Lu Sun, and many more.

A Notebook on Linux Operating System

To my family, 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

Foreword

If software or e-books can be made completely open-source, why not a notebook?

This brings me back to the summer of 2009 when I started my third year as a high school student in Harbin No. 3 High School. In around the end of August when the results of Gaokao (National College Entrance Examination of China, annually held in July) are released, people from photocopy shops would start selling notebooks photocopies that they claim to be from the top scorers of the exam. Much curious as I was about what these notebooks look like, never have I expected myself to actually learn anything from them, mainly for the following three reasons.

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

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

And lastly, as a student in the top-tier 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 requesting a copy from him or her directly?

However, things had changed after my becoming an undergraduate student in 2010. There were so many modules and materials to learn in a university, and as an unfortunate result, students were often distracted from digging deeply into a module (For those who were still able to do so, you have my highest respect). The situation became even worse as I started pursuing my Ph.D. in 2014. As I had to focus on specific research areas entirely, I 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, just to force myself to spent time learning new subjects out of my comfort zone. I used to take hand-written notebooks. My very first notebook was on *Numerical Analysis*, an entrance level module for engineering background graduate students. Till today I still have on my hand dozens of these notebooks. Eventually, 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?

—

As most of the open-source software, this notebook (and it applies to the other notebooks in this series as well) 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 absolutely 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 helpful with your research, please 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 lecturer!) or for lecturers 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 experience.

—

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.

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

List of Tables

Part I

Linux Basics

1

Brief Introduction to Linux

CONTENTS

This chapter gives a brief introduction to Linux, including some of its key features and advantages, disadvantages over 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 (desktop, laptop, server, mobile devices, edge devices and other equipment capable and sophisticated enough to host an OS) that manages hardware resources and supports application software in the system. An OS shall be able to

- detect and prepare hardware;
- manage process;
- manage memory;
- provide user interface and user authentication;
- manage file system;
- 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 websites such as 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 thinking that it is running on a dedicated machine.
- Cloud computing: flexible resources management is achieved by running applications on clouds on virtual machines running Linux OS.
- 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 successful OSs. Among the three OSs, Linux is the only one that is completely open-source, in the sense that its source code can be viewed and customized by the users per requested.

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 inspired by UNIX. UNIX operating system was created by AT&T in 1969 as a software development environment that AT&T used internally. In 1973, UNIX was rewritten in C language, thus gaining 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 it. 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) of UNIX.

From then on, UNIX moved towards two separate directions: BSD continued in the "open" and "share" manner, while AT&T started steering UNIX towards 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 pre-installed. For this reason the price for the source code had to be set high as it is targeted for PC manufactures, not for end users. This largely prevented an end user from procuring UNIX source code from AT&T directly. The PC manufactures were in fact more successful than AT&T just by selling UNIX based PC and workstations to the end users. Overall, although the community acknowledged that UNIX was useful, UNIX source code was extremely costly and was not popular among the end users.

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 the 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, which only worked on 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 emerged. They share the same Linux OS kernel but differ from each other in many ways such as software management tools and user interfaces.

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 is publicly available as required by GPL license (GPL requires that any modified versions of a GPL-licensed product shall also be made open-source with a GPL license, as long as the modifications spread in the community), 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 convene the life of the users.

- Red Hat Based Distributions
  - Red Hat Enterprise Linux (RHEL)

  - Fedora
  - CentOS
- Debian Based Distributions
  - Debian
  - Ubuntu
  - Linux Mint
  - Elementary OS
  - Raspberry Pi OS

1.3.1 Red Hat Based Distributions

Some of the main features of Red Hat based distributions are as follows. Red Hat created the Red Hat Package Manager (RPM) 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 OS that can host mission-critical applications for big business and governments. To use RHEL, customers buy subscriptions which allow them to deploy any version of RHEL as desired. Different levels of support are available 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 users 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 be tested on Fedora before implementation on RHEL.

1.3.2 Debian Based Distributions

Different from Red Hat based distributions that use RPM, Debian and Debian based distributions use Advanced Packaging Tool (APT) to manage software. ATP simplifies the process of managing software by automating the

FIGURE 1.1
GNOME desktop environment.

retrieval, configuration and installation of software packages, either form pre-compiled files or by compiling source code. Among all Debian based distributions, Ubuntu is the most successful and popular one. Ubuntu has a variety of graphical tools and focuses on full-featured desktop system while still offering popular server packages. It 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 because Ubuntu servers as a stable platform while Fedora is more of a "testbed". Ubuntu is more for casual users and beginners, while Fedora more for advanced users or developers, especially developers for RHEL.

1.4 Linux Graphical Desktop

Though not necessary, many Linux distributions support graphical desktops. During the installation of these distributions, the user can choose whether to install a graphical desktop environment along with the OS. The most common choice is GNOME. There are other choices such as KDE, LXDE and Xfce desktops. GNOME and KDE are more for regular computers while LXDE and Xfce are light in size, thus more for low-power-demanding systems.

Figures 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.

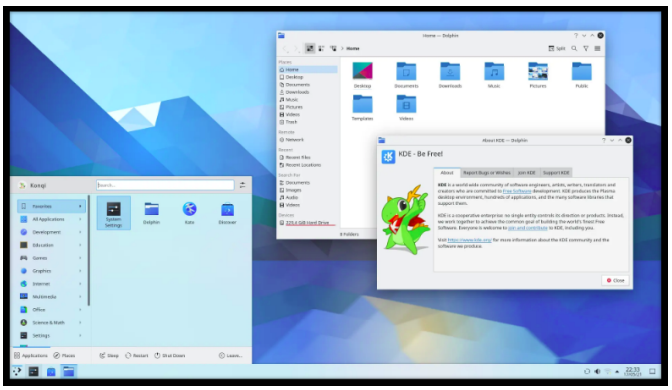

FIGURE 1.2
KDE desktop environment.

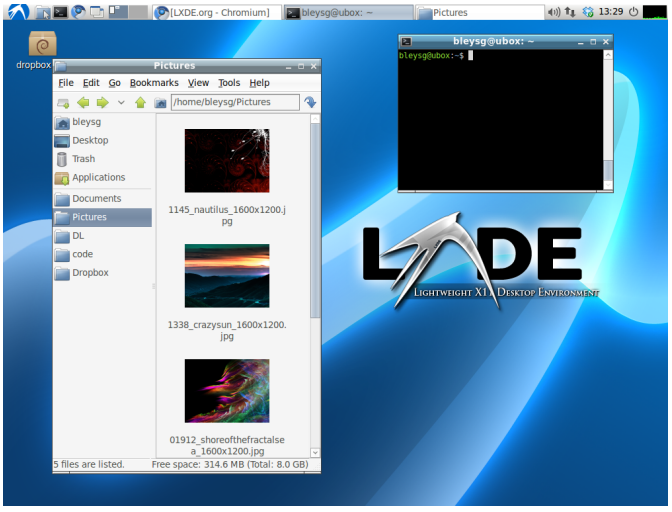

FIGURE 1.3
LXDE desktop environment.

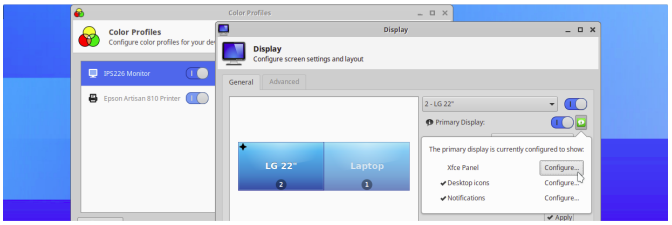

FIGURE 1.4
Xfce desktop environment.

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

1.5 Linux Installation

Linux can be installed both on a fixed hard drive or on a mobile storage such as a thumb drive. The installation of different distributions may differ. Thanks to the graphical installation tools for the popular distributions, the installations can be done fairly easily.

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*. Red Hat provides different types of RHEL licenses for different using purpose, including developer license, which is cheaper than a standard enterprise-level license and serves well for learning purpose.

2

Shell Basics

CONTENTS

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, flexible and better-supported by the community.

Linux shell will be used rapidly in the remaining parts of this notebook.

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 Fibonacci series is calculated "1, 1, 2, 3, 4, 8, 13, 21, 34, 55".

```
#!/usr/bin/bash
n=10
function fib
{
  x=1; y=1
  i=2
  echo "$x"
  echo "$y"
  while [ $i -lt $n ]
  do
      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 is sometimes more convenient than the bash shell. In case where the Linux distribution does not have these shells pre-installed, the user can install and use these shells just like installing other software.

In this notebook, bash shell is assumed unless otherwise mentioned.

2.1.2 Prompt and Basic Concepts

After opening the shell or terminal, a string (usually containing username, hostname, current working directory, etc.) followed by either a `$` or `#` appears, following which the user inputs the shell command. The string may look like the following:

```
<username>@<hostname>:~$
```

The above string is called a *prompt*, indicating the start of a user 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 2.1.3, 2.2.5 for details about environment variable and shell configuration, respectively.

For the term "root user", we are referring to a special user with username and user ID (UID) "root" and 0 respectively. This UID gives him the administration privilege over the machine, such as adding/removing users, changing

ownership of files, etc. To avoid vital damage by human error, root user shall not be used unless absolutely necessary. For this reason, the root user's authentication is often deactivated by default (by setting its login password to invalid).

Notice that the root user is different from a "sudoer", later of which is basically a regular user equipped with *sudo privilege*. A sudoer can temporarily switch to root user by using `sudo su` as follows.

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

More about sudo privilege, `sudo` and `su` commands are introduced later.

Key in a command after the prompt followed by `Enter` key to execute the command. A Linux shell command usually has the following form.

```
$ <command> [<option>] [<input>]
```

2.1.3 Set Shell Environment Variables

To execute a command by its name, the OS needs to know where the command is located at. Commonly used commands shall be included in the PATH environment of the shell, so that they can be executed anytime from any working directory. The 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>: ...
```

where `echo` displays a line of text, and `$PATH` is a built-in enviornmental 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 integrate more 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`, `mlocate` to search all the accessible files in the system. An example is given below.

```
$ type <command>
<command-location>
```

In addition to `PATH`, there is a list of other shell environment variables for the user to monitor and control and status of the system. Table 2.1 summarizes commonly used shell environment variables. Command `echo $<variable-name>` can be used to check the values of these variables. Use command `env` to check a list of environment variables in the shell.

