A Notebook on Linux Operating System

To all family members, friends and communities members who have been dedicating to the presentation of this notebook, and to all students, researchers and faculty members who might find this notebook helpful.

Contents

Fo	rewo	ord	ix
Pı	refac	е	xi
Li	st of	Figures	xiii
Li	st of	Tables	$\mathbf{x}\mathbf{v}$
Ι	Liı	nux Basics	1
1	Brie	ef Introduction to Linux	3
	1.1	Brief Introduction	3
	1.2	A Short History of Linux	4
	1.3	Linux Distributions	5
	1.4	Linux Graphical Desktop	7
	1.5	Linux Installation	7
2	She	ll Basics	11
	2.1	Brief Introduction	11
		2.1.1 Shell Types	12
		2.1.2 Prompt and Basic Concepts	12
	2.2	Useful Commands	13
		2.2.1 Set Shell Environment Variables	13
		2.2.2 Display User Information	15
		2.2.3 Display Machine Information	15
		2.2.4 Perform Simple Files Operations	15
		2.2.5 Set Alias and Shortcuts	16
		2.2.6 Others	16
	2.3	A Taste of Bash Shell Script Programming	17
3	Tex	t File Editing	21
	3.1	General Introduction to Vim	21
	3.2	Vim Modes	22
	3.3	Vim Profile Configuration	25
		3.3.1 Mapping Shortcuts	25
		3.3.2 Syntax and Color Scheme	26
		3.3.3 Plug Tools	26

	3.4	Basic Operations in Vim	28 28
		3.4.2 Search in the Text	31
	3.5	Visual Modes of Vim	32
	3.6	Vim Macros	33
	3.7	Other Text Editors	33
4	File	Management	37
	4.1	Filesystem Hierarchy Standard	37
	4.2	Commonly Used File Management Commands	41
		4.2.1 Print Working Directory	41
		4.2.2 List Information about the Files	41
		4.2.3 Create an Empty or a Simple Text File	42
		4.2.4 Create an Empty Directory	43
		4.2.5 Move, Copy-and-Paste, and Remove Files and Directo-	
		ries	44
		4.2.6 Use of Metacharacters	45
	4.3	Access Control List	45
		4.3.1 Change Ownership and Group of a File or Directory .	46
		4.3.2 Change Permissions of a File or Directory	46
	4.4	Search through the System	48
		4.4.1 Look for a Command	48
		4.4.2 Look for a File by Metadata	48
		4.4.3 Look for a File by Content	49
5	Soft	ware Management	51
	5.1	Linux Kernel Management	51
	5.2	General Introduction to Linux Package Management Tools .	51
	5.3	Installation of Software	51
	5.4	Software Upgrade	52
	5.5	Uninstallation of Software	52
6	Pro	cess Management	53
	6.1	General Introduction to Process	53
	6.2	Running Process Management on Linux	53
II	\mathbf{Li}	nux Advanced	55
7	Adn	ninistration Basics	57
	7.1	Introduction to Linux Administration	57
8	Acc 8.1	ount Management Quick Account Management	59 59
9	Disk	x Management	61
		Hard Drive Check	61

Contents	vii
10 Advanced Software Management	63
10.1 Git for Software Version Control and Joint Development	63
10.1.1 Brief Introduction to Git	63
10.1.2 Local Repository Operations	63
10.1.3 Remote Repository Operations	63
III Linux Server Management	65
11 Virtualization and Containerization	67
11.1 Virtual Machine	70
11.2 Container	70
11.3 Docker Basics	71
11.3.1 Docker Installation	71
11.3.2 Launch a Container	72
11.3.3 Access and Manage a Container	74
11.3.4 Publish a Container	76
11.4 Docker Images	78
IV Linux Security	7 9
V Linux on Cloud	81
Bibliography	83

Foreword

If a piece of software or an e-book can be made completely open source, why not a notebook?

This brings me back to the summer of year 2009, when I just started my third year as a high school student in Harbin No. 3 High School. In around August and September of every year, that is, when the results of Gaokao (National College Entrance Examination of China, annually held in July) are released, people from photocopy shops will start selling notebooks photocopies that they claim to be of the top scorers of the exam. Much as I was curious about what these notebooks look like, I myself did not expect to actually learn anything from them, mainly for the following three reasons.

First of all, some (in fact many) of these notebooks were more tough to understand than the textbooks. I guess we cannot blame the top scorers for being too smart and making things sometimes extremely brief or overwhelmingly complicated.

Secondly, why would I wanted to adapt to notebooks of others when I had my own, which should be as good as theirs.

And lastly, as a student in the top high school myself, I knew that the top scorers of the coming year would probably be a schoolmate or a classmate. Why would I want to pay that much money to a complete stranger in a photocopy shop for my friend's notebook, rather than asked from him or her directly?

However, things had changed after my becoming an undergraduate student in year 2010. Since in the university there were so many modules and materials to learn, students were often distracted from digging into one book or module very deeply. (For those who were still able to do so, you have my highest respect.) The situation got even worse as I became a Ph.D. student in year 2014, this time due to that I had to focus on one research topic entirely, and could hardly split much time on other irrelevant but still important and interesting contents.

This motivated me to start reading and taking notebooks for selected books and articles such as journal papers and magazines, just to force myself to spent time learning new subjects. I usually used hand-written notebooks. My very first notebook was on *Numerical Analysis*, an entrance level module for engineering background graduate students. Till today I have on my hand dozens of notebooks, and one day it suddenly came to me: why not digitalize them, and make them accessible online and open source, and let everyone read and edit it?

x Foreword

As majority of open source software, this notebook (and it applies to the other notebooks in this series) does not come with any "warranty" of any kind, meaning that there is no guarantee for the statement and knowledge in this notebook to be exactly correct as it is not peer reviewed. **Do NOT cite this notebook in your academic research paper or book!** Of course, if you find anything here useful with your research, feel free to trace back to the origin of the citation, and double confirm it yourself then on top of that determine whether or not to use it in your research.

This notebook is suitable as:

- a quick reference guide;
- a brief introduction for beginners of the module;
- a "cheat sheet" for students to prepare for the exam (Don't bring it to the exam unless it is allowed by your lecture!) or for lectures to prepare the teaching materials.

This notebook is NOT suitable as:

- a direct research reference;
- a replacement to the textbook;

because as explained the notebook is NOT peer reviewed and it is meant to be simple and easy to read. It is not necessary brief, but all the tedious explanation and derivation, if any, shall be "fold into appendix" and a reader can easily skip those things without any interruption to the reading.

Although this notebook is open source, the reference materials of this notebook, including textbooks, journal papers, conference proceedings, etc., may not be open source. Very likely many of these reference materials are licensed or copyrighted. Please legitimately access these materials and properly use them if necessary.

Some of the figures in this notebook is drawn using Excalidraw, a very interesting tool for machine to emulate hand-writing. The Excalidraw project can be found in GitHub, excalidraw/excalidraw.

Preface

Some references of this notebook are the Linux Bible (10th edition) that I borrowed from National Library Singapore, and also many Bilibili and YouTube videos, which I will cite as I go through the notebook.

List of Figures

1.1	GNOME desktop environment	7
1.2	KDE desktop environment	8
1.3	LXDE desktop environment	8
1.4	Xfce desktop environment	8
3.1	Mode switching between normal mode and insert mode, and	
	basic functions associated with the modes	23
3.2	A flowchart for simple creating, editing and saving of a text file	
	using Vim	24
3.3	A piece of text of "William Shakespeare", for demonstration.	28
3.4	Search "he" in the piece of text of "William Shakespeare"	32
3.5	An example of visual mode where a block of text is selected	32
3.6	Vim (with user's profile customization as introduced in this	
	${\rm chapter}). \ . \ . \ . \ . \ . \ . \ . \ . \ . \$	33
3.7	<i>Nano.</i>	34
3.8	Emacs	34
3.9	<i>Gedit.</i>	35
3.10	Visual Studio Code	35
3.11	<i>Atom.</i>	36
4.1	Linux file system hierarchy	38
4.2	A rough categorization of commonly used directories in Linux	
	file hierarchy standard	39
4.3	List down information of files and subdirectories in the current	
	working directory.	42
4.4	Change ownership and group of a file	47
4.5	Change 9-bit permission (mode) of a file	47
4.6	Search for files and directories using <i>locate</i>	49
11.1	System architectures of PC, VM and container	68
11.2	PC implementation: a cook in a kitchen	69
11.3	VM implementation: many cooks in a kitchen, each with a different cookbook	69
11 4	Container implementation: one in a kitchen, handling multiple	00
	dishes, each has a cookbook and stays in its own pan	70

Х	tiv	List of Figures

11.5	An example of running apline container, with interactive TTY	
	and name test-apline	73
11.6	List the up running container test-apline	73
11.7	List the exited container test-apline	73
11.8	A simplified architecture where containers are used to host the	
	web service.	77

List of Tables

2.1	Commonly used shell environment variables	14
2.2	Shell configuration files	17
3.1	Commonly used modes in Vim	23
3.2	Commonly used shortcuts to switch from normal mode to insert	0.4
	mode	24
3.3	Commonly used operators related to delete/cut, change, copy	20
	and paste	30
3.4	Commonly used motions	30
1 1	T	
4.1	Introduction to commonly used directories in Linux file hierar-	
	chy standard	40
4.2	Commonly used commands to navigate in the Linux file system.	41
4.3	Commonly used arguments and their effects for <i>ls</i> command.	43
4.4	Commonly used arguments and their effects for mv and cp com-	
	mand	44
4.5	Commonly used arguments and their effects for <i>rm</i> command.	45
4.6	Commonly used metacharacters	45
4.7	Three types of permissions	46
11.1	Commonly used docker commands to launch a container	75

Part I Linux Basics

1

Brief Introduction to Linux

CONTENTS

1.1	Brief Introduction	3
1.2	A Short History of Linux	4
1.3	Linux Distributions	5
1.4	Linux Graphical Desktop	6
1.5	Linux Installation	7

This chapter gives a brief introduction to Linux, including some of its key features and advantages/disadvantages comparing with other operating systems.

1.1 Brief Introduction

Linux is an operating system (OS). An OS is essentially a special piece of software running on a machine (computer, server, mobile devices, or other electrical device that is capable and sophisticated enough to host an OS) that manages hardware resources of the system and provide services to the application software in the upper layer. An OS shall be able to

- detect and prepare hardware;
- manage processes;
- manage memory;
- provide user interface and user authentication;
- manage file systems;
- provide programming tools for creating applications.

Linux has been overwhelmingly successful and adopted in many areas. For example, Android operating system for mobile phones is developed using Linux. Google Chrome is also backed by Linux. Many famous websites including Facebook are also running on Linux servers.

Some of the most favorable features of Linux (especially to large size enterprises) are as follows.

- Clustering: multiple machines work together as a whole, and they appear to be a single machine to upper layer applications.
- Visualization: one machine hosts multiple applications, each of which thinks that it is running on a dedicated machine.
- Cloud computing: flexible resources management is achieved by running applications on clouds on virtual Linux computers.
- Real-time computing: embedded Linux is implemented on micro-controllers or micro-computers for real-time edge control.

Linux differs from Microsoft Windows and MacOS in many ways, though they are all very good OSs. Among the three OSs, Linux is the only OS that is completely open source (in the sense that all its code can be viewed and modified per requested) and can be customized per requested by the users very flexibly.

1.2 A Short History of Linux

The initial motivation of Linux is to create a UNIX-like operating system that can be freely distributed in the community.

