg --import carol.pub.key
gpg: /home/ina/.gnupg/trustdb.gpg: trustdb created
gpg: key 19BBEFD16813034E: public key "carol <carol@debian>" imported
gpg: Total number processed: 1
gpg:    imported: 1
ina@halof:~> gpg --list-keys
/home/ina/.gnupg/pubring.kbx
-----
pub  rsa3072 2020-07-03 [SC] [expires: 2022-07-03]
```

```
D18FA0021F644CDAF57FD0F919BBEFD16813034E
uid          [ unknown] carol <carol@debian>
sub   rsa3072 2020-07-03 [E] [expires: 2022-07-03]
```

Next you will create a file by writing some text into it and then encrypt it using `gpg` (because you did not sign carol's key, you will be explicitly asked if you want to use that key):

```
ina@halof:~> echo "This is the message ..." > unencrypted-message
ina@halof:~> gpg --output encrypted-message --recipient carol --armor --encrypt unencrypted-
message
gpg: 0227347CC92A5CB1: There is no assurance this key belongs to the named user
sub   rsa3072/0227347CC92A5CB1 2020-07-03 carol <carol@debian>
  Primary key fingerprint: D18F A002 1F64 4CDA F57F  D0F9 19BB EFD1 6813 034E
    Subkey fingerprint: 9D89 1BF9 39A4 C130 E44B  1135 0227 347C C92A 5CB1

It is NOT certain that the key belongs to the person named
in the user ID.  If you really know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

Let us break down the `gpg` command:

### **--output encrypted-message**

Filename specification for the encrypted version of the original file (`encrypted-message` in the example).

### **--recipient carol**

Recipient's USER-ID specification (`carol` in our example). If not provided, GnuPG will ask for it (unless `--default-recipient` is specified).

### **--armor**

This option produces ASCII armored output, which can be copied into an email.

### **--encrypt unencrypted-message**

Filename specification of the original file to encrypt.

You can now send the `encrypted-message` to `carol` on `debian` using `scp`:

```
ina@halof:~> scp encrypted-message carol@debian:/home/carol/
carol@debian's password:
```

```
encrypted-message
1.8MB/s  00:00
```

100% 736

If you log in as `carol` now and try to read the `encrypted-message`, you will confirm that it is actually encrypted and — therefore — unreadable:

```
carol@debian:~$ cat encrypted-message
-----BEGIN PGP MESSAGE-----

hQGMAwInNHZJKlyxAQv/brJ8Ubs/xya35sbv6kdRkm1C70NLxL30ueWA4mCs0Y/P
GBna6ZEUCrMEgl/rCyByj3Yq74kuiTmzxAIRUDdvHfj0Ttr0WjVAqIn/fPSfMkjK
dTxKo1i55tLJ+sJ17dGMZDcNBInBTP4U1atuN71A5w7vH+XpcesRcFQLKiS0mYTt
F7SN3/5x5J6io4ISn+b0KbJgiJNNx+Ne/ub4Uzk4N1K7tmBklyC1VRualtxcG7R9
1k1BPYSld6fTdDwT1Y4MofpyILAIgMZvUR1RXauEKf70IzwC5gWU+UQPSgeCdKQu
X7QL0ZIBS0Ug2XKr01k93lmDjf8PwsRIm16n/hNe1a0BA3HMP0b60zv1gFeEsFvC
IxhUYpb+rfuNFTMEB7xIO94AAmWB9N4qknMxdDqNE8WhA728Plw6y8L2ngsplY15
MR4lIFDpljA/CcVh4BXVe9j0TdFWDUkrFMfaIfcPQwKLXEYJp19XYIaaEazk0s5D
W4pENN0Y0cX0KWyAYX6r0l8BF0rq/HMenQwqAVXMG3s8ATuU0eqjBbR1x1qCvRQP
CR/3V73aQwc2j5ioQmhwYppxiro0yKX2Ar/E6rZyJtJYrq+CUk803JoBaudknNFj
pwuRwF1amwnSZ/MZ/9kMKQ==
=g1jw
-----END PGP MESSAGE-----
```