TABLE 2.1
Commonly used shell environment variables.

Variable	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 assigned 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.

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. Notice that each time the shell is restarted, the environmental variables are also reset. To permanently change the value of an environmental variable, consider changing it in the bash start scripts such as `~/.bashrc`.

2.2 Basic Commands

Some basic and useful commands are briefly introduced in this section by categories.

2.2.1 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 UID and GID information of the current user.

2.2.2 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. Sometimes the displayed information can be too detailed, thus it is more convenient to use `-short` argument.

```
$ sudo lshw
```

2.2.3 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
<working-directory-path>
$ ls
<file-list>
```

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.

2.2.4 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 scripts such as `~/.bashrc`.. See Section 2.2.5 for details.

2.2.5 Other Commands

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.

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 (`|`),

TABLE 2.2
Shell configuration files.

File pathname	Description
`/etc/profile`	The environment information for every user, which executes 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 executes upon the user logs out or exit the last bash shell.

ampersand (`&`), semicolon (`;`), dollar (`$`), parenthesis (`()`), square bracket (`[]`), less than sign (`<`), greater than sign (`>`), double greater than sign (`>>`), error greater than sign (`2>`) and a few more more, are used for this feature. Details are given below.

The pipe (`|`) 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 | 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 2.1.3.

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
```

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
```

The error greater sign `2>`, `2>>` works similarly with `>`, `>>` except that instead of redirecting standard output messages, it redirects the error messages.

3

Text File Editing

CONTENTS

Many text editors are supported in Linux, to name a few, *Vim*, *Emacs*, *gedit*, *Visual Studio Code*. These text editors come with different features. Some of the text editors may even play the role of an integrated development environment (IDE).

Among the vast number of choices, *Vim* is probably the most popular one. It works perfectly in a shell environment without relying on graphical desktop, thus is adopted by many Linux distributions as the built-in text editor. *Vim* is introduced in this chapter, followed by a brief review of 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; as a result, **the brain does not need to halt and wait for the hand to grab and move the mouse**, thus speeding up the text editing.
- *Vim* is highly flexible and can be customized according to the user's habit (for example, through `~/.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 which boost its features to a higher level.

Vim can become very powerful and convenient for the user if he is very used to it. However, *Vim* is not as intuitive as other text editors such as *gedit*, and there might be a learning curve for the 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* maps keyboard inputs with the text file just 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 and quit.

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
Commonly used modes in *Vim*.

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 commonly 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.
Cmdline	It takes in a single line command input and perform actions accordingly, such as save and quit.

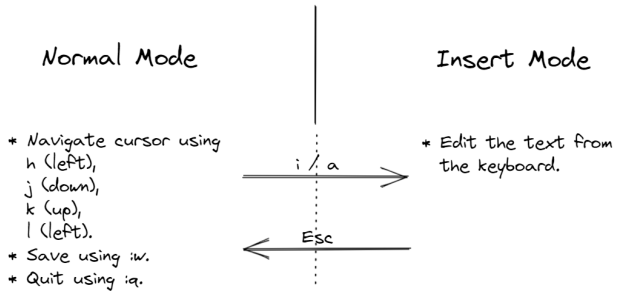

FIGURE 3.1
Mode switching between normal mode and insert mode, and basic functions associated with the modes.

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.

In the opened file, use `Esc` and `i/a` to switch between normal mode and insert mode. In the normal mode, use `h`, `j`, `k`, `l` to navigate the position of the cursor. Finally, in the normal mode, use `:w` to save the file, and `:q` to quit *vim*, or use `:wq` to save and quit *Vim*.

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.

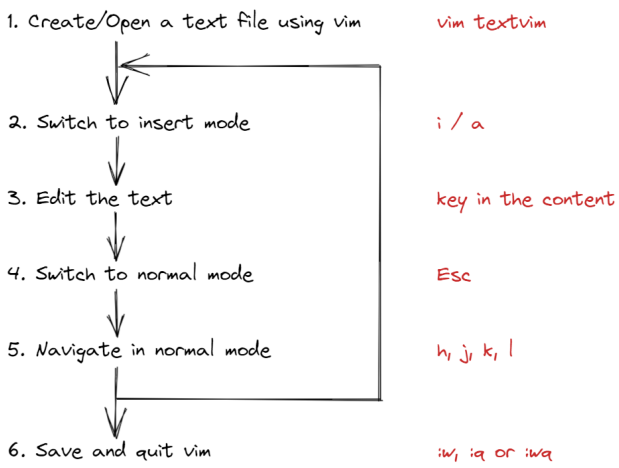

FIGURE 3.2
A flowchart for simple creating, editing and saving of a text file using *Vim*.

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

Operator	Description
`i`	Insert before the character at the cursor.
`I`	Insert at the beginning of the row at the cursor.
`a`	Insert after the character at the cursor.
`A`	Insert at the end of the row at the cursor.
`o`	Create a new row below the cursor and switch to insert mode.
`O`	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 a better 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 creating 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 the text editing. For example, by mapping `jj` to `Esc` in insert mode, one can switch from insert mode to normal mode more quickly (notice that consequent "jj" is rarely used in English). Another example coud be mapping `j`, `k`, `i` to `h`, `j`, `k` respectively in normal and visual modes, making the navigation more intuitive. In that case, an alternative key needs to be mapped to `i`.

The mapping of keys and keys combinations can be done as follows in *vimrc*.

```
inoremap jj <Esc>
noremap j h
noremap k j
noremap i k
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 normal mode are originally used to delete and substitute texts, and they are rarely used due to the more powerful shortcut `c` which does similar tasks. We can re-map `S` to saving, and disable `s`. Similarly, upper case letter `Q` is mapped to quitting *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 colored contents on a black background. Use the following command in *vimrc* to enable syntax highlighting or change color schemes. Use `:colorscheme` in cmdline mode in *Vim* to check for available color schemes.

```
syntax on
colorscheme default
```

The following setups in *vimrc* 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
```

3.3.3 Other Useful Setups

Use `scrolloff` to make sure that when scrolling in *Vim*, there are always margins lines in the top and bottom of the screen, so that the cursor is always close to the centre of the screen.

```
set scrolloff=3
```

Enable spell check in *Vim* as follows.

```
map sc :set spell!<CR>
```

where `sc` can be used to quickly turn on and off the spell check function. In addition, when the cursor is put on the wrongly spelled word, use `z=` to open a list of possible corrections.

3.3.4 Plug Tools

In the Linux community, many plug tools have been 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 instructions given by GitHub under *junegunn/vim-plug*, to use *vim-plug* on Linux, the very first step is to use *cURL*, a command-line tool for transferring data specified with URL syntax, to download *vim-plug*. To confirm that *cURL* is already installed (this is often the case), use the following command in Linux shell

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

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

```
curl:
  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
```

If cURL is not pre-installed, use `sudo apt install curl` to install curel.

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 very beginning of *vimrc*, add the following to specify the plugins to be installed. As an example, *vim-airline/vim-airline* and *joshdick/onedark.vim* are to be 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. Use `colorscheme onedark` instead of `colorscheme default` in *vimrc* to enjoy the onedark color scheme.

Notice that instead of setting up configurations permanently in *vimrc*, the user can also apply a setup in cmdline mode for temporary use in an open session. A full list of *vimrc* configurations used in this chapter can be found in the appendix.

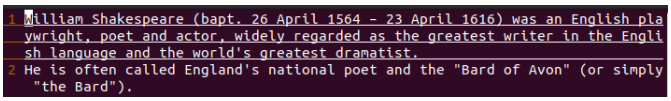

FIGURE 3.3
A piece of text of "William Shakespeare", for demonstration.

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` deletes (cuts) the entire row at the cursor and `p` pastes the content in the clipboard to the cursor position. For beginners, remembering shortcut keys can be difficult. Therefore, it is suggested looking for the consistent patterns of the different commands, rather than 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 without a space in between.

```
<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 separate row as shown in 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").

Use either `x` or `X` to delete the character at the cursor or previous to the cursor, respectively. To delete multiple characters, one way is to press `x` or `X` multiple times (or hold the keys). 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, `10l` executes `l` for 10 times, moving the cursor to the right for 10 characters.

Operator `d`, similar with `dd`, deletes the contents of the text, but it requires a motion command and can be used more flexibly. The motion shall tell *Vim* the applicable range to delete/cut.

For example, `dl` 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, `d20l` deletes 20 characters to the right, where "20l" as a whole plays as the motion of "20 characters to the right". A combination by using things like `5d4l` also works and leads to the same result as $20 = 5 \times 4$.

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 `l`, except that `l` 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 `l` 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, then apply `dw`. 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`, `l` 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".

The operators and motions introduced so far are summarized 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 after removing the content. To copy a piece of text to clipboard, use `y` (stands for "yank") followed by 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.

TABLE 3.3
Commonly used operators related to delete/cut, change, copy and paste.

Operator	Description
`x`	Delete/Cut the character at cursor.
`X`	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.
`yy`	Copy the entire row.
`y`	Copy selected text according to the motion command.
`p`	Paste clipboard to the cursor.

TABLE 3.4
Commonly used motions.

Motion	Description
`h`, `l`	One character to the left or right.
`j`, `k`	One row to the up or down.
`b`	First character of the current word, or fist character of the previous word.
`e`	Last character of the current word.
`w`	First character of the next word.
`(`, `)`	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[`, `i{`	inner-block for different types of brackets.
`0`	Beginning of the row.
`$`	Ending of the row.
`gg`	Beginning of the text.
`G`	Beginning of the last row of the text.

In addition to 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

In normal mode, use `/<content>` to search a keyword or a phrase. The following setup in *vimrc* shall give a better searching experience.