Many modern OSs including MacOS and Linux are derived from UNIX. UNIX operating system was created by AT&T in 1969 as a better software development environment that AT&T used internally. In 1973, UNIX was rewritten in C language, thus adding more useful features such as portability. Today, C is still the primary language used to create UNIX (and also Linux) kernels.

AT&T, who originally owned UNIX, tried to make money from UNIX. Back then AT&T was restricted from selling computers by the government. Therefore, AT&T decided to license UNIX source code to universities for a nominal fee. Researchers from universities started learning and improving UNIX, which speeded up the development of UNIX. In 1976, UNIX V6 became the first UNIX that was widely spread. UNIX V6 was developed at UC Berkeley and was named the Berkeley Software Distribution (BSD).

From then on, UNIX moved towards two separate directions: BSD continued forward in the "open" and "share" manner, while AT&T started steering UNIX toward commercialization. By 1984 AT&T was pretty ready to start selling commercialized UNIX, namely "AT&T: UNIX System Laboratories (USL)". USL did not sell very well. As said, AT&T could only sell the OS

source code to other PC manufactures, but not the PC itself with UNIX preinstalled. For this reason the price for the source code had to be set higher as it is targeted for PC manufactures, not for end users. This largely prevented an end user from procuring UNIX from AT&T directly. Other companies, such as SCO and Sun Microsystems, were more successful than AT&T by selling UNIX based PC and workstations for high-end users. Overall, UNIX source code was extremely expensive.

In 1984, Richard Stallman started the GNU project as part of the Free Software Foundation. It is recursively named by phrase "GNU is Not UNIX", intended to become a recording of entire UNIX that could be open and freely distributed. The community started to "recreate" UNIX based on the defined interface protocols published by AT&T.

Linus Trovalds started creating his version of UNIX, i.e. Linux, in 1991. He managed to publish the first version of the Linux kernel on August 25, 1991, initially only worked for a 386 processor. Later in October, Linux 0.0.2 was released with many parts of the code rewritten in C language, making it more suitable for cross-platform usage. This Linux kernel was the last and the most important piece of code to complete a UNIX-like system under GNU General Public License (GPL). It is so important that people call this operating system "Linux OS" instead of "GNU OS", although GNU is the host of the project and Linux kernel is just a part (the most important part) of it.

1.3 Linux Distributions

As casual Linux users, people do not want to understand and compile the Linux source code to use Linux. In response to this need, different Linux distributions have merged. They share the same OS kernel but differ from each other in many ways such as software management and user interface.

Today, there are hundreds of Linux distributions in the community. The most famous two categories of distributions are as follows. Notice that although the source code of them are available in public, as required by GPL license (GPL requires that any modified code made available to the users if the product is made public), some of the distributions may come with a "subscription fee". The subscription fee is not for the source code, but for the technical support, paid maintenance, and other services that convenes the life of the users.

- Red Hat Distribution
 - Red Hat Enterprise Linux (RHEL)
 - Fedora
 - CentOS

- Debian Distribution
 - Ubuntu
 - Linux Mint
 - Elementary OS
 - Raspberry Pi OS

Some of the main features of Red Hat distributions are as follows. Red Hat created the RPM packaging format to manage the installation and upgrading of software. The RPM packaging contains not only the software files but also its metadata, including version tracking, the creator, the configuration files, etc. In the OS, a local RPM database is used to track all software on the machine. Anaconda installer simplifies the installation of Red Hat Linux, meantime leaving users enough flexibility for customization. Red Hat OS is integrated with simple graphical tools for device management (such as adding a printer), user management and other administration work.

Red Hat Enterprise Linux (RHEL) is a commercial, stable and well-supported product that works on features needed to handle mission-critical application for big business and government. To use RHEL, customers buy subscriptions which allow them to deploy any version of RHEL as desired. Different levels of support are available for RHEL depending on customers needs. Many add-on features, including cloud computing integration, are available for the customers.

CentOS is a "recreation" version of RHEL using freely available RHEL source code. In this sense, CentOS experience should be very similar with RHEL and it is free of charge, but the user will not enjoy the professional technical support from RHEL engineers. Recently, Red Hat took over the development of CentOS project.

Fedora is a free, cutting-edge Linux distribution sponsored by Red Hat. It is less stable than RHEL, and plays as the "testbed" for Red Hat to interact with the community. From this perspective, Fedora is very similar to RHEL, just with more dynamics and uncertainties. Some functions, especially server related functions, will eventually be implemented in RHEL after successfully tested on Fedora.

Ubuntu is the most successful Debian Linux distribution. It not only has an easy-to-use software managing tool like other Debian distributions, but also builds in a simple graphical installer and other graphical tools. It focuses on full-featured desktop system while still offering popular server packages. Ubuntu has a very active community to support its development.

Ubuntu has larger software pool than Fedora. Ubuntu and its associated software usually have a longer "lifespan" than Fedora in the sense that Ubuntu is target for more stable use but Fedora is more of a "testbed". In this sense, Ubuntu is more for casual users and Fedora more for advanced users or developers, especially developers for RHEL.



FIGURE 1.1 GNOME desktop environment.

1.4 Linux Graphical Desktop

Though not necessary for Linux, both Ubuntu and Fedora distributions (and many other Linux distributions) support graphical desktops. By default, both systems come with GNOME graphical desktop environment. There are of course other choice of graphical desktops available on line, such as KDE, LXDE and Xfce desktops. GNOME and KDE are more for regular computers while LXDE and Xfce are more light in size, thus more for low-power demanding systems.

Figs. 1.1, 1.2, 1.3 and 1.4 give the flavors of each desktop environment mentioned above. From the figures we can see that GNOME adopts a more Linux/MacOS style desktop environment, while KDE has a "Windows 7" style desktop. LXDE and Xfce are more simple in graphics presentations and they are more for embedded systems.

It is possible to install multiple desktop environment in one computer. In such a case, the user can choose which desktop environment to use each time the computer is powered on.

1.5 Linux Installation

Linux can be installed on a local PC hard disk, or on a mobile device such as a thumb drive. The installation of different distributions might differ. Thanks to the graphical installation tools for the popular distributions, the installations



FIGURE 1.2

KDE desktop environment.



FIGURE 1.3

 $\ensuremath{\mathsf{LXDE}}$ desktop environment.



FIGURE 1.4

Xfce desktop environment.

can be done easily by just following the instructions on the associated official sites.

Instructions of installing Ubuntu is given by https://ubuntu.com Instructions of installing Fedora is given by https://getfedora.org For the use of RHEL, consult with Red Hat at https://www.redhat.com

CONTENTS

2.1	Brief I	ntroduction	11
	2.1.1	Shell Types	11
	2.1.2	Prompt and Basic Concepts	12
2.2	Useful	Commands	13
	2.2.1	Set Shell Environment Variables	13
	2.2.2	Display User Information	14
	2.2.3	Display Machine Information	15
	2.2.4	Perform Simple Files Operations	15
	2.2.5	Set Alias and Shortcuts	16
	2.2.6	Others	16
2.3	A Tast	te of Bash Shell Script Programming	16

Linux command line tool, usually known as the "shell", is the most powerful tool for Linux operations including configuration and control of the OS. Notice that the use of the shell is not compulsory for casual users when the graphical desktop is present. Though the shell is not as intuitive as the graphical tools, it is more powerful and flexible, and well-supported by the community.

Linux shell will be used repeatedly in the remaining sections of this notebook for different functions.

2.1 Brief Introduction

Linux command line tool, usually known as Linux shell, was invented before the graphical tools, and it has been more powerful and flexible than the graphical tools from the first day. On those machines where no graphical desktops are installed, the use of shell is critical.

2.1.1 Shell Types

There are different types of shells. The most commonly used shell is the "bash shell" which stands for "Bourne Again Shell", derived from the "Bourne Shell" used in UNIX. An example calculate_fib.sh written in bash shell script is given below, where the first 10 terms in Fibonacci series is calculated "1, 1, 2, 3, 4, 8, 13, 21, 34, 55". This example will used multiple times in this chapter.

```
#!/usr/bin/bash
n=10
function fib
{
  x=1; y=1
  i=2
  echo "$x"
  echo "$y"
  while [ $i -lt $n ]
      i='expr $i + 1 '
     z='expr $x + $y '
     echo "$z"
     x=$y
     y=$z
  done
}
r='fib $n'
echo "$r"
```

Some other shells such as "C Shell" and "Korn Shell" are also popular among certain users or certain Linux distributions. For example, C Shell supports C-like shell programming, which can sometimes be more convenient then bash shell. In case where your Linux distribution does not have these shells pre-installed, you can install and use these shells just like installing other software

In this notebook, we will mostly focus on the bash shell.

2.1.2 Prompt and Basic Concepts

After opening the shell or terminal, you will see a string (usually containing username, hostname, current working directory, etc.) followed by either a \$ or #, starting from where you can input your shell command. For example, it may look like the following:

```
username@hostname:~$
```

The above displayed string is called a *prompt*, indicating the start of a manually input command. By default, for regular user, the ending of the prompt is \$ while for the root user, the ending is #. The prompt can be

customized by changing the environment variable PS1. See Sections 2.2.1, 2.2.6 for details about environment variable and shell configuration, respectively.

By saying root user, we are referring to a special user whose username and user ID (UID) "root" and 0 respectively. This UID gives him the administration privilege over the machine, such as adding/removing users, change ownership of files, etc. To avoid vital damage by human error, root user shall not be used unless it is definitely necessary. For this reason, in many servers the root user is deactivated (for example, by setting its login password to invalid).

Notice that a root user is different from regular user equipped with *sudo* privilege, though a regular user with sudo privilege can temporarily switch to root user by using **su** as follows.

```
regularuser@hostname:~$ sudo su
[sudo] password for regularuser:
root@hostname:/home/regularuser#
```

More about sudo privilege, **sudo** and **su** commands are introduced later parts of the notebook.

You can key in a command after the prompt, and execute the command by pressing the Enter key. A Linux shell command usually has the following form.

\$ <command> <configuration-arguments> <input>

2.2 Useful Commands

Some useful commands are introduced in this section by categories.

Text file editing, files management (such as changing ownership of files), software management (such as installation of software, checking version, upgrading software, etc.), process management (such as checking CPU usage, terminating a process) are commonly used in Linux operations. They are not included in this chapter, but introduced separately in later chapters.

2.2.1 Set Shell Environment Variables

The command to be executed must have been stored somewhere in the PATH environment of the shell. PATH environment is a series of directories (locations) in the system, and it is initialized automatically when the shell is started. Check the PATH environment by

```
$ echo $PATH
<directory 1>:<directory 2>:<directory 3>: ...
```

TABLE 2.1 Commonly used shell environment variables.

Vaniable	Description
<u>Variable</u>	Description
BASH	Full pathname of the bash command.
BASH_VERSION	Current version of the bash command.
EUID	Effective user ID number of the current user, which is as-
	signed when the shell starts, based on the user's entry in
	/etc/passwd.
HISTFILE	Location of the history file.
HISTFILESIZE	Maximum number of history entries.
HISTCMD	The number index of the current command.
HOME	Home directory of the current user.
PATH	Path to available commands.
PWD	Current directory.
OLDPWD	Previous directory.
SECONDS	Number of seconds since the shell starts.
RANDOM	Generating a random number between 0 and 99999.

where echo displays a line of text, and \$PATH is a built-in variable that records the PATH environment of the current bash. It is possible to include new directories to PATH environment either temporarily or permanently to include new commands.

Most Linux-defined user commands are stored under /bin, /usr/bin, and administrative commands in /sbin, /usr/sbin. Commands local to a specific user can be stored under /home/<username>/bin. To determine the location of a particular command, use type if the command is in \$PATH, or locate to search everywhere accessible files in the system. An example is given below.