However, as you are in possession of the private key, you can easily decrypt the message by passing `gpg` the `--decrypt` option followed by the path to the encrypted file (the private key's passphrase will be required):

```
carol@debian:~$ gpg --decrypt encrypted-message
gpg: encrypted with 3072-bit RSA key, ID 0227347CC92A5CB1, created 2020-07-03
"carol <carol@debian>"
This is the message ...
```

You can also specify the `--output` option to save the message into a new unencrypted file:

```
carol@debian:~$ gpg --output unencrypted-message --decrypt encrypted-message
gpg: encrypted with 3072-bit RSA key, ID 0227347CC92A5CB1, created 2020-07-03
"carol <carol@debian>"
carol@debian:~$ cat unencrypted-message
This is the message ...
```

## Signing and Verifying Files

Apart from encrypting, GPG can also be used to sign files. The `--sign` option is relevant here. Let us start by creating a new message (`message`) and signing it with the `--sign` option (your private key's passphrase will be required):

```
carol@debian:~$ echo "This is the message to sign ..." > message
carol@debian:~$ gpg --output message.sig --sign message
(...)
```

Breakdown of the `gpg` command:

### `--output message`

Filename specification of the signed version of the original file (`message.sig` in our example).

### `--sign message`

Path to original file.

#### NOTE

Using `--sign` the document is compressed and then signed. The output is in binary format.

Next we will transfer the file to `ina` on `halof` using `scp message.sig ina@halof:/home/ina`. Back as `ina` on `halof`, you can now verify it by using the `--verify` option:

```
ina@halof:~> gpg --verify message.sig
gpg: Signature made Sat 04 jul 2020 14:34:41 CEST
gpg:                using RSA key D18FA0021F644CDAF57FD0F919BBEFD16813034E
gpg: Good signature from "carol <carol@debian>" [unknown]
(...)
```

If you also want to read the file, you have to decrypt it to a new file (`message` in our case) using the `--output` option:

```
ina@halof:~> gpg --output message --decrypt message.sig
gpg: Signature made Sat 04 jul 2020 14:34:41 CEST
gpg:                using RSA key D18FA0021F644CDAF57FD0F919BBEFD16813034E
gpg: Good signature from "carol <carol@debian>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:                There is no indication that the signature belongs to the owner.
Primary key fingerprint: D18F A002 1F64 4CDA F57F  D0F9 19BB EFD1 6813 034E
ina@halof:~> cat message
```

```
This is the message to sign ...
```

## GPG-Agent

We will round this lesson off by briefly touching upon `gpg-agent`. `gpg-agent` is the daemon that manages private keys for GPG (it is started on demand by `gpg`). To view a summary of the most useful options, run `gpg-agent --help` or `gpg-agent -h`:

```
carol@debian:~$ gpg-agent --help
gpg-agent (GnuPG) 2.2.4
libgcrypt 1.8.1
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Syntax: gpg-agent [options] [command [args]]
Secret key management for GnuPG

Options:

  --daemon                run in daemon mode (background)
  --server                run in server mode (foreground)
  --supervised            run in supervised mode
  -v, --verbose           verbose
  -q, --quiet             be somewhat more quiet
  -s, --sh                sh-style command output
  -c, --csh               csh-style command output
  (...)                  ...
```

**NOTE** For more information, consult the `gpg-agent` man page.

# Guided Exercises

1. Complete the table by providing the correct filename:

Description	Filename
Trust database	
Directory for revocation certificates	
Directory for private keys	
Public keyring	

2. Answer the following questions:

- What type of cryptography is used by *GnuPG*?

- What are the two main components of public key cryptography?

- What is the KEY-ID of the public key fingerprint 07A6 5898 2D3A F3DD 43E3 DA95 1F3F 3147 FA7F 54C7?

- What method is used to distribute public keys at a global level?

3. Put the following steps in the right order concerning private key revocation:

- Make the revoked key available to your correspondents.
- Create a revocation certificate.
- Import the revocation certificate to your keyring.

The correct order is:

Step 1:	
Step 2:	
Step 3:	

4. Regarding file encryption, what does the `--armor` option imply in the command `gpg --output encrypted-message --recipient carol --armor --encrypt unencrypted-`

message?