When searching in the text for a particular word or phrase (searching in the text will be will be covered in a later of the chapter), to make the searching result highlighted, add the following line to the user profile *vimrc*.

```
set hlsearch
exec "nohlsearch"
set incsearch
noremap <Space> :nohlsearch<CR>
set ignorecase
noremap = nzz
noremap - Nzz
```

where `hlsearch` enables highlighting all matching results in the text, and `incsearch` enables highlighting texts along with typing the keyword. *Vim* remembers the keyword from the previous search and may automatically highlight them in the text on a new session, which can be confusing sometimes. The command `exec "nohlsearch"` (`exec` command in the user profile makes *Vim* execute that command when starting a new session) that comes after `set hlsearch` forces *Vim* to clear its searching memory on a new session. Finally, to quit searching, use `:nohlsearch` in cmdline mode and the highlights shall be gone. For convenience, consider mapping it with a customized shortcut key as well, for example to `Space`. As a bonus, set `ignorecase` to ignore case-sensitive during the searching.

Keys `n` and `N` is used to navigate through the searching result, and `zz` is used to pin cursor position in the central of the screen. They are mapped to `-` and `=` in *vimrc*. Notice that they can also be used as motion together with delete/cut, change and copy as given in Table 3.3. Many users in the community have posted their recommended *Vim* user profile configurations, which could be good references when setting up the configurations of your own.

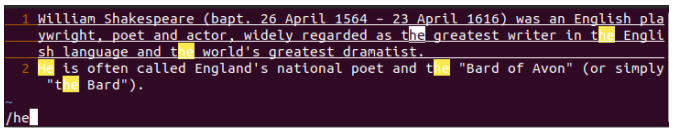

FIGURE 3.4
Search "he" in the piece of text of "William Shakespeare".

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 and enabling free move of the cursor.

3.4.3 Other Tips

Use `Ctrl+o`, `Ctrl+i` to "undo" and "redo" cursor positions, respectively. They only move the cursor position and don't change the actual texts.

To save as a file, use `:w <new path>` in cmdline mode.

In cmdline mode, use `!` to interact with the bash shell. For example, consider a case where a read-only file needs to be edited and saved by a sudoer who forgot to start *Vim* using `sudo`. The common `:w` will be rejected. In this case, use `:w !sudo tee %` to perform the save, where `tee` is a Linux command that takes standard input and writes to a file, and `%` stands for the current file. In another example where an existing file's content is to be insert into the current text, navigate the cursor to the place to insert the text, and use `:r !cat <filename>`.

To convert and save the current text into an *html* file, use `:%TOhtml`. From there, the file can be further converted into a PDF file.

3.5 Visual Modes

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

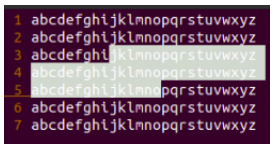

FIGURE 3.5
An example of visual mode where a block of text is selected.

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 for each line. This allows convenient editing of multiple lines of text all together. For example, use `V` to select a few lines of contents, followed by `:normal 0iprefix-` (or `:normal 0hprefix-`, if the insert key `i` has been mapped by `h`), and "`prefix-`" will be added to the beginning of each of the selected lines.

In visual-block mode, after selecting a block of content, use `I` to enter insert mode and insert content in the first row of the selected block. When existing the insert mode using `<Esc>`, the changes will apply to all selected rows.

3.6 Vim Macros

Vim macros can become handy for frequent and repetitive works. Use `q` in normal mode to start and end a macro recording. The syntax follows `q<macro-name><operations><q>` where `<macro-name>` is a single character that labels the macro.

Consider the following example where there is a text file as follows

```
apple.jpg
pear.jpg
orange.jpg
banana.jpg
peach.jpg
```

and we would like to edit it to get the following revised text

```
image: 'apple.jpg'
ttl: 5
image: 'pear.jpg'
ttl: 5
image: 'orange.jpg'
ttl: 5
```

```
image: 'banana.jpg'
ttl: 5
image: 'peach.jpg'
ttl: 5
```

In this example, repetitive work is involved and it is time consuming to do it manually if there are thousands of items in the file, and it is better to record a macro to automate the procedure. Navigate the cursor to the first row of the text, and type the following sequence of characters.

```
q<macro name>0iimage: '<Esc>A'<Enter>ttl: 5<Esc>jq
```

where `<macro name>` can be any character, for example `s`. The string in the middle `0iimage: '<Esc>A'<Enter>ttl: 5<Esc>j` is the necessary procedure to perform the revision for one row. If everything is done correctly, the following text should be obtained

```
image: 'apple.jpg'
ttl: 5
pear.jpg
orange.jpg
banana.jpg
peach.jpg
```

and the cursor should be somewhere at row `pear.jpg`.

To repeat the recorded procedures, use `@<macro-name>`. In this example, just key in `@s` in the normal mode, and the text shall become

```
image: 'apple.jpg'
ttl: 5
image: 'pear.jpg'
ttl: 5
orange.jpg
banana.jpg
peach.jpg
```

Repetitively using `@s` proceeds with the revision. In the case the procedure needs to be repeated for many times, use `<number>@<macro-name>`, in this example `3@s`, to complete the remaining task.

3.7 File Explorer and Screen Splitting

Many IDEs come with project folder navigation and screen splitting features. In these IDEs, there is often a built-in file explorer, from where the user can navigate in the file system, select a file to edit, and the IDE will split the window for the selected file. *Vim* has similar features that supports file explorer and screen splitting, either via built-in functions or third-party support tools.

In *Vim*, use `:Explore`, `:Sexplore` or `:Vexplore` (or `:Ex`, `:Sex`, `:Vex` for short) in cmdline mode to open a file explorer, from where the user can navigate the cursor to select a file. *Vim* will then open the file in a split window that allows the user to further editing the file.

There are many third-party plugin tools that enable convenient file explorer functions. For example, *github.com/scrooloose/nerdtree*. More details can be found in the associated repository on *GitHub*.

Use `:split` and `:vsplit` for horizontal and vertical screen splitting, respectively. A second split window would show up with the same text file opened. For simplicity, these commands can be mapped in *vimrc* as follows.

```
noremap sj :set nosplitright<CR>:vsplit<CR>
noremap sl :set splitright<CR>:vsplit<CR>
noremap si :set nosplitbelow<CR>:split<CR>
noremap sk :set splitbelow<CR>:split<CR>
```

where `splitright` and `splitbelow` is used to setup the default cursor position after splitting the screen.

In a split window, open a new file using `:e <path>`. To navigate the cursor across different split windows, use `Ctrl+w` followed by `h`, `j`, `k` and `l`. For simplicity, they can be mapped as follows.

```
noremap <C-j> <C-w>h
noremap <C-l> <C-w>l
noremap <C-i> <C-w>k
noremap <C-k> <C-w>j
```

where `<C->` stands for `Ctrl+`.

Resize the selected split window using `:res+<number>`, `:res-<numer>`, `:vertical resize+<number>`, `:vertical resize-<number>`. For simplicity, map these commands as follows.

```
noremap J :vertical resize-2<CR>
noremap L :vertical resize+2<CR>
noremap I :res+2<CR>
noremap K :res-2<CR>
```

3.8 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.

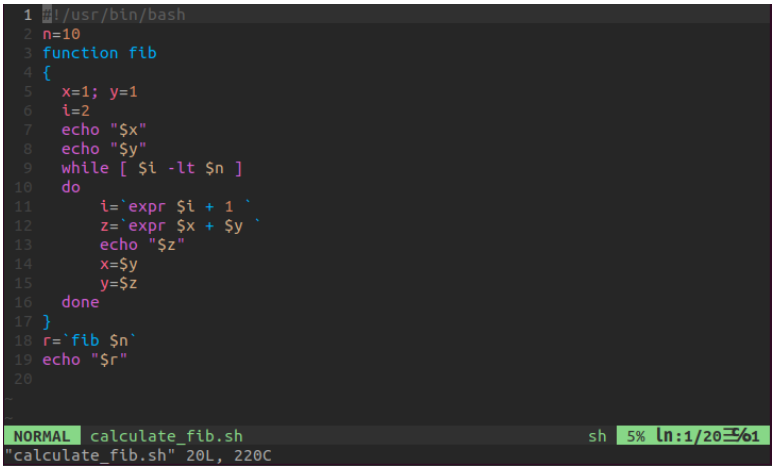

FIGURE 3.6
Vim (with user's profile customization as introduced in this chapter).

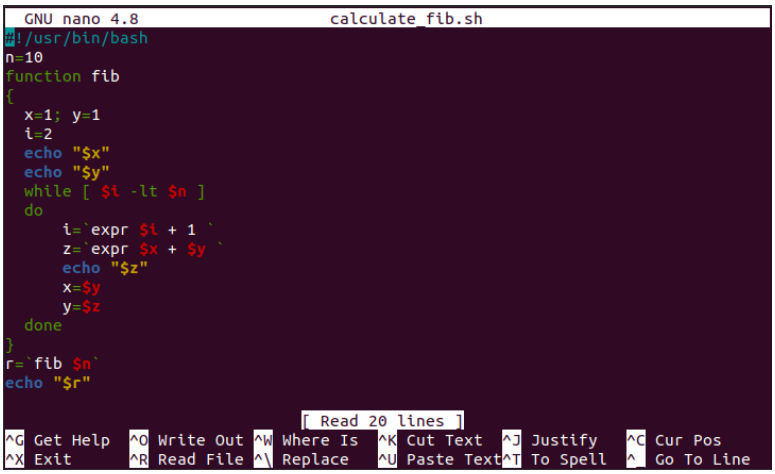

FIGURE 3.7
Nano.

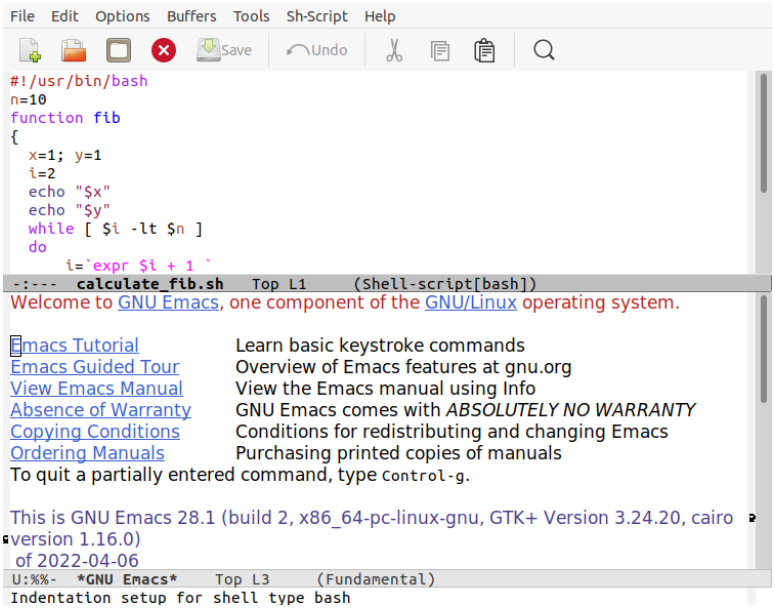

FIGURE 3.8
Emacs.