\$ type <command> <command location>

Similar with PATH, there is a list of shell environment variables for the user to monitor and control and status of the system. Table 2.1 summarizes common shell environment variables. Command echo can be used to check the values of these variables.

The environmental variables can be edited and new environmental variables can be created as follows.

```
<variable name> = <variable value> ; export <variable name>
For example,
```

```
PATH = $PATH:/getstuff/bin ; export PATH
```

adds a new directory /getstuff/bin to the PATH environmental variable.

Use command env to check a list of environment variables in the shell.

2.2.2 Display User Information

Administrative users may need to frequently check the basic system information, such as hardware configuration, OS version, username, hostname, disk usage, running process, system clock, etc. Some useful commands are summarized below.

The following commands show basic information of a user.

```
$ whoami
<username>
$ grep <username> /etc/passwd
<username>:x:<uid>:<gid>:<gecos>:<home directory>:<shell>
```

In the above, whoami is used to display the current login user's username. Command grep is used to search a content (in this case, the user name) in the selected file /etc/passwd where the user information is stored. This should return the username, the password (for encrypted password, an "x" is returned), UID, group id (GID), user id info (GECOS), home directory and default shell location of the user. Another command id also returns the user id and group id information of the current user.

2.2.3 Display Machine Information

The following commands show the date and hostname of the machine.

```
$ date
<date, time and timezone>
$ hostname
<hostname>
```

The following command lshw lists down hardware information in details. Sudo privilege is recommended when using this command, to give detailed and accurate information of the system. Since the displayed information is so detailed and can take up many screens, sometimes it is more convenient to use -short argument.

\$ sudo lshw

2.2.4 Perform Simple Files Operations

The most important commands for navigating in the file system is to display the current working directory (may be included as part of prompt) and list down files and directories in the current working directory as follows.

```
$ pwd
<absolute working directory>
$ ls
<a list of files/directories in the working directory>
```

The aforementioned ls command can be used flexibly. Commonly seen arguments that come with ls are -l (implement long listing with more details of each item), -a (include hidden item in the list) and -t (list by time).

More file operations related commands are introduced in Chapter 4.

2.2.5 Set Alias and Shortcuts

Command alias is used to create short-cut keys for commands and associated options, which makes it more convenient for the system operators to work on the shell. Some alias has already been created automatically when the shell is started. Use alias to check the existing alias in the shell. An example is given below.

```
$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

A temporary alias can be added to the shell by using

\$ alias <shortcut command>='<original command and options>'
for example

```
$ alias pwd='pwd; ls -CF'
```

To permanently add alias to the shell, the alias needs to be added to the bash start script, which is usually ~/.bashrc for a user. See Section 2.2.6 for details.

2.2.6 Others

Many commands can be used flexibly and it is impossible to illustrate all their details. Consider use the following two methods to check the detailed manual about a command.

```
$ man <command>
$ <command> --help
```

Use history to check history commands. Use !<history command index> to repeat a history command, or use !! to repeat the latest previous command. It is possible to disable history recording function for privacy purpose.

Shell configuration files are loaded each time a new shell starts. User-defined permanent configurations (such as useful alias) can be put into these files so that the configurations can be implemented automatically. Some useful files are summarized in Table 2.2.

TABLE 2.2 Shell configuration files.

Shell comiguration i	nes.
File pathname	Description
/etc/profile	The environment information for every user, which ex-
	ecutes upon any user logs in. Root privilege is required
	to edit this file.
/etc/bashrc	Bash configuration for every user, which executes upon
	any user starts a shell. Root privilege is required to edit
	this file.
~/.bash_profile	The environment information for current user, which
	executes upon the user logs in.
~/.bashrc	Bash configuration for current user, which executes
	upon the user starts a shell.
~/.bash_logout	Bash log out configuration for current user, which exe-
	cutes upon the user logs out or exit the last bash shell.

2.3 A Taste of Bash Shell Script Programming

A truly power feature of the shell is its ability to redirect inputs/outputs of commands, thus to chain the commands together. Meta-characters pipe (|), ampersand (&), semicolon (;), dollar (\$), parenthesis (()), square bracket ([]), less than sign (<), greater than sign (>) and double greater than sign (>>), and a few more more, are used for this feature. Details are given below.

The pipe (I) connects the output of the first command to the input of the second command. The following example searches keyword "function" in calculate_fib.sh which was given previously.

\$ cat calculate_fib.sh | grep function function fib

where cat concatenates files and print on the standard output, and grep prints lines that match patterns in each file.

The semicolon (;) allows inputting multiple commands in the same line in the script. The commands are then executed one after another from left to right.

The ampersand (&) can be put in the end of a line so that the command on that line will run in the background. The commands or process running in the background does not occupy the shell standard display, and the users can continue working on other commands in parallel. This is particularly useful when a task is going to take a long time to be executed. To manage the tasks running in the background, check more details in Chapter 6.

Use the dollar sign \$ (not the prompt) to indicate a command expansion. The command in \$(<command>) will be executed as a whole, then treated as

a single input. The content in () is sometimes called sub-shell. For example, to display the function defined in calculate_fib.sh previously,

```
$ echo Display functions: $(cat calculate_fib.sh | grep function
)
Display functions: function fib
Use $[<arithmetic expression>] for simple calculations, such as
$ echo 1+1=$[1+1]
1+1=2
```

Another example to count the number of files/folders in the current directory is

```
\ echo There are (ls -a \mid wc -w) files in this directory. There are 69 files in this directory.
```

where wc counts the number of lines, words or bytes in a file.

The dollar sign \$\$ is also used to expand the value of a variable, either environmental variable or self-defined variable, as explained previously in 2.2.1.

The less than sign < and greater than sign > are used for input/output direction of a file. They are useful when a command needs to pull input and/or push output to a file instead of the standard input and output. An example using command sort together with input direction < is given as follows. Considering sorting characters "a", "c", "b", "g", "e", "f", "d" using sort command. The letters are input from the console as follows. Use ctrl+D to quit the input, and the output after sorting will be displayed in the console as follows.

```
$ sort
a
c
b
g
e
f
d
a
b
c
d
e
f
g
```

For demonstration purpose, create a file before_sort in the current working directory. Inside before_sort are letters "a", "c", "b", "g", "e", "f", "d", each occupying a separate row. There are several ways to create the file, which will

be explained later. For now, just assume that the file already exists. Use cat to quickly check its content as follows.

```
$ cat before_sort
a
c
b
g
e
f
d
```

Use sort to sort before_sort as follows. In this case, the input to sort becomes a file, rather than the standard input from the keyboard. Notice that in this example, sort before_sort also works, as sort will by default take its first argument as the location of the file to be sorted.

```
$ sort < before_sort
a
b
c
d
e
f
g</pre>
```

Use > to redirect the output of a command to a file as given in the following example.

```
$ sort < before_sort > after_sort
$ cat after_sort
a
b
c
d
e
f
g
```

where sort does not output the result to the console, but instead saves the result in a file named after_sort. The double greater sign >> works similarly with > except that >> will append the output to an existing file, while > overwrites the existing file.

With the above been said, it is possible to use the following to create the before_sort file that has been used in the example.

```
$ echo -e "a\nc\nb\ng\ne\nf\nd\n" > before_sort
```

Text File Editing

CONTENTS

3.1	General Introduction to Vim		
3.2	Vim Modes		
3.3	Vim F	Profile Configuration	23
	3.3.1	Mapping Shortcuts	25
	3.3.2	Syntax and Color Scheme	26
	3.3.3	Plug Tools	26
3.4	Basic Operations in Vim		27
	3.4.1	Cut, Change, Copy and Paste	28
	3.4.2	Search in the Text	31
3.5	Visual	Modes of Vim	32
3.6	Vim Macros		32
3 7	Other Text Editors		33

There are many applications that supports text files or programs editing in Linux, to name a few, *Vim*, *Atom*, *Visual Studio Code*, and many more. These text editors come with different features, and some of them may support advanced functions such as compiling and executing codes.

Among the popular text editors in the Linux community, *Vim* is probably the most popular one that can work in the shell without desktop graphic environment, thus the default built-in text editor for many Linux distributions. *Vim* is introduced in this chapter, followed by some other commonly used text editors.

3.1 General Introduction to Vim

Vim is a free and open-source software initially developed by Bram Moolennar, and has become the default text editor of many Unix/Linux based operating systems.

Some people claim Vim to be the most powerful text file editor as well as integrated development environment for programming on a Linux machine

(and potentially on all computers and servers). The main reasons are as follows.

- *Vim* is usually built-in to Linux during the operating system installation, making it the most available and cost-effective text editor.
- Vim can work on machines where graphical desktop is not supported.
- Vim is light in size and is suitable to run even on an embedded system.
- Vim operations are done mostly via mode switch and shortcut keys, so that the brain does not need to halt and wait for the hand to grab and move the mouse which slows down the text editing and interrupts the logic flow.
- Vim is highly flexible and can be customized according to the user's habit (for example, through ~/.vim/vimrc), and it allows the users to define shortcut keys.
- Vim can automate repetitive operations by defining macros.
- Vim can be integrated with third-party tools for useful functions such as browsing project folders.

Vim can be come very powerful and convenient for the user if he is very used to it. On the other hand, however, Vim is not as intuitive as other text editors such as gedit and notepad++, and there might be a learning curve for beginners.

3.2 Vim Modes

Unlike other text editors, *Vim* defines different "modes" during the operation, each mode has some unique features. For example, in the *insert* mode, *Vim* puts keyboard inputs into the text file like an conventional text editor. In the *normal* mode (this is the default mode when opening *Vim*), *Vim* uses useful and customizable shortcut keys to quickly navigate the document and perform operations such as cut, copy, paste, replace, search, and macro functions. In the *virtual* mode, *Vim* allows the user to select partial of the document for further editing. In the *cmdline* mode, *Vim* takes order from command lines and interact with Linux to perform tasks such as save, quit or even navigating folders.

The following Table 3.1 summarizes the commonly used modes in Vim.

As a start, the following basic commands can be used to quickly create, edit and save a text file using vim. In home directory, start a shell and key in

\$ vim testvim

TABLE 3.1

Mode	Description
Normal	Default mode. It is used to navigate the cursor in the text, search
	and replace text pieces, and run basic text operations such as
	undo, redo, cut (delete), copy and paste.
Insert	It is used to insert keyboard inputs into the text, just like com-
	monly used text editors today.
Visual	It is similar to normal mode but areas of text can be highlighted.
	Normal mode commands can be used on the highlighted text.
$\overline{\mathrm{Cmdline}}$	It takes in a single line command input and perform actions
	accordingly, such as save and quit.

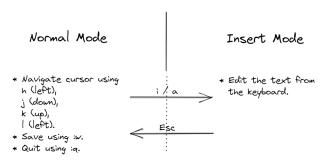


FIGURE 3.1

Mode switching between normal mode and insert mode, and basic functions associated with the modes.

to create a file named "testvim" and open the file using *Vim*. Notice that in some Linux versions, *vi* might be aliased to *vim* by default.

The above basic commands and their relationships are summarized in Fig. 3.1. A flowchart to create/open, edit, save, and quit a text file using the aforementioned commands are given in Fig. 3.2.

There are other shortcuts to switch from normal mode to insert mode. Some of them are summarized in Table 3.2.

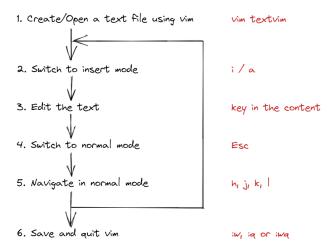


FIGURE 3.2

A flowchart for simple creating, editing and saving of a text file using Vim.

TABLE 3.2 Commonly used shortcuts to switch from normal mode to insert mode.

Operator	Description
i	Insert before the character at the cursor.
ī	Insert at the beginning of the row at the cursor.
a	Insert after the character at the cursor.
Ā — — — —	Insert at the end of the row at the cursor.
0	Create a new row below the cursor and switch to insert mode.
0	Create a new row above the cursor and switch to insert mode.

3.3 Vim Profile Configuration

With the basic operations introduced in Section 3.2, we are able to create and edit a text file as we want to, just like using any other text editor. Though at this point the advantages of using *Vim* over other text editors are not obvious yet, the *Vim* editor is at least useable.

Before introducing more advanced features of *Vim* for more convenient user experience, we can now customize the user profile to suit our individual habit. Notice that the customization is completely optional and personal. This section only introduces the ideas and basic methods of such customization, such as re-mapping keys and create user-defined shortcuts. Everything introduced here are merely examples and it is completely up to the user how to design and implement his own profile.

In Linux, navigate to home directory. Create the following path and file ~/.vim/vimrc or ~/.vimrc. Open the *vimrc* file as a blank file using *Vim*. The individual user profile can be customized here.

3.3.1 Mapping Shortcuts

It is desirable to re-map some keys to speed up editing. For example, people may want to map jj to Esc in insert mode for more convenient mode switching to normal mode (consequent "jj" is rarely used in English). Other people may feel like mapping j, k, i to h, j, k respectively in normal and visual modes, making the navigation more intuitive. In that case, a different key needs to be mapped for i since it is an important key for switching to insert mode.

It is possible re-map certain key (or keys combination) in selected modes. The following configuration in *vimrc* file re-maps the aforementioned keys.