## Explorational Exercises

1. Most `gpg` options have both a long and a short version. Complete the table with the corresponding short version:

Long version	Short version
<code>--armor</code>	
<code>--output</code>	
<code>--recipient</code>	
<code>--decrypt</code>	
<code>--encrypt</code>	
<code>--sign</code>	

2. Answer the following questions concerning key export:

- What command would you use to export all of your public keys to a file called `all.key`?

- What command would you use to export all of your private keys to a file called `all_private.key`?

3. What `gpg` option allows for carrying out most key management related tasks by presenting you with a menu?

4. What `gpg` option allows you to make a cleartext signature?



## Summary

This lesson has covered the *GNU Privacy Guard*, an excellent choice to encrypt/decrypt and digitally sign/verify files. You learned:

- how to generate a pair of keys.
- how to list the keys in your keyring.
- the contents of the `~/ .gnupg` directory.
- what `USER-ID` and `KEY-ID` are.
- how to distribute public keys to your correspondents.
- how to globally distribute public keys through key servers.
- how to revoke private keys.
- how to encrypt and decrypt files.
- how to sign and verify files.
- the basics of the *GPG-Agent*.

The following commands were discussed in this lesson:

### **gpg**

*OpenPGP* encryption and signing tool.

# Answers to Guided Exercises

1. Complete the table by providing the correct file name:

Description	Filename
Trust database	trustdb.gpg
Directory for revocation certificates	opengp-revocs.d
Directory for private keys	private-keys-v1.d
Public keyring	pubring.kbx

2. Answer the following questions:

- What type of cryptography is used by *GnuPG*?

Public key cryptography or asymmetric cryptography.

- What are the two main components of public key cryptography?

The public and the private keys.

- What is the KEY-ID of the public key fingerprint 07A6 5898 2D3A F3DD 43E3 DA95 1F3F 3147 FA7F 54C7?

FA7F 54C7

- What method is used to distribute public keys at a global level?

Key servers.

3. Put the following steps in the right order concerning private key revocation:

- Make the revoked key available to your correspondents
- Create a revocation certificate
- Import the revocation certificate to your keyring

The correct order is:

<b>Step 1:</b>	Create a revocation certificate
<b>Step 2:</b>	Import the revocation certificate to your keyring

**Step 3:**

Make the revoked key available to your correspondents

4. Regarding file encryption, what does the `--armor` option imply in the command `gpg --output encrypted-message --recipient carol --armor --encrypt unencrypted-message`?

It produces ASCII armored output, which allows you to copy the resulting existing encrypted file into an email.

## Answers to Explorational Exercises

1. Most `gpg` options have both a long and a short version. Complete the table with the corresponding short version:

Long version	Short version
<code>--armor</code>	<code>-a</code>
<code>--output</code>	<code>-o</code>
<code>--recipient</code>	<code>-r</code>
<code>--decrypt</code>	<code>-d</code>
<code>--encrypt</code>	<code>-e</code>
<code>--sign</code>	<code>-s</code>

2. Answer the following questions concerning key export:

- What command would you use to export all of your public keys to a file called `all.key`?

```
gpg --export --output all.key or gpg --export -o all.key
```

- What command would you use to export all of your private keys to a file called `all_private.key`?

```
gpg --export-secret-keys --output all_private.key or gpg --export-secret-keys -o all_private.key (--export-secret-keys can be replaced by --export-secret-subkeys with a slightly different outcome—check man gpg for more information).
```

3. What `gpg` option allows for carrying out most key management related tasks by presenting you with a menu?

```
--edit-key
```

4. What `gpg` option allows you to make a cleartext signature?

```
--clearsign
```

## Imprint

© 2023 by Linux Professional Institute: Learning Materials, “LPIC-1 (102) (Version 5.0)”.

PDF generated: 2023-07-13

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0). To view a copy of this license, visit

<https://creativecommons.org/licenses/by-nc-nd/4.0/>

While Linux Professional Institute has used good faith efforts to ensure that the information and instructions contained in this work are accurate, Linux Professional Institute disclaims all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

The LPI Learning Materials are an initiative of Linux Professional Institute (<https://lpi.org>). Learning Materials and their translations can be found at <https://learning.lpi.org>.

For questions and comments on this edition as well as on the entire project write an email to: [learning@lpi.org](mailto:learning@lpi.org).