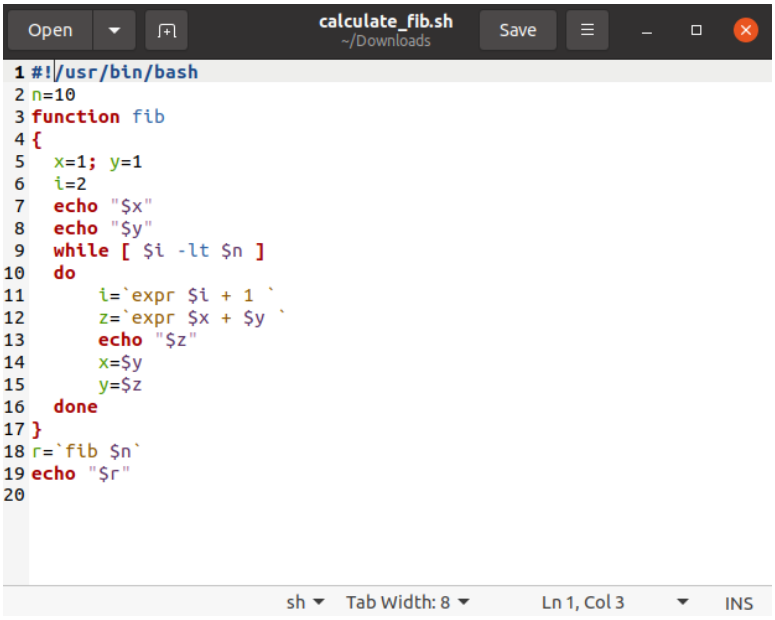

FIGURE 3.9
Gedit.

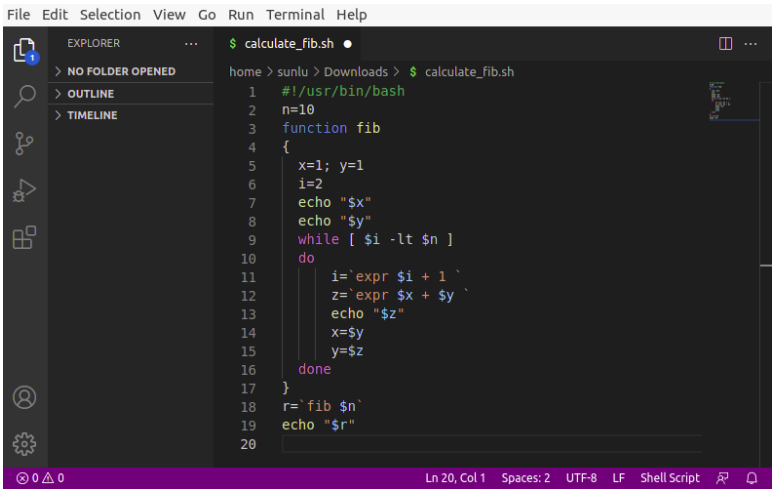

FIGURE 3.10
Visual Studio Code.

4

File Management

CONTENTS

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

```
/<directory>/<subdirectory>/.../<directory-name>
/<directory>/<subdirectory>/.../<file-name>
```

FIGURE 4.1
An example of Linux file system hierarchy.

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

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 different Linux distributions may differ slightly on how the architecture looks like. The "`/`" 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 pre-

FIGURE 4.2
A rough categorization of commonly used directories in Linux file hierarchy standard.

determined 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 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 regular users.
`/usr/lib`	Libraries to support `/usr/bin` and `/usr/sbin`.
`/opt`	Application software installed by OS and administrator 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 applications.
`/usr/src`	Source files or packages managed by software manager.
`/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>`	A regular user's home directory, containing executables, configurations and files specifically belong to this user.

TABLE 4.2
Commonly used commands to navigate in the Linux file 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, `ls` 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>]
```

```
sunlu@SUNLu-Laptop:~$  ls
anaconda3  Documents  Dropbox  eclipse-workspace  gurobi.log  octave-workspace  Public  snap       texmf
Desktop    Downloads  eclipse  gurobi             Music       Pictures          R       Templates  Videos
sunlu@SUNLu-Laptop:~$  ls -l
total 72
drwxrwxr-x 27 sunlu sunlu 4096 Dec 21 01:54 anaconda3
drwxr-xr-x  2 sunlu sunlu 4096 Apr 30  2021 Desktop
drwxr-xr-x  3 sunlu sunlu 4096 Jun 15 10:42 Documents
drwxr-xr-x  2 sunlu sunlu 4096 Jun 15 14:37 Downloads
drwx------ 13 sunlu sunlu 4096 Jun 15 10:38 Dropbox
drwxrwxr-x  3 sunlu sunlu 4096 Dec 21 12:34 eclipse
drwxrwxr-x  3 sunlu sunlu 4096 Feb  3  2021 eclipse-workspace
drwxrwxr-x  3 sunlu sunlu 4096 Mar  1  2021 gurobi
-rw-rw-r--  1 sunlu sunlu  488 Mar  3  2021 gurobi.log
drwxr-xr-x  2 sunlu sunlu 4096 Jan 26  2021 Music
-rw-rw-r--  1 sunlu sunlu   43 May  5 16:48 octave-workspace
drwxr-xr-x  2 sunlu sunlu 4096 Mar 29 08:58 Pictures
drwxr-xr-x  2 sunlu sunlu 4096 Jan 26  2021 Public
drwxrwxr-x  3 sunlu sunlu 4096 Feb  7  2021 R
drwx------  9 sunlu sunlu 4096 Apr  8 14:03 snap
drwxr-xr-x  2 sunlu sunlu 4096 Feb  3  2021 Templates
drwxrwxr-x  4 sunlu sunlu 4096 Feb 23  2021 texmf
drwxr-xr-x  2 sunlu sunlu 4096 Jan 26  2021 Videos
sunlu@SUNLu-Laptop:~$  ls Do*
Documents:
MATLAB

Downloads:
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 `ls` 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 `-l` 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 `ls` command manual, which is accessible via `ls --help`. Some commonly used `ls` 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, `ls -al` aggregates the effects of using `ls -a` and `ls -l`.

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>
```

For example,

TABLE 4.3
Commonly used arguments and their effects for *ls* 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.
`-l`	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.

```
$ 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>
```

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.

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.

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.

```
$ mv [<option>] <source> <target>
```

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> <target>
```

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.

TABLE 4.6
Commonly used wildcard characters.

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

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

```
$ shred --help
```

4.2.6 Use of Wildcard Characters

When performing moving, copying, removing or otherwise acting on files, wildcard characters 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 4.6.

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 `ls -l`. 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 `d` and `-` indicate subdirectory and regular file respectively. Other commonly seen indicators are `l` for a symbolic link, `b` for a block device, `c` for a character device, `s` for a socket and `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 `ls` 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.

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.

```
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.

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 `ls` 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.

```
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.

4.3.2 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>] <new_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-digit numbers such as `664` as shown in Fig. 4.5 can also represent a permission. The first digit is associated with the permission given to the "user", where in this example `6` represents `rw-`. Notice that each permission is assigned a number: `r` is 4, `w` is 2, and `x` is 1. From the sum of the numbers, the OS understands which permission(s) are assigned. For example, $6 = 4 + 2$, hense `r` and `w` permissions. Likewise as another two quick examples, a `7` would mean `rwx` and `5`, `r-x`, respectively. The second and third digits are associated with the permission given to "group" and "other" respectively. Hense, `664` would assign `rw-rw-r--` to the 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 Looking for a Command

Use `type` to look for a command as follows.

```
$ type <command>
```

For example

```
$ type cd
cd is a shell builtin
$ type python
python is /usr/bin/python
$ type ls
ls is aliased to 'ls --color=auto'
```

4.4.2 Looking 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 <file-name>
```

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` 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

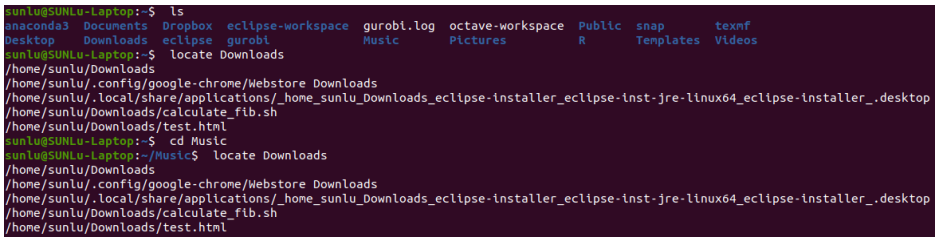

FIGURE 4.6
Search for files and directories using *locate*.

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` as follows.

```
$ find [<options>] [<path>] <expression>
```

where `options` can be used to specify whether to follow a symbolic link in the result, and expression the filtering method, such as `-f` for file, `-size` for searching by size. It is possible to search by name, size, extension, type, size, and other metadata.

4.5 File Archive

The `tar`, `gzip` and `zip` commands can all be used in files archive, but with different features, as given in Table 4.8.

Only `tar` command is introduced here, as it is the most commonly used and can satisfy most use cases. A common way of using `tar` is as follows. Use

```
$ tar -cvzf <archive-file> <file1> <file2> <file3> ...
```

to archive and zip files, and

TABLE 4.8
Commonly used file archive tools.

Directory	Description
`tar`	Save many files together into a single tape or disk archive, and can restore individual files from the archive. By default, it does not compress the files. However, `-z` option can be used in combination of the command to add compression feature.
`gzip`	Compress or restore files.
`zip`	Compress multiple files one-by-one and integrate them together into a single file.