```
inoremap jj <Esc>
noremap j h
noremap J H
noremap K j
noremap K J
noremap i k
noremap I K
noremap h i
noremap H I
```

where inoremap is used to map keys (combinations) in insert mode, and noremap in normal and visual modes.

The upper case letter S and lower case letter —s— in control mode are originally used to delete and substitute texts. They may be not so important in practice as there functions are overlapped by another shortcut key c, which is powerful in replacing characters and is more frequently used. We can re-map

S for saving the text, and disable s to prevent mis-touching. Similarly, upper case letter Q is mapped to quit Vim.

```
noremap s <nop>
map S :w<CR>
map Q :q<CR>
```

where <nop> stands for "no operation" and CR stands for the "enter" key on the keyboard. The keyword map differs from noremap in the sense that map is for recursive mapping.

3.3.2 Syntax and Color Scheme

By default *Vim* displays white color contents on black background. Use the following command in *vimrc* to enable syntax highlighting or change color scheme. Use :colorscheme in normal mode in *Vim* to check for available color schemes.

```
syntax on colorscheme default
```

The following command displays the row index and cursor line (a underline at cursor position) of the text, which can become handy during the programming. Furthermore, it sets auto-wrap of text when a single row is longer than the displaying screen.

```
set number
set cursorline
set wrap
```

The following command opens a "menu" when using cmdline mode, making it easier to key in commands.

```
set wildmenu
```

Many users in the community have posted their recommended *Vim* user profile configuration online, such as on *GitHub*. For the convenience of the reader, in the rest of the notebook, we will assume that **no re-map of keys combinations or shortcuts** are implemented, when introducing the commands

Notice that the configurations introduced in this section can also be activated with the *Vim* already started. Simply type: to switch from normal mode to cmdline mode, then key in the configuration. For example, :syntax on to activate the syntax display.

3.3.3 Plug Tools

In the Linux community, many plug tools have created to add useful features for *Vim.* As a demonstration, in this section *vim-plug*, a light-size vim plugin

management tool created on GitHub, is used to install selected Vim plugins. Details about vim-plug can be found at GitHub under junegunn/vim-plug.

Following the instruction given by GitHub under junegunn/vim-plug, to use vim-pluq on Linux, the very first step is to use cURL, a command-line tool for transferring data specified with URL syntax (very likely to be builtin to the user's Linux distribution), to download vim-plug. To confirm cURLinstallation, use the following command in Linux shell

```
$ apt-cache policy curl
```

and if cURL is installed, the shell is expected to return somthing like

```
Installed: 7.68.0-1ubuntu2.7
Candidate: 7.68.0-1ubuntu2.7
Version table:
*** 7.68.0-1ubuntu2.7 500
      500 http://cn.archive.ubuntu.com/ubuntu focal-updates/
          main amd64 Packages
      500 http://security.ubuntu.com/ubuntu focal-security/
          main amd64 Packages
      100 /var/lib/dpkg/status
   7.68.0-1ubuntu2 500
      500 http://cn.archive.ubuntu.com/ubuntu focal/main amd64
           Packages
```

With cURL installed, use the following in the shell to install vim-plug

```
$ curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
   https://raw.githubusercontent.com/junegunn/vim-plug/master/
       plug.vim
```

In the beginning vimre, add the following to indicate the plugins to be installed. Here as an example, vim-airline/vim-airline and joshdick/onedark.vim are installed, the first of which adds a status line at the bottom of the Vim window, and the second adds a popular color scheme "onedark".

```
call plug#begin()
Plug 'vim-airline/vim-airline'
Plug 'joshdick/onedark.vim'
call plug#end()
```

Finally, reload *vimrc*, then run: PlugInstall in cmdline mode to install the plugins.

```
1 Milliam Shakespeare (bapt. 26 April 1564 - 23 April 1616) was an English pla
ywright, poet and actor, widely regarded as the greatest writer in the Engli
sh language and the world's greatest dramatist.
2 He is often called England's national poet and the "Bard of Avon" (or simply
"the Bard").
```

FIGURE 3.3

A piece of text of "William Shakespeare", for demonstration.

3.4 Basic Operations in Vim

In normal mode, the most frequently used operation is probably u, which stands for undo. Other commonly used operations, such as delete, cut, copy, paste, replace and search, are mostly done in normal mode through shortcut keys. For example, dd delete (cut) the entire row at the cursor and p paste the content in the clipboard to the cursor position. For beginners, remembering shortcut keys can be difficult. In such case, it is suggested looking for the consistent patterns of the different commands, instead of brute-force remembering the operations.

Many *Vim* shortcut keys in normal mode has the following structure, namely an operator command followed by a motion command, as shown below

<operator><motion>

The operator command tells *Vim* what to do (say, copy), and the motion command tells the applicable range of the operation (say, a row, or a word, or a character). Some operator commands may work alone without motion commands.

3.4.1 Cut, Change, Copy and Paste

The following lines taken from Wikipedia under "William Shakespeare" is used as an example to demonstrate delete/cut, change, copy and paste functions. In the text file, each sentence takes a new row as given by Figure 3.3.

William Shakespeare (bapt. 26 April 1564 – 23 April 1616) was an English playwright, poet and actor, widely regarded as the greatest writer in the English language and the world's greatest dramatist.

He is often called England's national poet and the "Bard of Avon" (or simply "the Bard").

To quickly delete/cut a single character, use either x and X to delete the character at the cursor and previous to the cursor respectively. In summary, x and X play like delete and backspace respectively in other text editors.

To delete multiple characters, one way is to press x or X multiple times. Alternatively, it is possible for Vim to automatically repeat the procedure. For example, 20x tells Vim to perform x for 20 times. The same applies for other operators or motions commands. For example, 101 executes 1 for 10 times, making the navigation faster.

Operator d also deletes the contents of the text, but it requires a motion command and can be used more flexibly. The motion shall tell *Vim* what to delete/cut.

For example, d1 deletes one character to the right, i.e. deletes the character at the cursor just like x. Likewise, dh deletes one character to the left just like X. Similarly, d201 deletes 20 characters to the right, where "201" as a whole plays as the motion of "20 characters to the right". A combination by using things like 5d41 also works, since $20 = 5 \times 4$.

Command d can be used even more flexibly. For example, by using word-related motions, d can delete/cut by words instead of by characters. Move the cursor to the beginning of a word, (for example, "S" in "Shakespeare"), use dw to delete the word. The word motion w is similar with 1, except that 1 directs to the next character, while w directs to the beginning of next word. Similarly, b directs to the beginning of the current/previous word. Thus, db can be used to delete word to the left. Examples d10b, 10db, d20w, 5d4w can be used to delete multiple words at a time. Motions w and b can also be used to navigate in the text just like 1 and h.

When in the middle of a word, dw will delete the characters from the cursor to the beginning character of the next word. For example, if the cursor is currently at "k" in "Shakespeare", dw will delete "kespeare" (notice that the space between "Shakespeare" and "(bapt." will also be deleted). To delete from the beginning of the word instead, you can use b first to navigate back to the beginning of the word. Alternatively, use "inner-word" motion iw to indicate that the whole word of the cursor shall be deleted.

In addition to character-related motions h, 1 and word-related motions b, w, there are similar motions for sentence ((previous),) (next) and paragraph { (previous), } (next). There are also inner-sentence motion is, inner-paragraph motion ip, inner-quotation motion i', i", i' and inner-block motion i(, i<, i{, and many more. For example, when the cursor is at "A" of "26 April 1564", di(will delete everything inside "()", i.e. deleting "bapt. 26 April 1564 - 23 April 1616".

To conclude, the operators and motions introduced so far are listed in Tables 3.3 and 3.4. Notice that motions aw, as, ap are also given in the table. They are similar with their corresponding iw, is, ip except that when deleting, the consequent blank space (for word and sentence) or blank row (for paragraph) will also be deleted.

To change contents, use operator c, which works the same way as d but it automatically switch to insert mode at where the content is removed.

To copy a piece of text to clipboard, use y (stands for "yank") followed by

TABLE 3.3

Commonly used operators related to delete/cut, change, copy and paste.

	, , , , ,
Operator	Description
x	Delete/Cut the character at cursor.
- <u>X</u>	Delete/Cut the character before cursor.
dd	Delete/Cut the entire row.
d	Delete/Cut selected text according to the motion command.
cc	Change the entire row.
c	Change selected text according to the motion command.
уу	Copy the entire row.
-	Copy selected text according to the motion command.
p	Paste clipboard to the cursor.

TABLE 3.4

Commonly used motions.

Motion	Description
h, 1	One character to the left or right.
j, k	One row to the up or down.
b, w	One word to the previous or next.
(,)	One sentence to the previous or next.
{,}	One paragraph to the previous or next.
iw, is, ip	inner-word, inner-sentence, inner-paragraph.
aw, as, ap	a word, a sentence, a paragraph (including the end blank).
i', i", i'	inner-quotation for different types of quotations.
i(, i<, i[, }	inner-block for different types of brackets.
0	Beginning of the row.
\$	Ending of the row.
	Beginning of the text.
- <u>G</u>	Beginning of the last row of the text.

Text File Editing 31

its associated motion to indicate the range of text. The motions also follow Table 3.4.

To paste the text in the clipboard to the cursor, use p. No motion is required.

In addition Table 3.4, another commonly used type of motion is to "find by character". For example, consider the following row of text. The cursor is currently at letter "A".

ABCDEFG; HIJKLMN; OPQ; RST; UVW; XYZ

In normal mode, using f followed by a character will navigate the cursor to the nearest corresponding character that appears in the text. For example, fG will move the cursor to letter "G". Similarly, f; will move the cursor to the ";" between "G" and "H". Key in f; again and the cursor will move to ";" between "N" and "O". From here key in 2f: and the cursor will go to ";" between "T" and "U", as it is equivalent to executing f; twice. If df; is used when the cursor is at letter "A", "ABCDEFG;" will be deleted.

3.4.2 Search in the Text

It is common to search and highlight a particular word or phrase in the text. To make the searching result highlighted, add the following line to the user profile *vimrc*.