```
$ tar -xvzf <archive-file>
```

to restore files from the archive file. The commonly used archive file name, in this scenario, is `<filename>.tgz`.

The detailed explanation to all available options for `tar` can be found using `tar --help`. The most commonly used options are `-c`, `-x`, `-z`, `-f` and `-v`, standing for creating compress tape, extracting (restoring) file, adding compressing feature, using file archive, and listing processed files in the console, respectively.

4.6 *Ranger* for File Management

Ranger is a *Vim* based terminal file explorer tool that is highly flexible and customizable. Just like *Vim*, Ranger does not rely on a mouse or a graphical interface, and can manipulate file system in an intuitive way.

To install Ranger, use

```
$ sudo apt install ranger
```

and to run ranger in the current working directory, simply type `ranger`.

An example to demonstrate ranger interface is given in Fig. XXX.

5

Software Management Basics

CONTENTS

...

5.1 Linux Kernel Management

...

5.2 General Introduction to Linux Package Management Tools

...

5.3 Software Management

...

5.3.1 Searching for Software

...

5.3.2 Installation of Software

...

5.3.3 Upgrading Software

...

5.3.4 Uninstallation of Software

...

6

Process Management

CONTENTS

A process refers to an instance of a computer program that is running in the system. Managing processes is one of the essential tasks of an OS. In a Windows system, the user can use the task manager, which is a graphical tool to check and manage all the running processes. In a Linux system, the user can manage process in the prompt console using bash commands.

6.1 General Introduction to Process

A process is the fundamental unit for the OS to manage the resources used by a running program.

6.1.1 Process

To improve the efficiency of the CPU, the OS allows multiple processes (also called tasks, jobs) to share the computational capability and memory of the system, each thinking that it is exclusively using the all machine resources, as shown in Fig. 6.1.

The status of a process is stored in its *Process Control Block (PCB)*. The PCB is a special data structure used to describe the dynamic of a process. The OS manages the PCBs and control the processes accordingly. Some of the attributes of a PCB are summarized in Table 6.1.

A process shall at least have the following states. The fundamental states of a process and their transferring are introduced in Fig 6.2, where "blocked" indicates that the process is waiting for other inputs to carry out the remaining part of the program, and "suspended" indicates that the process is hold for

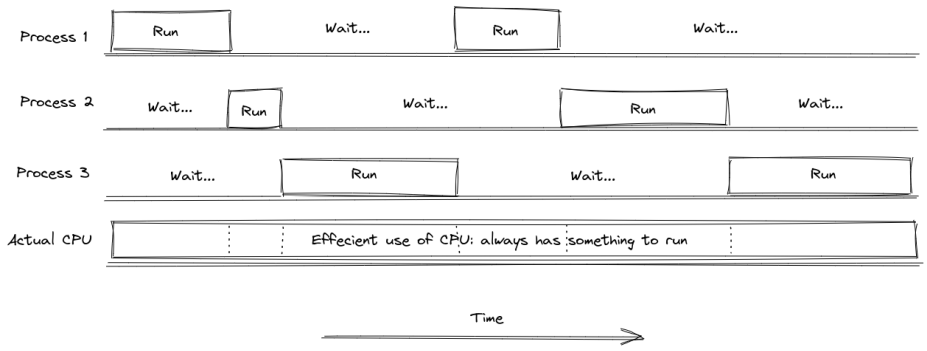

FIGURE 6.1
A demonstration of running multiple processes on a single-core CPU.

TABLE 6.1
Some attributes of a PCB.

Name	Description
Identifier	The unique ID of the process.
State	The state of the process, for example, running, suspended, terminated.
Priority	Priority level in comparison with other processes.
Program Counter	A pointer to the next line of program to be executed.
Memory regions	A pointer to the RAM where the code and data of the process is stored.
Accounting Information	Time limits, clock time used, etc.

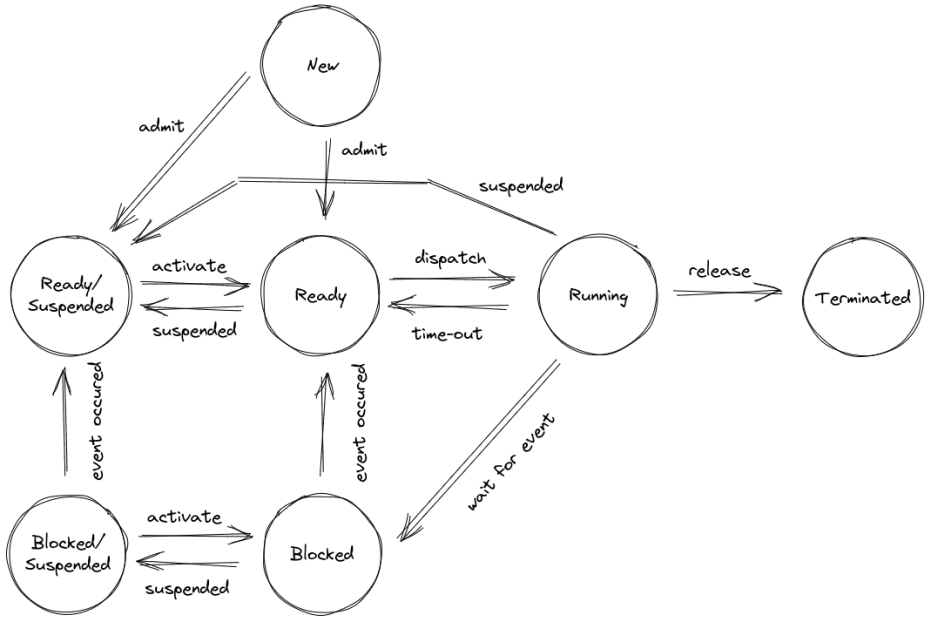

FIGURE 6.2
Fundamental states of a process and their transferring.

some reasons. When a process is offloaded from the CPU, its context is moved from the CPU registers to the PCB of the progress.

There are different types of processes. For example, based on the source of the processes, there are OS triggered processes and user triggered processes, the first of which usually has a higher priority. Based on the running environment, there are front-ground processes and background processes. Based on the resources used, there are CPU processes and I/O processes.

A process usually has a relatively isolated environment, and it does not share memory storage with other processes. Special inter process communication mechanism, which is often referred as "pipe", is required for processes to talk to each other. Inter process communication requires OS level controls.

6.1.2 Thread

A "thread" is like a work dispatch inside a process. There can be multiple threads in a process, as shown in Fig 6.3. Each thread has its own CPU register values and stack, but they share the same program, memory and file storage addresses.

Threads differ from processes in the following aspects:

- A thread is lighter than a process, occupying less resources to create.

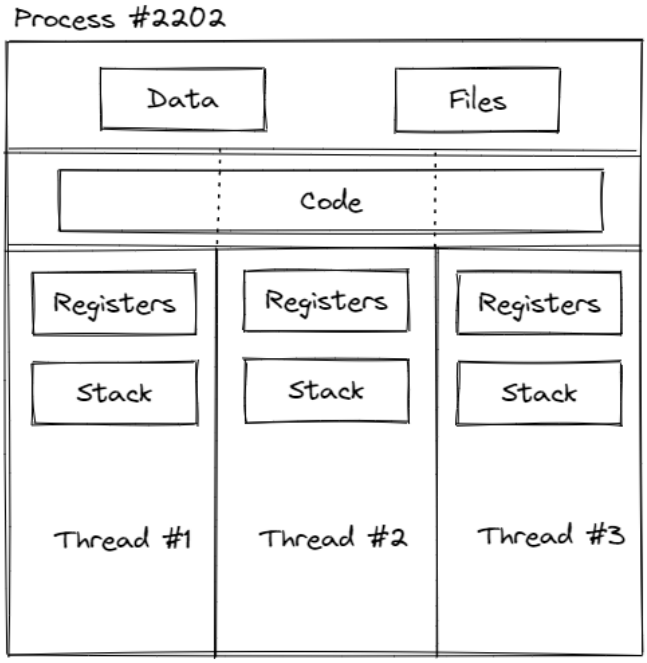

FIGURE 6.3
A demonstration of multiple threads in a process.

```
sunlu@sunlu-laptop-ubuntu:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 15:34 ?        00:00:02 /sbin/init splash
root           2       0  0 15:34 ?        00:00:00 [kthreadd]
root           3       2  0 15:34 ?        00:00:00 [rcu_gp]
root           4       2  0 15:34 ?        00:00:00 [rcu_par_gp]
root           5       2  0 15:34 ?        00:00:00 [netns]
root           7       2  0 15:34 ?        00:00:00 [kworker/0:0H-events_highpri]
root          10       2  0 15:34 ?        00:00:00 [mm_percpu_wq]
root          11       2  0 15:34 ?        00:00:00 [rcu_tasks_rude_]
root          12       2  0 15:34 ?        00:00:00 [rcu_tasks_trace]
root          13       2  0 15:34 ?        00:00:00 [ksoftirqd/0]
root          14       2  0 15:34 ?        00:00:00 [rcu_sched]
root          15       2  0 15:34 ?        00:00:00 [migration/0]
root          16       2  0 15:34 ?        00:00:00 [idle_inject/0]
root          17       2  0 15:34 ?        00:00:00 [cpuhp/0]
root          18       2  0 15:34 ?        00:00:00 [cpuhp/1]
root          19       2  0 15:34 ?        00:00:00 [idle_inject/1]
root          20       2  0 15:34 ?        00:00:00 [migration/1]
root          21       2  0 15:34 ?        00:00:00 [ksoftirqd/1]
root          23       2  0 15:34 ?        00:00:00 [kworker/1:0H-events_highpri]
root          24       2  0 15:34 ?        00:00:00 [cpuhp/2]
root          25       2  0 15:34 ?        00:00:00 [idle_inject/2]
```

FIGURE 6.4
Execution of `ps` command.

- Sharing memories and resources among threads in a process is easier than sharing among processes, because they naturally share address space.
- It is easier to enable parallel computation for the threads in the process when it is running on a multi-core CPU.

Notice that for many OS, including Linux, the kernel can provide thread level services.

6.2 Process Management in Linux

Two commands, `ps` and `top`, are widely used in monitoring the running process in the OS. They can be used stand-alone, without additional arguments like follows.