```
set hlsearch
exec "nohlsearch"
set incsearch
set ignorecase
```

where hlsearch enables highlighting all matching results in the text, and incsearch enables highlighting texts as the keyword is keying in. Vim remembers the keyword from the previous search and may automatically hight them in the text on a new session, which can be problematic sometimes. The command exec "nohlsearch" (exec command in the user profile make Vim execute that command when starting a new session) that comes after set hlsearch resolves this issue by forcing Vim to clear its memory. Finally, ignorecase allows case insensitive while searching.

In normal mode, use /<keyword> to search keywords or phrase. With the above setup, searching for "he" using /he leads to the following result given in Fig. 3.4. From Fig. 3.4, it can be seen that all appearances of "he" (case insensitive) is highlighted, and the cursor is automatically moved to its first appearance, i.e. "he" in "and the world's greatest dramatist". Click Enter to confirm the searching content. Use n and N to navigate the cursor from the highlighted results. Notice that at this point, n and N can be used as motion together with delete/cut, change and copy as given in Table 3.3.

To quit searching, use *cmdline* command :nohlsearch in normal mode and the highlights shall be gone. For convenience, people may prefer to map it with a customized shortcut key as well, for example in *vimrc* use

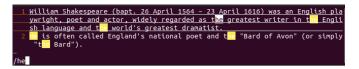


FIGURE 3.4

Search "he" in the piece of text of "William Shakespeare".

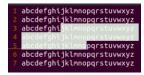


FIGURE 3.5

An example of visual mode where a block of text is selected.

noremap <Space> :nohlsearch<CR>

so that Space can be used to clear the search highlights.

3.5 Visual Modes of Vim

The use of a mouse makes selecting a block of text very intuitive. In most text editors, the selected text will be highlighted, as if the cursor expands from one character to the entire block of text. Sequentially, operations such as delete and copy can be performed on the selected text.

The three visual modes of *Vim*, namely "visual", "visual-line" and "visual-block", provide similar experience where the user can select and highlight a block of text.

Use v to enter the visual mode, then navigate the cursor to select a block of text. This allows the user to select text between any two characters. An example is given by Fig. 3.5. Alternatively, use V to enter the visual-line mode where multiple lines can be easily selected, and use <ctrl>+v to enter visual-block mode to select a rectangular block of text.

In any of the above visual mode, use :normal + <operation> to execute operation(s) form the normal mode. This allows convenient editing of multiple lines of text all together.

Text File Editing

FIGURE 3.6

Vim (with user's profile customization as introduced in this chapter).

3.6 Vim Macros

Vim macros can become handy for frequent and repetitive works.

...

3.7 Other Text Editors

Apart from Vim, many other text editors are also widely used in Linux, each with different features. For demonstration purpose, Vim and other text editors are used to open a bash shell script that calculates the first 10 elements of fibonacci series.

FIGURE 3.7

Nano.

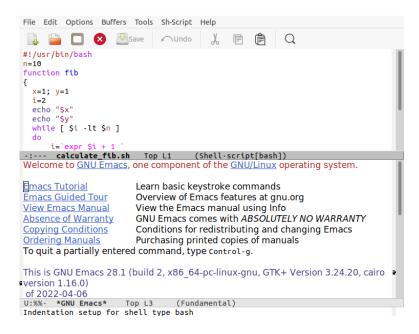


FIGURE 3.8

Emacs.

Text File Editing

```
calculate_fib.sh
        Open
                                            Save
1#!/usr/bin/bash
 2 n=10
 3 function fib
4 {
 5
   x=1; y=1
 6
    i=2
    echo "$x"
 7
    echo "$y"
    while [ $i -lt $n ]
10
    do
        i=`expr $i + 1 `
11
        z=`expr $x + $y `
echo "$z"
12
13
14
        x=$y
15
        y=$z
16
    done
17 }
18 r=`fib $n`
19 echo "$r"
20
           sh ▼ Tab Width: 8 ▼ Ln 1, Col 3 ▼ INS
```

FIGURE 3.9

Gedit.

FIGURE 3.10

 $Visual\ Studio\ Code.$

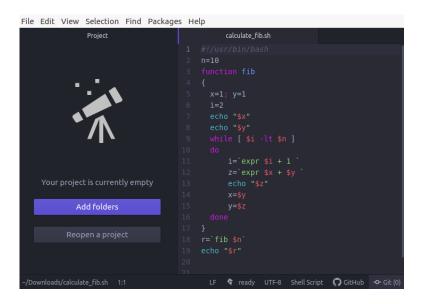


FIGURE 3.11

Atom.

4

File Management

CONTENTS

4.1	Filesystem Hierarchy Standard		37
4.2	Commonly Used File Management Commands		39
	4.2.1	Print Working Directory	41
	4.2.2	List Information about the Files	41
	4.2.3	Create an Empty or a Simple Text File	42
	4.2.4	Create an Empty Directory	43
	4.2.5	Move, Copy-and-Paste, and Remove Files and	
		Directories	43
	4.2.6	Use of Metacharacters	45
4.3	Access	Control List	45
	4.3.1	Change Ownership and Group of a File or Directory	46
	4.3.2	Change Permissions of a File or Directory	46
4.4	Search	through the System	48
	4.4.1	Look for a Command	48
	4.4.2	Look for a File by Metadata	48
	4.4.3	Look for a File by Content	49

File management is a big portion of OS functionality. In Linux, each device (such as a printer) is treated and managed as a file, and Linux uses a tree hierarchy to manage devices and files. This chapter introduces the filesystem hierarchy and commonly used file management commands.

4.1 Filesystem Hierarchy Standard

The root directory is denoted by a single forward slash "/". All sub directories or files can be located by its full path, which looks like the following

/<subdirectory>/<subsubdirectory>/.../<directoryname>
/<subdirectory>/<subsubdirectory>/.../<filename>

where the first / in each row represents the root directory, and sequential / represents entering a subdirectory.

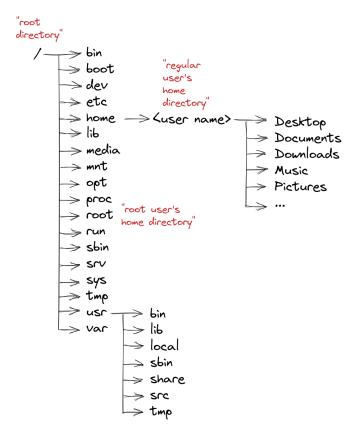


FIGURE 4.1 Linux file system hierarchy.

Upon Linux installation, a file hierarchy is by default created under the root directory. A user can create new files under this hierarchy framework, but should not change the framework itself. The hierarchy is given in Fig. 4.1. Notice that "/" in the figure, as introduced, stands for the root directory, and "root" in the figure is a subdirectory under / whose directory name is "root" and it is used store root user related documents. They are two different directories.

A (regular) user's home directory is often located at /home/<user name>. When logging in as a regular user, his home directory is shorten by the tilde ~ for convenience. Hence, for example ls ~ will list down the files and directories under his home directory.

As can be seen from Fig. 4.1, the hierarchy contains quite a few predetermined subdirectories, each has some unique purpose. For convenience of illustration, these subdirectories are categorized by functionalities and depressibilities given in Fig. 4.2. Notice that this categorization is rough and

	"Used by the OS"	"Used by all users"	"Used by a specific user"
	Administration (System) Level	All-Users Level	Individual User Level
Executable	/bin /sbin	/usr/bin /usr/sbin	/home/Kuser name>
Library	∕іњ	/usr/lib	/home/Kuser name>
Data / Program Storage	/opt /var	/usr/local /usr/share /usr/src	/home/Kuser name>
Device	/dev /media /mnt		
Configuration	/etc		/home/Kuser name>
System	/boot /proc /sys		
Other	/tmp /usr /root	/usr/tmp	

FIGURE 4.2

A rough categorization of commonly used directories in Linux file hierarchy standard.

may not reflect the truth for all applications. For example, the commonly used command ls may appear under /bin or /usr/bin depending on the Linux distributions.

A brief introduction to the directories are summarized in Table 4.1.

Linux file hierarchy standard differs from MS-DOS and Windows in several ways. Firstly, Linux stores all files (regardless of their physical location) under root directory, while Windows uses drive letters such as C:\, D:\ to distinguish different hard drives. Secondly, Linux uses slash (/) to separate directory names, e.g. /home/username while Windows uses back slash (\), e.g. C:\Users\username. Lastly, Linux uses "magic numbers" to tell file types and permissions and ownership to tell whether a file is executable, while Windows (almost always) uses suffixes to tell file types and distinguish executables. Distinguishing file types using magic numbers can be more reliable than using suffixes, though a bit less intuitive.

Magic numbers of a file refer to the first few bytes of a file that are unique to a particular file type, for example, PNG file is hex 89 50 4e 47. Linux compare the magic numbers of a file with an internal database to decide the file types and features.

TABLE 4.1

Introduction to commonly used directories in Linux file hierarchy standard.			
Directory	Description		
/bin, /sbin	Executables used by the OS, the administrator, and		
	the regular users.		
/lib	Libraries to support /bin and /sbin.		
/usr/bin, /usr/sbin	Executables used by the administrator and the reg-		
	ular users.		
/usr/lib	Libraries to support /usr/bin and /usr/sbin.		
/opt	Application software installed by OS and adminis-		
	trator for all users.		
/var	Directories of data used by applications.		
/usr/local	Application software installed by administrator for		
	all users.		
/usr/share	Architecture-independent sharable text files for ap-		
	plications.		
/usr/src	Source files or packages managed by software man-		
	ager.		
/dev	Files representation of devices, such as CPU, RAM,		
	hard disks.		
/media	System mounts of removable media.		
/mnt	Manual mounts of devices.		
/etc	Configuration files for OS, users, and applications.		
/boot	Linux bootable kernel and initial setups.		
/proc	System resources information.		
/sys	Linux kernel information, including a mirror of the		
	kernel data structure.		
/tmp, usr/tmp	Temporary files.		
/root	Root user's home directory.		
/home/ <user name=""></user>	A regular user's home directory, containing exe-		
	cutables, configurations and files specifically belong		
	to this user.		

TABLE 4.2Commonly used commands to navigate in the Linux file system.

Commonly used commands to havigate in the Linux me system.		
Command	Description	
pwd	Print working directory.	
ls	List the subdirectories and files (and their detail information)	
	in a given directory.	
touch	Create an empty file.	
mkdir	Create an empty subdirectory.	
mv	Move (cut-and-paste) a directory or a file; change name of a	
	directory or a file.	
cp	Copy-and-paste a directory or a file.	
rm, rmdir	Remove a directory or a file (not to Trash, but just gone).	
chmod	Change permission.	
chown	Change ownership.	

4.2 Commonly Used File Management Commands

Some of the most widely used file management commands are summarized in Table 4.2. Notice that chmod and chown are administration related commands that change the accessibility of a directory or a file, and will be introduced in a later sections together with the Linux permission system. The rest commands are categorized and introduced in the following subsections.

4.2.1 Print Working Directory

As introduced earlier in Chapter 2 Table 2.1, HOME, PWD and OLDPWD are three default environmental variables used to store the home directory, the current directory and the previous directory of the shell, respectively. Therefore, to print the current working directory in the console, use command echo \$PWD. Alternatively, just use pwd which has the safe effect, as follows.

\$ pwd

4.2.2 List Information about the Files

As one of the most frequently used commands, 1s lists down information about the files in the current directory (or any other directory if specified), and by default sort the entries alphabetically. Features can be specified for the items to be listed, to make the result more selective. The user is able to decide what information to be displayed, such as file name, file access control list, etc., as follows.

\$ ls [OPTION] [PATH]

```
gurobi.log
Music
                                                                                      octave-workspace
Pictures
               Documents
                                           eclipse-workspace
drwxrwxr-x
                   sunlu sunlu 4096 Dec 21 01:54 anaconda3
                   sunlu sunlu 4096
                                           Арг
                   sunlu sunlu 4096
sunlu sunlu 4096
                                                    10:42 Documents
14:37 Downloads
 rwxr-xr->
 rwxr-xr-x
                   sunlu sunlu 4096 Jun 15 10:38 Dropbox
sunlu sunlu 4096 Dec 21 12:34 eclipse
lrwx-
                                                      2021 eclipse
2021 gurobi
 rwxrwxr->
                   sunlu sunlu 4096
                   sunlu sunlu 4096
 rwxrwxr->
                   sunlu sunlu
                                     488 Mar
                                                      2021 gurobi.log
2021 Music
                   sunlu sunlu
                   sunlu sunlu
sunlu sunlu
                                                    16:48 octave-workspace
08:58 Pictures
                                      43 May
 rwxr-xr->
                   sunlu sunlu 4096
                                                      2021 Public
2021 R
                   sunlu sunlu
                   sunlu sunlu 4096
sunlu sunlu 4096
                                                    14:03 snap
2021 Templates
                   sunlu sunlu 4096 Feb
sunlu sunlu 4096 Jan
                                                       2021
MATLAB
 ownloads:
calculate_fib.sh
```

FIGURE 4.3

List down information of files and subdirectories in the current working directory.

An example is given in Fig. 4.3 below. By default, command 1s alone shows only the name of files and subdirectories (excluding hidden files and subdirectories). With the additional arguments (option, as given in the syntax above), more information can be displayed. For example in Fig. 4.3, the -1 argument displays the information in long listing mode, which includes the owner and access control list information. More details about files and directories access control list are given in later part of this section.

More information can be found by reading the 1s command manual, which is accessible via 1s --help. Some commonly used 1s arguments are summarized in Table 4.3. It is also possible to use a combination of arguments in a single line of command. For example, 1s -al aggregates the effects of using 1s -a and 1s -1.

Notice that some Linux distributions may come by default an alias about ls, which usually helps to displays the information in a clearer manner. For example, when ls='ls --color-auto' is used, the displayed content will be colored based on the type of the files and subdirectories.

4.2.3 Create an Empty or a Simple Text File

To create an empty file in the current working directory, simply use touch followed by the path of the file (including file name) as follows.

```
$ touch [OPTION] PATH
```

TABLE 4.3 Commonly used arguments and their effects for ls command.

commonly about arguments and their encous for to command.			
Directory	Description		
-a,all	Include hidden files and subdirectories in the display,		
	including current directory "." and parent directory		
	"" in the list.		
-A,almost-all	Include hidden files and subdirectories in the display,		
	excluding "." and "".		
-C,color[=WHEN]	Colorize the output.		
-1	Use a long listing format.		
-s,size	Print the allocated size of each file, in blocks.		
-S	Sort the displayed content.		
-t	Sort by modification time.		

For example,

\$ touch ~/test

will create an empty file "test" under the user's home directory. If only the name of the file is given, it will by default create the file under the current working directory. Notice that if a file or subdirectory name starts with ".", it will be treated as a hidden file or subdirectory automatically.

Multiple empty files will be created if multiple paths are given in the command separated by spaces.

To create a simple text file, such as a text file with a single line of contents inside, consider using echo command with > as follows. It is more convenient than using *Vim* for the same task, although also possible.

\$ echo 'content' > PATH

For example,

\$ echo '<html><body><h1>Hello world!</h1></body></html>' > ~/
test.html

creates a simple static HTML web page that says "Hello world!" in the home directory.

4.2.4 Create an Empty Directory

Similar with touch, use mkdir followed by the path of the directory (including directory name) to create a directory as follows.

\$ mkdir [OPTION] PATH

TABLE 4.4

Commonly used arguments and their effects for mv and cp command.

	Directory	Description
	-b	Make a backup before overwrite.
-	-u	Overwrite only when source target item is newer than the target
		path item.
-	-i	Prompt before overwrite.
-	-f	Do not prompt before overwrite.

4.2.5 Move, Copy-and-Paste, and Remove Files and Directories

To move a file or a directory from an existing PATH to a new PATH, simply use mv command as follows.

\$ mv [OPTION] SOURCE_PATH TARGET_PATH

Different from the conventional cut-and-paste, while moving the item, it is possible to also rename the item simultaneously. For example,

\$ mv ~/dog.png ~/Pictures/puppy.png

will not only move the file dog.png in the home directory to the subdirectory Pictures, but also chance the file name to puppy.png. For this reason, mv can also be used to rename an item rather than moving the item, just by "move" it to the same directory but with a differen name.

Some commonly used arguments of mv is summarized in Table 4.4, many of which concerns about the case where there is already an existing item with the identical name in the target path.

The copy-and-paste command cp works similar with the move command mv, except that it will not remove the item from the source path. Similar syntax applies to cp as follows, and arguments in Table 4.4 also apply to cp.

\$ cp [OPTION] SOURCE_PATH TARGET_PATH

To permanently delete an item, use rm command as follows.

\$ rm [OPTION] PATH

For safety, usualy when using rm, the OS will keep prompting messages asking user to confirm whether to permenantly delete an item or not. In some OS setups, it is by default forbidden to delete a directory, unless all files and subdirectories in that file have been priorily removed. The following arguments in Table 4.5 can be used to change the setup.

It is possible though, that removed items using rm be recovered by expertise. For greater assurance that the deleted contents are truly unrecoverable, consider using shred which can physically overwrite the portion of hardware drive where the item is located. More details of shred can be found by using

TABLE 4.5

Commonly used arguments and their effects for rm command.

Directory	Description
-f	Ignore nonexistent files and arguments and do not prompt.
-r	Remove directories and their contents recursively.
-i	Prompt before every removal.
-d	Remove empty directories.

TABLE 4.6

Commonly used metacharacters.

Directory	Description
*	Matches any number of characters.
?	Matches one character.
[]	Matches characters given in the square bracket, which can in-
	clude a hyphen-separated range of characters.

\$ shred --help

4.2.6 Use of Metacharacters

When performing moving, copying, removing or otherwise acting on files, metacharacters can be used to make the work more efficient sometimes. For example, ls a* will list all items in the current directory that starts with letter "a". Commonly used metacharacters are summarized in Table .

4.3 Access Control List

Each file or directory in the Linux OS is assigned with an owner and a permission list. The permission list prevents unauthorized persons to access the item. The permission list of a file can be checked by using 1s -1. An example is given in Fig. 4.3.

The first column of the output in Fig. 4.3 gives the type and permission of the item. The leading $\tt d$ and – indicate subdirectory and regular file respectively. Other commonly seen indicators are 1 for a symbolic link, $\tt b$ for a block device, $\tt c$ for a character device, $\tt s$ for a socket and $\tt p$ for a named pipe.

Following by the item type indicator is the 9-bit permission that may look like rwxrwxrwx. The characters r, w and x stand for three types of permissions "read", "write" and "execute" respectively. An explanation to these permissions is summarized in Table 4.7 and more details can be found in the 1s

TABLE 4.7

Three types of permissions.			
Directory	Description		
r	View what is in the file or directory.		
w	Change file contents; rename file; delete file. Add or remove files		
	or subdirectories in a directory.		
x	Run a file as a program. Change to the directory as the current		
	directory; search through the directory; access metadata (file		
	size, etc.) of files in the directory.		

command manual available using ls --help. The 9-bit permission of an item indicates the permissions of 3 types of users to the item, the first 3 bits the file owner, the middle 3 bits the file group, and the last 3 bits other users. If any bit in the 9-bit permission is overwritten by a dash -, it means that the associated permission for the associated users is banned.

Commands chown and chmod can be used to change the ownership and 9-bit permission of an item respectively. Details are given in following subsections.

4.3.1 Change Ownership and Group of a File or Directory

Administrative privilege is required to run chown command to change the ownership and group of a file or a directory as follows.

chown [OPTION] [NEW_OWNER] [:NEW_GROUP] PATH

For example, in Fig. 4.4,

\$ sudo chown root:root calculate_fib.sh

is used to change the ownership and group of file calculate_fib.sh from sunlu to root. Use 1s with longlist to check the ownership of a file. Notice that elevated privilage is required to change its ownership, otherwise the request will be rejected as shown in Fig. 4.4.

4.3.2 Change Permissions of a File or Directory

Both the owner of a file or directory and the users with administrative privilege can change the 9-bit permission of the file using chmod as follows. The 9-bit permission, in this context, is called the mode of the file.

\$ chmod [OPTION] MODE PATH

For example, in Fig. 4.5, g-w is used to subtract "writing" permission from "group", and go+w is used to add "writing" permission to 'group" and "other", respectively. Here, u, g and o represents "user" (owner), "group" and "other", and r, w and x, "read", "write" and "execute", respectively. Alternatively, 3-

```
sunlu@SUNLu-Laptop:~/Downloads$ ls -lA
total 8
-rw-rw-r-- 1 sunlu sunlu 221 Jun 23 13:36 calculate_fib.sh
-rw-rw-r-- 1 sunlu sunlu 48 Jun 15 16:24 test.html
sunlu@SUNLu-Laptop:~/Downloads$ chown root:root calculate_fib.sh
chown: changing ownership of 'calculate_fib.sh': Operation not permitted
sunlu@SUNLu-Laptop:~/Downloads$ sudo chown root:root calculate_fib.sh
sunlu@SUNLu-Laptop:~/Downloads$ ls -lA
total 8
-rw-rw-r-- 1 root root 221 Jun 23 13:36 calculate_fib.sh
-rw-rw-r-- 1 sunlu sunlu 48 Jun 15 16:24 test.html
```

FIGURE 4.4

Change ownership and group of a file.