```
$ ps
```

or

```
$ top
```

The major difference between these two commands is that `ps` provides a screenshot (in a text format) of a list of processes including their names, process IDs (PID) and owners, etc., running at the instance, while `top` provides a frequently-refreshing display of the running processes as well as their associated resources usage. Figs 6.4 and 6.5 give a quick demo of how the execution of the two commands look like.

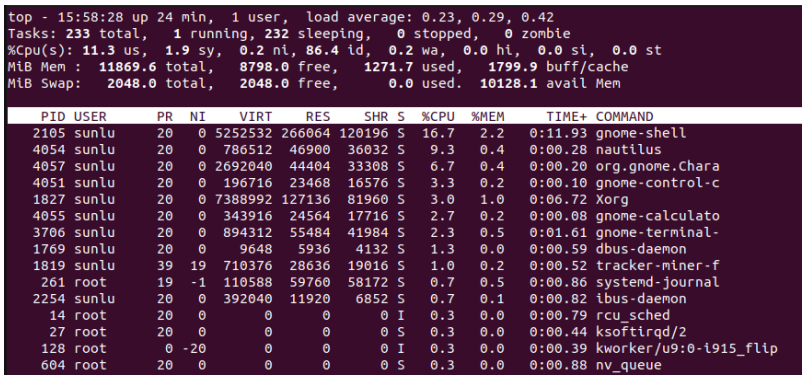

FIGURE 6.5
Execution of `top` command.

Notice that `top` is a running application where the user can keep interacting with to filter for particular processes, while `ps` is more of a one-time command and its output can be saved into a text file for further processing.

To kill a process, use the `kill` command as follows.

```
$ kill <option> <process ID>
```

The `kill` command offers different options to kill a process. Use `kill -l` to list down the options as given below (and many more).

```
1) SIGHUP   2) SIGINT   3) SIGQUIT  4) SIGILL
5) SIGTRAP  6) SIGABRT  7) SIGBUS   8) SIGFPE
9) SIGKILL 10) SIGUSR1 11) SIGSEGV 12) SIGUSR2
13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGSTKFLT
17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
...
```

Commonly used `kill` options are `kill -9` and `kill -15` followed by the PID. Notice that in Linux the process is arranged in a tree structure; killing a parent process will automatically terminate its children processes, and killing a child process may result in its parent process to restart a new child process.

7

Programming on Linux

CONTENTS

“nobreak

7.1 IDE

“nobreak

7.2 Python Programming

“nobreak

7.3 MATLAB/Octave Programming

Part II

Linux Advanced

8

Administration

CONTENTS

...

8.1 Introduction to Linux Administration

...

8.2 Root Account Management

Managing and protecting the *root* user account is a key portion in Linux administration. It is worth mentioning that the root user is different from a "*sudoer*", i.e., a user that can use `sudo` command, although they both have elevated privileges when comes to system administration. A more detailed explanation is given below.

The root user refers to the user that is created by the system upon first installation of the OS. Its username is by default "root", with a UID of 0, and a GID of 0. Its home directory is by default `/root` instead of `/home/<user name>`. The default prompt for the root user is often `#` instead of `$` for other users. The root user has administrative privileges and can do almost anything without being denied or questioned by the system. As a commonly used safety practice, the password for the root user is usually disabled, thus nobody can login to the system as the root user using username and password authentication.

Notice that in theory the root user does not necessarily need to use the username "root", although it is a default convention. The administrative

comes with the UID of 0, not the username. Thus, it is possible to assign the administrative account a different username. It is also possible to create multiple accounts with administrative privileges by assigning UID of 0 to multiple accounts, although it is not recommended to share the same UID among different users.

Although the administrative privilege of root user does not come with its username, in some systems, the username "root" is given elevated privilege anyway. More details about this is introduced later.

Check the root user information as follows.

```
$ cat /etc/passwd | grep ^root
root:x:0:0:root:/root:/bin/bash
```

where the third and fourth column of above give the UID and GID of the root user, respectively.

For comparison, a regular user would have far different UID and GID as shown below.

```
$ cat /etc/passwd | grep ^sunlu
sunlu:x:1000:1000:Sun Lu,,,:/home/sunlu:/bin/bash
```

A sudoer refers to the regular users who can temporarily elevate its privileges and execute administration commands using `sudo <privileged-command>`. A sudoer can also switch to root user prompt using `sudo su` (and quit root user prompt using `exit`). Their sudo privilege comes from the fact that they are included in the "sudo group".

Use `groups` to check existing defined groups in the system, and `groups <user name>` the groups a user is engaged. An example is given below.

```
$ groups
sunlu adm cdrom sudo dip plugdev lpadmin lxd sambashare docker
$ groups sunlu
sunlu : sunlu adm cdrom sudo dip plugdev lpadmin lxd sambashare docker
```

The elevated privilege of the sudoer group is defined in `/etc/sudoers`. A section of the file is given below, where `%` appeared in front of `admin` and `sudo` is used to indicate group name.

```
# User privilege specification
root   ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
```

It is possible to give the root account a password, thus enabling root authentication. This approach is sometimes used for troubleshooting and recovering purposes, but it is not a good practice in routine operations.

8.3 User Management

xxx

9

Storage Management

CONTENTS

Upon installation of the system, the OS shall scan the machine and automatically mount (the majority part of) the hard disks under the root directory `/`. All software, data files, devices, etc., shall be registered somewhere in the filesystem hierarchy in Fig. 4.1 introduced in Section 4.1.

Although it is often not necessary for the user to manage the storage and the filesystem by himself, Linux provides flexible tools to monitor and manage the storage of the machine, including manipulating partition table, format partition, and managing its mounting point.

9.1 Monitor Storage Status

A secondary storage, such as a hard disk, needs to be "registered" in the OS before it can be displayed correctly to the user. The registration procedure includes creating disk partitions, and specify mount point and filesystem type for each partition, which looks like Fig.

Use `df` to monitor the status of the mounted storages, including the filesystem name, size, and used percentage. Command `df -h` gives a nice snapshot of the storage usage in a human-readable format. An example is given below.

```
$ df -h
Filesystem     Size  Used Avail Use% Mounted on
tmpfs          1.2G  2.1M  1.2G   1% /run
/dev/sda3      916G   51G  818G   6% /
tmpfs          5.8G     0  5.8G   0% /dev/shm
tmpfs          5.0M  4.0K  5.0M   1% /run/lock
/dev/sda2      512M  5.3M  507M   2% /boot/efi
```

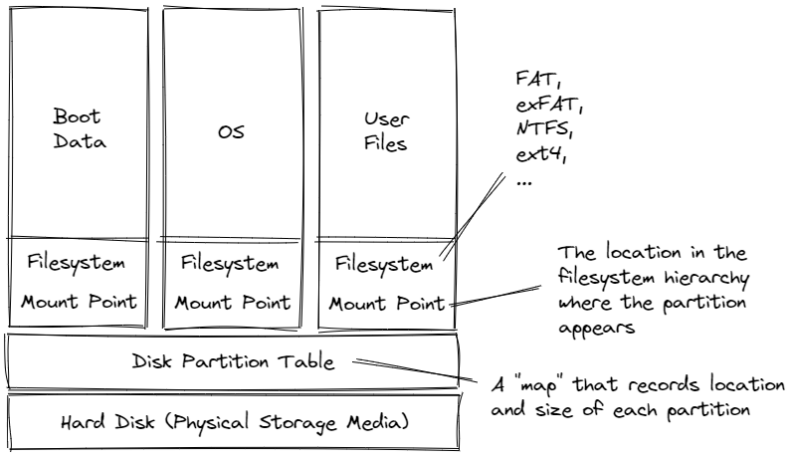

FIGURE 9.1
A demonstration of how a hard disk is "registered" in the OS.

```
tmpfs          1.2G  120K  1.2G   1% /run/user/1000
```

from where it can be seen that the most of the storage of the system, in this case $818G$, is mounted on the root directory.

Use `lsblk` to list information about block devices in the OS. Block devices refer to nonvolatile mass storage devices whose information can be accessed in any order, such as hard disks and CD-ROMs. An example is given below.

```
$ lsblk
NAME  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
loop0   7:0    0    4K  1 loop /snap/bare/5
loop1   7:1    0   62M  1 loop /snap/core20/1593
loop2   7:2    0   62M  1 loop /snap/core20/1611
loop3   7:3    0 163.3M 1 loop /snap/firefox/1670
loop4   7:4    0  177M  1 loop /snap/firefox/1749
loop5   7:5    0 400.8M 1 loop /snap/gnome-3-38-2004/112
loop6   7:6    0 248.8M 1 loop /snap/gnome-3-38-2004/99
loop7   7:7    0  81.3M 1 loop /snap/gtk-common-themes/1534
loop8   7:8    0  91.7M 1 loop /snap/gtk-common-themes/1535
loop9   7:9    0  45.9M 1 loop /snap/snap-store/575
loop10  7:10   0   47M  1 loop /snap/snapd/16010
loop11  7:11   0  45.9M 1 loop /snap/snap-store/582
loop12  7:12   0   47M  1 loop /snap/snapd/16292
loop13  7:13   0  284K  1 loop /snap/snapd-desktop-integration/10
loop14  7:14   0  284K  1 loop /snap/snapd-desktop-integration/14
sda     8:0    0 931.5G 0 disk
|-sda1  8:1    0    1M  0 part
|-sda2  8:2    0  513M  0 part /boot/efi
|-sda3  8:3    0  931G  0 part /
```

```
sr0     11:0    1  1024M 0 rom
```

which gives the size, the type and the mount point of all the storages.

Use `blkid` to print block device attributes. An example is given below.