```
sunlu@SUNLu-Laptop:~/Downloads$ ls -l
total 8
-rw-rw-r-- 1 sunlu sunlu 221 Jun 23 13:36 calculate_fib.sh
-rw-rw-r-- 1 sunlu sunlu 48 Jun 15 16:24 test.html
sunlu@SUNLu-Laptop:~/Downloads$ chmod g-w calculate_fib.sh
sunlu@SUNLu-Laptop:~/Downloads$ ls -l
total 8
-rw-r--r-- 1 sunlu sunlu 221 Jun 23 13:36 calculate_fib.sh
-rw-rw-r-- 1 sunlu sunlu 48 Jun 15 16:24 test.html
sunlu@SUNLu-Laptop:~/Downloads$ chmod go+w calculate_fib.sh
sunlu@SUNLu-Laptop:~/Downloads$ ls -l
total 8
-rw-rw-rw- 1 sunlu sunlu 221 Jun 23 13:36 calculate_fib.sh
-rw-rw-r-- 1 sunlu sunlu 48 Jun 15 16:24 test.html
sunlu@SUNLu-Laptop:~/Downloads$ chmod 664 calculate_fib.sh
sunlu@SUNLu-Laptop:~/Downloads$ ls -l
total 8
-rw-rw-r-- 1 sunlu sunlu 221 Jun 23 13:36 calculate_fib.sh
-rw-rw-r-- 1 sunlu sunlu 48 Jun 15 16:24 test.html
```

FIGURE 4.5

Change 9-bit permission (mode) of a file.

4.4 Search through the System

There are roughly 3 types of searching commands that a user would use frequently:

- Look for the location of a command using its name
- Look for the location of a file using its name (and other metadata such as size, permission, etc.)
- Look for the location of a file using a portion its content

There can be multiple ways to reach each of the above goals. Details are as follows.

4.4.1 Look for a Command

...

4.4.2 Look for a File by Metadata

Many Linux distributions come with built-in command locate that can be used to quickly locate a file by (a fraction of) its path as follows. Notice that as long as a file or directory's full path contains the searched content, there is a chance that it will appear in the result. As a result, if the name of a directory is used for searching, all the items in that directory will likely to appear in the result (as their full paths contain the name of the directory).

\$ locate PATH

The mechanism behind locate is that behind the users' eyes, the OS runs updatedb in the background usually once a day to update an internal database that gathers the names of files, and locate searches the database for a file. Notice that locate may fail to find recently added files if it has not been added to the database by updatedb. Besides, not all files are covered by updatedb

```
sunlugSUNLU-Laptop:-$ ls
anaconda3 Documents Dropbox eclipse-workspace gurobi.log octave-workspace Public snap texmf
Desktop Downloads eclipse gurobi Music Pictures R Templates Videos
sunlugSUNLU-Laptop:-$ locate Downloads
/home/sunlu/Joonnloads
/home/sunlu/.config/google-chrome/Webstore Downloads
/home/sunlu/.local/share/applications/_home_sunlu_Downloads_eclipse-installer_eclipse-inst-jre-linux64_eclipse-installer_.desktop
/home/sunlu/Joonnloads/latsi.html
sunlugSUNLu-Laptop:-$ cd Music
sunlugSUNLu-Laptop:-$ cd Music
sunlugSUNLu-Laptop:-$ locate Downloads
/home/sunlu/Joonnloads/calculate_fib.sh
/home/sunlu/Joonnloads/applications/_home_sunlu_Downloads
/home/sunlu/.local/share/applications/_home_sunlu_Downloads
/home/sunlu/.local/share/applications/_home_sunlu_Downloads/calculate_fib.sh
```

FIGURE 4.6

Search for files and directories using locate.

by default, and a configuration file at /etc/updatedb.conf determines which files to be covered by updatedb. It is also worth mentioning that it will take some time to run updatedb for the first time, as it has a lot of things to add to the database during its initial run.

One may get confused by commands locate and mlocate. The concepts of the commands are very similar, and when both commands are available on a machine, mlocate functions by default even the user types locate.

An example of using locate is given in Fig. 4.6. It can be seen from this example that the searching is done globally and does not rely on the current working directory.

It is worth mentioning that for safety and privacy reasons, locate only shows the items that the user would be able to detect manually using cd and ls in the first place. Therefore, a regular user cannot locate any file under /root our other users' home directory using this method.

A more common and widely accepted way of looking for a file by its variety of attributes is using find.

4.4.3 Look for a File by Content

...

5

Software Management

CONTENTS

5.1	Linux Kernel Management	51
5.2	General Introduction to Linux Package Management Tools	51
5.3	Installation of Software	51
5.4	Software Upgrade	51
5.5	Uninstallation of Software	52

5.1 Linux Kernel Management

"nobreak

5.2 General Introduction to Linux Package Management Tools

"nobreak

5.3 Installation of Software

 ${\rm ``nobreak'}$

 $^{{\}rm ``nobreak'}$

5.4 Software Upgrade

 ${\rm ``nobreak'}$

5.5 Uninstallation of Software

6

Process Management

CONTENTS

6.1	General Introduction to Process	53
6.2	Running Process Management on Linux	53

 ${\rm ``nobreak'}$

6.1 General Introduction to Process

"nobreak

6.2 Running Process Management on Linux

$\begin{array}{c} {\rm Part~II} \\ {\rm Linux~Advanced} \end{array}$

Administration Basics

CONTENTS

7.1 Introduction to Linux Administration 57

 ${\rm ``nobreak'}$

7.1 Introduction to Linux Administration

Account Management

CONTENTS

8.1 Quick Account Management 59

 ${\rm ``nobreak'}$

8.1 Quick Account Management

$Disk\ Management$

CONTENTS

 ${\rm ``nobreak'}$

9.1 Hard Drive Check

Advanced Software Management

CONTENTS

10.1	Git for	Software Version Control and Joint Development	63
	10.1.1	Brief Introduction to Git	63
	10.1.2	Local Repository Operations	63
	10.1.3	Remote Repository Operations	63

10.1 Git for Software Version Control and Joint Development

 ${\rm ``nobreak'}$

10.1.1 Brief Introduction to Git

"nobreak

10.1.2 Local Repository Operations

"nobreak

10.1.3 Remote Repository Operations

 $^{{\}rm ``nobreak'}$

Part III Linux Server Management

Virtualization and Containerization

CONTENTS

11.1	Virtual	Machine	70
11.2	Contair	ner	70
11.3	Docker	Basics	70
	11.3.1	Docker Installation	71
	11.3.2	Launch a Container	72
	11.3.3	Access and Manage a Container	74
	11.3.4	Publish a Container	76
11.4	Docker	Images	76

One of the major differences between a server and a personal computer is that the server is usually shared among multiple users at the same time. Though working on the same physical server, a user would usually want a private working environment in the server not interrupted by other users. In other words, each and every user would want to "virtually" work on an independent and separated computer with his own CPU, RAM, I/O, OS, drives and hard disk storage, despite that the actual hardware is shared with others. This is done through *Virtualization*, which enables running multiple operating systems on a single physical server in an uninterrupted and logically separated manner. The virtually independent computer of such kind is often called a *virtual machine* (VM).

Deploying a new VM would generally consume considerably large amount of time and computational load, because it needs to load the OS in its first startup. Consider a case where there are hundreds of applications, each requiring running in a similar but separate environment. Launching a VM for each application can be a solution, but it can be expensive due to the time and computational burden. It would be batter to rather deploy only one VM (or physical server) with one OS, and put each application in a "container" with its own customized drives and configurations. A container is similar with a VM in the sense that it runs separately from other containers. However, a container is much "lighter" than a VM, thus is cheaper to launch and manage. This becomes possible thanks to containerization. It is worth mentioning that since a container contains all the configuration and minimum requirement information of the application, running a container on different platforms would

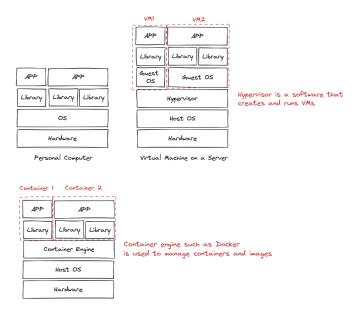


FIGURE 11.1 System architectures of PC, VM and container.

generate the same stable result. This is handy when comes to code transferring and cross-platform testing.

The similarity and differences of personal PC, VM, and container applications are summarized in Fig. 11.1.

An an analogy, think of running an APP as asking a restaurant to prepare a dish. The hardware is corresponding with kitchen (along with the cooktop, the frying pan, etc.) of the restaurant where every cooking procedure actually happens. The OS is corresponding with the cook, who uses the materials in the kitchen to make the dish. The OS needs associated drivers and libraries to run the APP correctly. The drivers and libraries are like the skill set expected from the cook in order to prepare the dish correctly. Therefore, installing and configuring drivers and libraries in the OS is like teaching the cook new skills (probably from a cookbook) so that he would know how to cook the dish. Finally, the APP is corresponding with the prepared dish.

In a traditional PC implementation, for each customer (user) or dish (APP), a new kitchen is constructed and its associated cook is hired. The cook is trained to master all necessary skills required by the customer or for the dish. A metaphor of the PC implementation is given in Fig. 11.2.

In a VM implementation, a larger and more capable kitchen is setup in advance. For each customer or a dish, a cook is hired. Each cook is trained with the skills necessary for his associated customer or dish. All cooks share the same kitchen. This implementation is more efficient than the previous "a

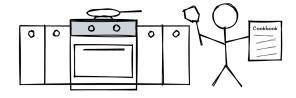


FIGURE 11.2

PC implementation: a cook in a kitchen.

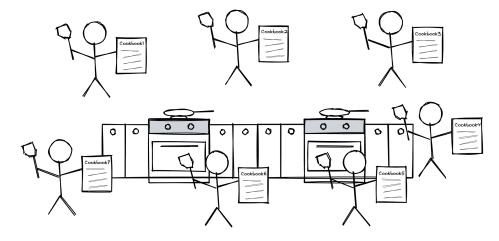


FIGURE 11.3

VM implementation: many cooks in a kitchen, each with a different cookbook.

cook in a kitchen" implementation in Fig. 11.2, as there is no need to scale up the kitchen for each new customer or APP. By sharing the resources among the cooks, it is more probable that the resources in the kitchen be used more effectively. A metaphor of the VM implementation is given in Fig. 11.3. This is indeed a popular implementation when comes to both enterprise-level server management and a local restaurant.

Deploy dedicated OS for each APP can be time consuming when the number of the APPs is large. This is also true in the restaurant, where it is very rare for each dish to have a dedicated cook. Instead, a cook usually handles a category of dishes that requires similar skill sets. A cook good at multi-task can handle many dishes by himself alone, as long as the dish recipes are provided. Of course, each dish will stay in its own frypan so that they would not affect each other. This is similar with a container implementation. The recipe for a dish, which describes the skill set of the dish, is called an "image" of the container that describes the drivers, libraries and basic configurations for the APP to run. The food cooked in the frypan is the APP. The frypan, the food, and the recipe, together form an instance of a container. A metaphor of

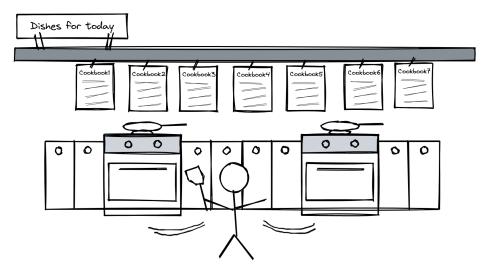


FIGURE 11.4

Container implementation: one in a kitchen, handling multiple dishes, each has a cookbook and stays in its own pan.

the container implementation is given in Fig. 11.4. Luckily, the cook (OS) is good at multi-tasking by nature, and with the help of a front desk manager (container management tool such as Docker), he can perform quite well.

A cookbook is a instruction of a dish. It guarantees that the same dish, even one day to be cooked in a different restaurant by a different cook, would taste the same. Similarly, the image of a container serves as the start-up instruction of the container, and it helps to maintain consistency of the performance of the APP even running on different machines between developers and machines, guaranteeing that a APP runs correctly on different machines. With an image, it is convenient to deploy similar container instances quickly.

11.1 Virtual Machine

11.2 Container

. . .

11.3 Docker Basics

"nobreak

11.3.1 Docker Installation

Docker engine is one of the most popular container management engine available online, and it is free of charge for personal non-commercial usage. To install Docker on a Linux machine, go to https://www.docker.com/ to for the instruction.

As an example, consider installing Docker engine on Ubuntu. Some of the key steps are summarized as follows.

Remove existing Docker engine, if any.

```
$ sudo apt-get remove docker docker-engine docker.io
```

\$ sudo apt-get remove containerd runc

Add Docker's official GPG key and set up the repository.