```
$ blkid
/dev/sda3: UUID="d0b15b7c-71f2-41f4-b67a-e7c69446feab" BLOCK_SIZE
    ="4096" TYPE="ext4" PARTUUID="4b570507-6aa0-46c7-ac1b-06cb4c8bdb61
    "
```

9.2 Disk Partition Table Manipulation

Due to the development of computer science and OS, manual disk partition is less necessary than before for a casual user. Nevertheless, Linux provides necessary tools for disk partition table manipulation and they are introduced in this section.

9.2.1 Disk Partition

Disk partitioning refers to the action of creating one or more regions on secondary storage (e.g., disk) so that they can be managed separately, as if they are different "virtual disks". An example of disk partitioning on a Windows PC would be to partition a $1TB$ hard drive into $512GB$ of `C:\` drive and $512GB$ of `D:\` drive. Partitioned regions are logically separated. The partitions locations and sizes are stored in the disk partition table.

Some of the reasons for using disk partition include:

- Due to capability limitation, OS cannot handle a very large disk storage as a whole (this is less the case nowadays).
- Different types of files, for example system data and user data, can be stored separately for easy management.
- Different filesystems can be used on different partitions.
- Different partitions can be configured differently with unique settings.
- Sometimes partitioning can speed up hard disk accessing.

9.2.2 Disk Partition Table Manipulation

From the output of `lsblk` command earlier, it is clear that the system's hard disk name is "`sda`". To get a bit more details on this disk and its current partitioning status, use

```
$ sudo fdisk -l | grep sda
Disk /dev/sda: 931.51 GiB, 1000204886016 bytes, 1953525168 sectors
/dev/sda1    2048      4095      2048   1M BIOS boot
/dev/sda2    4096   1054719   1050624 513M EFI System
/dev/sda3 1054720 1953523711 1952468992 931G Linux filesystem
```

From the above result, it can be seen that the disk registered under `/dev/` (this is where the devices are represented by default) has been partitioned into 3 partitions, and the user space that can be further modified is `/dev/sda3` which is $931GB$.

There are variety of tools that can be used for disk partitioning. A common way of doing that is to use

```
$ sudo fdisk /dev/sda3
```

to enter the fdisk utility, and follow the wizard.

Other tools such as `cfdisk` and `parted` can also be used similarly for disk partition.

9.3 Mount, Unmount and Format a Partition

The hard disk can be formatted by partition, from where a new filesystem can be created. To format a partition, double check using `lsblk` to make sure that it is not mounted in the system. Use `sudo umount <partition name>` to unmount a partition.

Use `sudo mkfs` to format and create a new formatted filesystem, then use `mount` to mount it back to the OS. The mount of a partition needs to be recorded into `/etc/fstab` so that the OS would remember it after a reboot.

10

Shell Advanced

CONTENTS

On top of Chapter 2, more advanced shell commands and script programming skills are introduced in this chapter.

10.1 Service Control

There are many services running in the background of the OS, some of which started by the OS while the other by the user. For example, *Apache service* might be used when the system is hosting a webpage. Other commonly used services include keyboard related services, bluetooth services, etc.

To quickly have a glance of the running services, use

```
$ systemctl --type=service
```

These services can be managed using service managing utilities such as `systemctl` and `service`. Some commonly used terminologies are concluded in Table 10.1 with explanations about their differences.

In short, `systemd` is the back-end service of Linux that manages the services. Both `systemctl` and `service` are tools to interact with `systemd` (and other back-end services) to manage the services. Generally speaking, `systemctl` is more straightforward, powerful and more complicated to use, while `service` is usually simpler and user-friendly.

Use the following commands to check the status of a service, and start, stop or reboot the service.

```
$ sudo systemctl status <service name>
$ sudo systemctl start <service name>
$ sudo systemctl stop <service name>
$ sudo systemctl restart <service name>
```

Use the following commands to enable and disable a service. An enabled

TABLE 10.1
Commonly seen terminologies regarding service control.

Term / Tool name	Description
`systemd`	The `systemd`, i.e., *system daemon*, is a suite of basic building blocks for a Linux system that provides a system and service manager that runs as PID 1 and starts the rest of the system.
`systemctl`	The `systemctl` command interacts with the `systemd` service manager to manage the services. Contrary to `service` command, it manages the services by interacting with the `Systemd` process instead of running the init script.
`service`	The `service` command runs a pre-defined wrapper script that allows system administrators to start, stop, and check the status of services. It is a wrapper for `/etc/init.d` scripts, Upstart's `initctl` command, and also `systemctl`.

service automatically starts during the system boot, and a disabled service does not.

```
$ sudo systemctl enable <service name>
$ sudo systemctl disable <service name>
```

Use the following command to mask and unmask a service. A masked service cannot be started even using `systemctl start`.

```
$ sudo systemctl mask <service name>
$ sudo systemctl unmask <service name>
```

The `service` command can be used in a similar manner as follows.

```
$ sudo service <service name> status
$ sudo service <service name> start
$ sudo service <service name> stop
$ sudo service <service name> restart
```

10.2 Advanced Shell Programming

...

11

Software Management Advanced

CONTENTS

In this chapter, advanced software management approaches are introduced. These approaches may relate to both OS and user applications management.

11.1 *Git*

Git is a distributed version-control system for tracking changes in source code during the software development and deployment, and it can be used in cross platforms including Windows, Unix/Linux and MacOS. This section introduces the basic use of *Git* on a local Linux machine and on a remote host such as *GitHub*. Some contents in this section is taken from an earlier article *A Git Tutorial*.

11.1.1 Brief Introduction to *Git*

Git, created by Linus Trovalds in 2005, is a distributed version-control system for tracking changes in source code and files. It is helpful with maintaining data integrity during the coordinated development of a software in distributed non-linear work flows. *Git* is free and open-source software under GNU general public license.

With *Git*, all computers participating in the software development store a copy of the full-fledged repository locally with complete history, and it can synchronize with a centralized remote server. *Git* uses “master” and “slave”

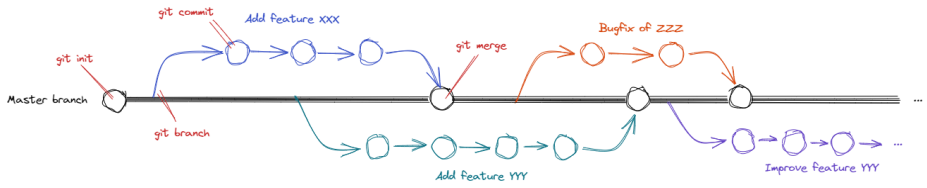

FIGURE 11.1
Git for software development management.

branches to manage the concurrent development of different features of the project, where the "master branch" is the stable and shared repository among everyone, and the "slave branches" are copies of the master branch where individual features can be developed. For a slave branch, once its developed feature is approved, it can be merged back to the master branch. A demonstration is given in Fig. 11.1.

CLI is often used to manage a *Git* repository. For example, `git init` starts a new repository with an empty master branch, and `git branch <branch-name>` creates a new slave branch from the master branch, etc. Some of these commands are shown in Fig. 11.1 and more details are introduced later. Notice that a graphical user interface is also available to interact with `git`. However, in the scope of this notebook, command line is mostly used.

11.1.2 Installation and Basic Configurations

Git and its relevant documents can be obtained from its official website *https://git-scm.com/*. In many Linux distributions, *Git* is pre-installed. If *Git* is missing in the machine, use the following command to install it

```
$ sudo apt install git
```

Upon successful installation, it is recommended to use `git config` for some basic configurations.

Notice that there are two types of configurations, namely the global configurations (apply to the machine and the user), and the repository configurations (apply to a particular *Git* repository). By default, the global level configurations are stored under `~/.gitconfig` and the repository level configurations under `./.git/config` in the repository. A good practice to edit these configurations is to use *Git* commands rather than editing the files directly.

To add user name and email to the global configuration, use

```
$ git config --global user.name '<user name>'
$ git config --global user.email <user email>
```

and use either `git config --global -l` or `cat ~/.gitconfig` to confirm the global configurations.

To revoke a global configuration, use

```
$ git config --global --unset <configuration>
```

For example,

```
$ git config --global --unset user.name
```

removes the user name.

More details about `git config` can be found at *https://git-scm.com/docs/git-config*. Usually, the user name and email configurations are mandatory, as they are very useful information in developing collaborative projects.

11.1.3 Local Repository Management

Navigate to the project directory. Use the following command to create a new *Git* repository for this project.

```
$ git init
```

From this point forward, *Git* monitors everything that happens inside this directory and its subdirectories, and try tracking changes to the files. Further *Git* commands can be applied to either check status or execute changes to the repository.

Git can spread a slave branch from the master branch, and merge the modified slave branch back to the master branch. When working on a branch, either master or slave, *Git* is able to do version-tracking for the changes made to the branch. The version-tracking related features and operations are introduced first, followed by the branch spreading and merging features.

Version-tracking

For convenience, without loosing generality assume that there is only a single branch in the project, namely the master branch. Notice that when there are multiple branches, the version-tracking works the same for each and every branch in a separate and independent manner.

The `./.git/` subdirectory separates the project directory into two areas, outside `./.git/`, namely the workspace, and inside `./.git/`, namely the *Git* repository. The workspace has the up-to-date project contents and it is directly managed by the user, while the *Git* repository is managed by *Git*, and it is accessible by the user using the `git` commands. Inside the *Git* repository are metadata of the workspace files such as which files have been changed since the last version, which files are newly added to the project, etc., and also a full back up of every historical versions for the project. It is worth mentioning that instead of recording the changes of a file from version to version, *Git* records the snapshot of the file in every version, unless it is left untouched between two consecutive versions.

Figure 11.2 gives a demonstration of how *Git* manages the project directory. *Git* classifies files in the workspace into different status, as shown in

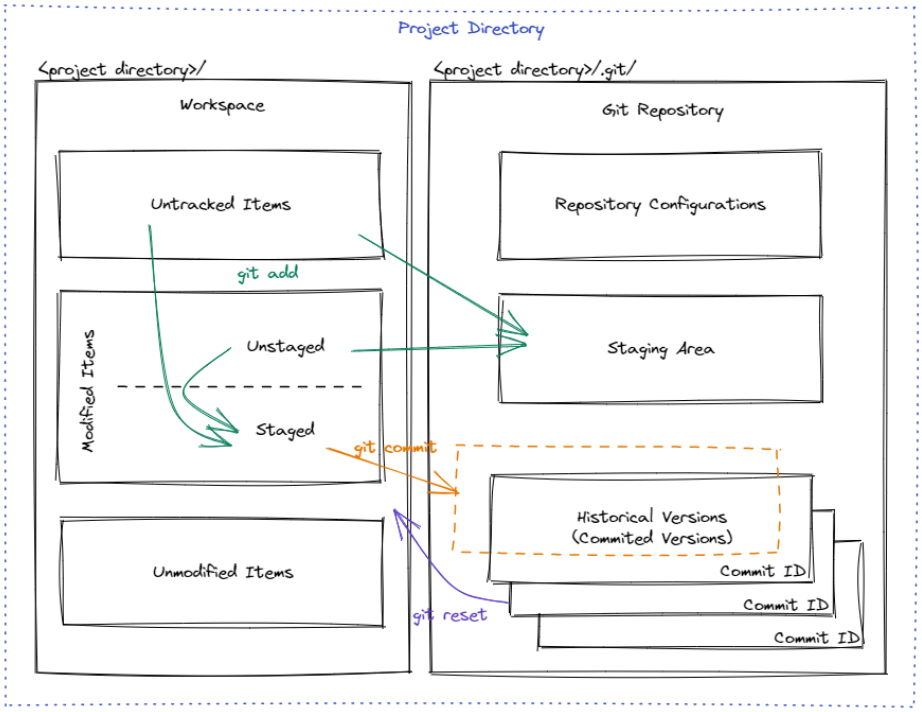

FIGURE 11.2
The project directory managed by *Git*.

Fig. 11.2. A brief explanation of each types is given in Table 11.1. Detail explanations of "stage" and "commit" are given later.

Use `git status` to check the file status in the project. An example is given below.