```
$ sudo apt-get update
```

- \$ sudo apt-get install ca-certificates curl gnupg lsb-release
- \$ sudo mkdir -p /etc/apt/keyrings
- \$ echo \
 - "deb [arch=\$(dpkg --print-architecture) signed-by=/etc/apt/
 keyrings/docker.gpg] https://download.docker.com/linux/
 ubuntu \
 - \$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/
 docker.list > /dev/null

Install Docker.

```
$ sudo apt-get update
```

\$ sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-compose-plugin

To test whether docker is installed correctly, run

\$ sudo docker run hello-world

and if everything is done correctly, a message started with "Hello from Docker!" will be displayed in the console, together with a brief introduction to how docker works.

Notice that to use docker commands, sudo privilege is required. To avoid typing **sudo** each time running a docker command, add the user to the docker group as follows.

\$ sudo usermod [user_name] -aG docker

In the rest of the section, sudo is neglected for docker commands.

11.3.2 Launch a Container

To run a container from an image, simply use

\$ docker run [image_name]

Docker will search the local and remote repository for the image, download the image if necessary, and run the container of that image. By default, after successful execution, the container will go into "Exited" status. To customize the container running, for example, assigning a name for the container, flags can be used. Details are introduced later.

To check images stored locally, use

\$ docker image ls

To check the status of a container, use

\$ docker container ls

or

\$ docker container ls -a

where -a indicates displaying both running and exited containers. Without -a, exited containers will not be displayed.

For example, consider running a container of *alpine* as follows. A screen shot is given in Fig. 11.5.

\$ docker run -it --name test-alpine alpine

where -i stands for interactive, which will keep the container's standard input (i.e., the console in this example) open so that the user can actively interract with the container. Option -t allocates a pseudo-TTY to the container, making the interactive interface a bit more user friendly. Finally, --name assign a name to the container. Without an assigned name, docker will randomly assign a name to the container.

It can be seen from Fig. 11.5 that once the container is started, the user can interact with the container using its shell, and perform actions such as upgrading the container, or deploy a web server, etc. While keep the container up running, open another terminal and use docker container 1s. The container test-alpine shall appear in the list, as shown in Fig. 11.6.

If exit from Fig. 11.5 (by using exit in the container), the container will transfer its status from up running to exited, as shown in Fig. 11.7.

It is also possible to launch a container and let it run in the background using -d flag. An example is given below.

\$ docker run -dt --name test-background-alpine alpine

```
sunlu@sunlu-laptop-ubuntu:-$ docker run -it --name test-alpine alpine
) # ls
oin dev etc home lib media mnt opt proc root run sbin srv sys tmp usr var
) # pwd
)
/ # pwd
// # whoami
root
/ # apk update
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/community/x86_64/APKINDEX.tar.gz
v3.16.0-302-g62bf0b8f5a [https://dl-cdn.alpinelinux.org/alpine/v3.16/main]
v3.16.0-304-g51632b3deb [https://dl-cdn.alpinelinux.org/alpine/v3.16/community]
0K: 17030 distinct packages available
/ # apk upgrade
(1/6) Upgrading alpine-baselayout-data (3.2.0-r20 -> 3.2.0-r22)
(2/6) Upgrading alpine-baselayout-data (3.2.0-r20 -> 3.2.0-r22)
(2/6) Upgrading alpine-baselayout (3.2.0-r22.pre-upgrade
(3/6) Upgrading alpine-baselayout-3.2.0-r22.pre-upgrade
(4/6) Upgrading libcrypto1.1 (1.1.10-r0 -> 1.1.1q-r0)
(5/6) Upgrading libcrypto1.1 (1.1.10-r0 -> 1.1.1q-r0)
(6/6) Upgrading libsvil.1 (1.1.10-r0 -> 1.1.1q-r0)
(6/6) Upgrading sclient (1.35.0-r13 -> 1.35.0-r14)
Executing busybox-1.35.0-r14.trigger
0K: 6 MlB in 14 packages
```

FIGURE 11.5

An example of running *apline* container, with interactive TTY and name *test-apline*.

```
sunlugsunlu-laptop-ubuntu:-5 docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
03b1039d4f4d alpine "/bin/sh" 28 minutes ago Up 28 minutes test-alpine
```

FIGURE 11.6

List the up running container test-apline.



FIGURE 11.7

List the exited container test-apline.

By changing -i to -d, the container runs in the background without accessing its interactive shell interface. The status of the container, after executing the above command, will stary up running and can be listed by docker container 1s.

In summary, selected key commands regarding launching a container is given in Table 11.1.

11.3.3 Access and Manage a Container

To start an existing (but exited) container, use

\$ docker start [container_name]

This command starts the exited container, and keep it up running in the background.

For a container running in the background, use docker exec to execute a shell command as follows.

\$ docker exec [container_name] [command]

To enable the TTY shell of a container running in the background, use

```
$ docker exec -it [container_name] [shell_name]
```

where depending on the environment of the container, the shell can be different. In the case of an *alpine* imgae based container, ash is the default shell. For a *ubuntu* image based container, on the other hand, usually bash should be specified.

There are multiple ways and protocols to interact with the file systems in a container, depending the I/O setup of the container. For a container running locally, docker cp can be conveniently used for file transfer between the container and the host machine as follows. From container to host machine:

\$ docker cp [container_name]:[source] [destination]

and from host machine to container:

\$ docker cp [source] [container_name]:[destination]

where [source] and [destination] refer to the path to the source and destination, respectively, located in the host machine or the container.

To stop or restart a container running in the background, use

- \$ docker stop [container_name]
- \$ docker restart [container_name]

respectively. When a container is stopped, it enters exited status.

To remove a stopped container, use

\$ docker container rm [container_name]

to remove a specific container, or

TABLE 11.1 Commonly used docker commands to launch a container.

Command Command Comma
lowed by the command. If the image cannot be found locally, it downloads the image from the remote repository automatically. Assign a random name to the container. Exit the container after execution. docker run -i Keep the standard input of the container open when launching the container. Launch the container in the background and keep it up running. Automatically remove the container when exiting. The removed container will not be listed in docker container will not be listed in docker container ls -a. This is usually used for testing and debugging. docker run -t Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
cannot be found locally, it downloads the image from the remote repository automatically. Assign a random name to the container. Exit the container after execution. The part of the container open when launching the container.
the image from the remote repository automatically. Assign a random name to the container. Exit the container after execution. Reep the standard input of the container. Launch the container in the background and keep it up running. Launch the container when exiting. The removed container will not be listed in docker container will not be listed in docker container ls -a. This is usually used for testing and debugging. Launch the container ls -a. This is usually used for testing and debugging. Launch the container ls -a. This is usually used for testing and debugging. Launch the container ls -a. This is usually used for testing and debugging. Launch the container ls -a. This is usually used for testing and debugging. Launch the container ls -a. This is usually used that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Launch the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
tory automatically. Assign a random name to the container. Exit the container after execution. Keep the standard input of the container open when launching the container.
name to the container. Exit the container after execution. Reep the standard input of the container open when launching the container.
name to the container. Exit the container after execution. Reep the standard input of the container open when launching the container.
Time
tainer open when launching the container. Daunch the container in the background and keep it up running. Automatically remove the container when exiting. The removed container will not be listed in docker container 1s -a. This is usually used for testing and debugging. Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
tainer open when launching the container. docker run
Launch the container in the background and keep it up running. Automatically remove the container when exiting. The removed container will not be listed in docker container 1s -a. This is usually used for testing and debugging. Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
ground and keep it up running. Automatically remove the container when exiting. The removed container will not be listed in docker container 1s -a. This is usually used for testing and debugging. Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Allocate in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
Automatically remove the container when exiting. The removed container will not be listed in docker container 1s -a. This is usually used for testing and debugging. docker run -t Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. docker run restart Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
tainer when exiting. The removed container will not be listed in docker container 1s -a. This is usually used for testing and debugging. Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
container will not be listed in docker container 1s -a. This is usually used for testing and debugging. docker run -t Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. docker run restart Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
docker container 1s -a. This is usually used for testing and debugging. Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
usually used for testing and debugging. Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
docker run -t Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
Allocate a pseudo-TTY. TTY stands for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
for "TeleTYpewriter". A simplified explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. docker run restart Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
explanation to -t is that it makes sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
sure that the I/O of the container follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
follows the typical terminal format. The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
The flag usually comes with the flags -i or -d, to form -it or -dt. Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations include restart no (do not automatically restart container when it exits),
-i or -d, to form -it or -dt. docker run -restart Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
docker run restart Indicates in which condition the container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
container needs to restart. This is usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
usually used on containers running in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
in the background, i.e., it usually comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
comes with the flag -dt. Commonly used restart configurations includerestart no (do not automatically restart container when it exits),
used restart configurations includerestart no (do not automati- cally restart container when it exits),
restart no (do not automatically restart container when it exits),
cally restart container when it exits),
restart on-lallure[:Max_retrles
(restart if the container exits
with a non-zero exit status),
restart always (restart regard-
less of the exit status). Notice that
for a container withrestart flag,
it is still possible to stop and remove
the container manually.
docker runname Assign a name to the container.
docker image 1s — List local images.
docker container 1s — List up running containers.
docker container 1s -a List all containers.

\$ docker container prune

to remove all stopped containers.

To rename a container (without changing its container ID or anything else), use

\$ docker rename [old_container_name] [new_container_name]

To quickly check container status (CPU, memory usage, etc.) of an up running container or all up running containers, use

\$ docker stats [container_name]

where the container name can be specified as an option.

To list down more detailed information of a container, including its status, gateway, IP address, etc., use

\$ docker inspect [container_name]

To create an image from a container, use

\$ docker commit [container_name] [image_name]

The docker commit command saves the container's file changes or settings into a new image, which allows easier populating containers or debugging in a later stage. Notice that docker commit does not save everything of the container into the image.

11.3.4 Publish a Container

A container can be configured to be accessible from not only the host machine but also other computers from outside the world. In a typical web service application, the host machine and the containers are often designed following the simplified architecture in Fig. 11.8. The containers shall be configured similarly to provide consistent services. A web server, such as *apache* or *nginx*, shall be installed and kept running in the containers. The load balancer is a special software toolkit that monitor the status of the containers, manage the dataflow, and scale up and down the number of containers depending on the total service requests.

An example of setting up a web server in containers from scratch is given in this section, where a static web service is established in the containers, and one of the container's IP port 80 (for HTTP protocol) is mapped with the host machine's IP port 80, thus published to the internet. Notice that in the example, for simplicity, the load balancer and the shared services are neglected, and only one container's IP port 80 is mapped with the host machine's IP port 80 (i.e., the internet can access only one of the containers).

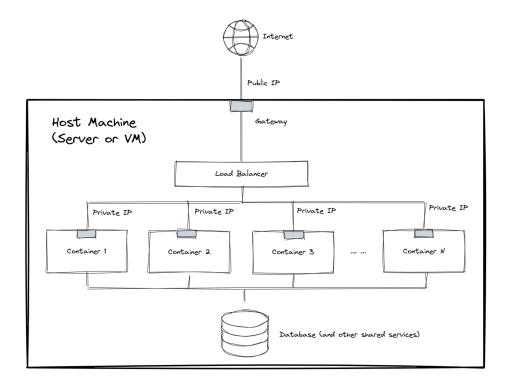


FIGURE 11.8

A simplified architecture where containers are used to host the web service.

11.4 Docker Images

In earlier Section 11.3, images have been used to create containers. An image performs like a template or source of a container, including its basic settings, initial configurations, required libraries, and other metadata. The above information is stored in *Dockerfile*. A minimal Dockerfile may look like the following

```
FROM alpine CMD ["echo", "Hello World!"]
```

More details are introduced in the rest of this section.

Part IV Linux Security

Part V Linux on Cloud

Bibliography