```
$ git status
On branch master

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:  chapters/ch-software-management-advanced/ch.tex

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:  appendix/ap.tex
        modified:  main.pdf

Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

TABLE 11.1
Different file status in a *Git* managed project.

Status	Description
Untracked	Newly added or renamed items in the project directory.
Modified (Unstaged)	Modified items from the last version that has not been registered in the staged area.
Modified (Staged)	Modified items from the last version that has been registered in the staged area. An untracked item can also be staged, in which case it also falls into this category.
Unmodified	Unmodified items from the last version.

```
chapters/ch-software-management-advanced/figures/
```

where `ch.tex` is a modified (staged) item; `ap.tex` and `main.pdf` modified (unstaged) items, and `figures/` an untracked item.

It takes a two-step procedure to back up the project in the *Git* repository. In the first step, the user flags the changed items (either newly added, renamed, or modified) to be backed up in the next version. In the second step, the user actually backs up the items. The first and second steps are called "stage" and "commit", respectively. Notice that it is possible to run a single line of command to execute both steps, but logically it still takes two steps.

Git tracks the name and content of the items that the user has staged in the "staging area", as shown in Fig. 11.2. The items in this area will be backed up in the next commit. To stage an item, use

```
$ git add <item name>
```

which registers the item in the staging area, thus also changes its status from untracked or modified (unstaged) to modified (staged). If an item is modified after it has been staged, *Git* will distinguish the "staged portion" and "unstaged portion" of that item. If using `git status` to check its status, the item will be listed as both staged and unstaged. Unstaged items, either untracked or modified, will remain its status after the commit. Sometimes for convinience, `git add -A` can be used to add all untracked or modifed items to the staging area.

Use `git commit` to commit the repository as follows.

```
$ git commit [<item name>]
```

The above command commits the project and creates a version in the *Git* repository. It is possible to specify items, in which case *Git* only commits the specified items and leave the rest items as they are. A commit ID is automatically assigned to the commit. Notice that the user will be asked to provide a "comment message" with the commit, which should be used to briefly explain what has been changed in this commit.

A flag `-a` with `git commit` stages all changes made to the project, then implements the commit command. A flag `-m` simplifies the message recording process and allows the user to key in the message directly after the command. An example is given below.

```
$ git status
On branch master

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:  A Notebook on Linux/chapters/ch-software-management-
            advanced/ch.tex
        modified:  A Notebook on Linux/main.pdf

no changes added to commit (use "git add" and/or "git commit -a")

$ git commit -am "add introduction to git command"
[master e2e977e] add introduction to git command
 2 files changed, 5 insertions(+), 4 deletions(-)

$ git status
On branch master

nothing to commit, working tree clean
```

To check the commit logs, i.e., all historical commits including their associated timestamps, authors, commit IDs and comment messages, use `git log` as shown in the example below. Notice that the commit logs can be very long. Only a few commits are given for illustration purpose.

```
$ git log
commit e3475e673d8c2a087de6b4423188c51e80af3e5d (HEAD -> master)
Author: sunlu <sunlu.electric@gmail.com>
Date:  Wed Aug 31 15:16:52 2022 +0800

    git

commit 3ab6d1473a7e48d4d890509ddb5a87274c023e6c
Author: sunlu <sunlu.electric@gmail.com>
Date:  Tue Aug 30 15:50:06 2022 +0800

    k8s

commit f6a1e3d779305e966602ecd7589c05f56dd2ad0f
Author: sunlu <sunlu.electric@gmail.com>
Date:  Mon Aug 29 16:31:18 2022 +0800

    more on docker and kubernetes
```

```
commit e2de5b28db7982f57d0ad51361d5161b884f2efe
Author: sunlu <sunlu.electric@gmail.com>
Date:  Sun Aug 28 21:38:48 2022 +0800

    add docker sections
```

where notice that `HEAD` is a reference that points to the latest commit in a branch. Filters can be added as optional arguments for the `git log` command. More details are given at *https://git-scm.com/docs/git-log*.

And finally, to restore to a previous commit version, use `git reset` or `git revert`. Notice that `git reset` and `git revert` can both be used to restore the workspace to a previous stage, but they differ significantly. In general, `git reset` "erases" the past commits, and it is often used in a private branch; `git revert` on the other hand create a new commit that undoes all the changes, and it is often used in a public branch.

The command `git reset` is often used as follows.

```
$ git reset <option> <commit ID>
```

where `<option>` is often `--hard`, `--mixed` or `--soft`, and `<commit ID>` can be the ID of any commit in the git log, or for shorcut `HEAD` the latest commit, `HEAD^` or `HEAD~1` the second latest commit, `HEAD~2` the third latest commit, and so on.

The options `--hard`, `--mixed` and `--soft` work differently. All these options move `HEAD` to the specified commit, and remove all commits afterwards from the repository. The `--hard` option reverts the workspace back to when the specified commit happened (meaning that there would be no way to undo the reset command). Both `--mixed` and `--soft` do not change the workspace. The `--mixed` option leaves all the changes from the specified commit to today as unstaged, while `--soft` leaves them as staged. If no `--hard`, `--mixed` or `--soft` is given, `--mixed` will be used as the default option.

Notice that the above `git reset` approaches does not help if an undesigned commit has already been pushed and synchronized to a remote server, in which case `git revert` should be used instead. More details are given in Section 11.1.4.

Branch Management

"Branch" is one of the core features of *Git*, and it plays an important role in collaborative development of a project. There are two types of branches, namely the local branches and the remote branches. The remote branches are mainly used for sharing and synchronizing the project development with others. The details will be introduced in Section 11.1.4. Only local branches are considered in this section.

To list down all the local branches, use

```
$ git branch
```

and the current working branch will be highlighted. The current working branch is also referred as the "head branch" or the "active branch".

To create a new branch from the current branch, or to delete a branch, use

```
$ git branch <branch name> [<commit ID>]
$ git branch -d <branch name>
```

respectively. The optional `<commit ID>` when creating a new branch allows the user to create a branch on top of a specified historical commit.

To rename a branch, use

```
$ git branch -m [<old branch name>] <new branch name>
```

where if no `<old branch name>` is specified, the current working branch will be renamed.

To switch to a different working branch, use `git checkout` or `git switch` as follows.

```
$ git checkout <branch name>
$ git switch <branch name>
```

Notice that when there are uncommited changes in the current branch, *Git* may or may not forbid the user to switch to another branch (the rules are too complicated to be explained here). Therefore, it is recommended to commit the changes before switching to a different branch. When switching to another branch, the workspace will change accordingly to the target branch.

To merge a branch back to the current branch, use

```
$ git merge <branch name>
```

and fill in the comments accordingly. Depending on the setup, *Git* may or man not automatically delete the merged branch.

An alternative way of integrating two branches is `git rebase`, which "rewires" the two branches into a linear single branch. Use `git rebase` as follows.

```
$ git rebase <branch name>
```

A demonstrative Fig. 11.3 explains the differences between `git merge` and `git rebase`. It is clear from Fig. 11.3 that by using `git rebase`, all feature commits are integrated into the master repositories to form a single repository tracking line, which differs largely from `git merge`.

11.1.4 Remote Repository Management

GitHub is an internet hosting service for software development and sharing using *Git*. A user can create, manage, share and co-develop remote repositories on *GitHub*. Notice that *GitHub* is not the only place to host remote repositories. Some alternatives include *GitLab* and *Gitee*. Using *Git*, it is possible to

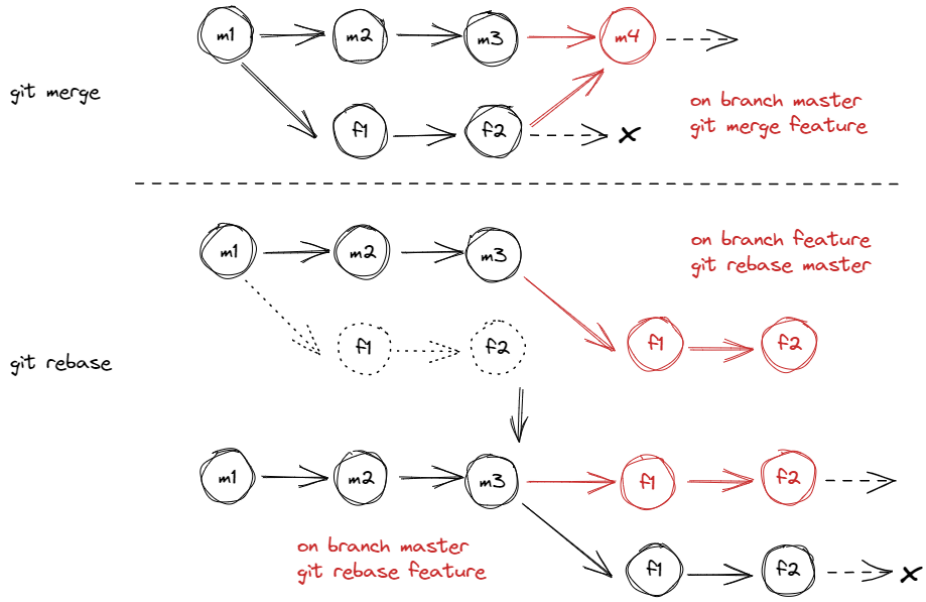

FIGURE 11.3
Two approaches of integrating branches, `git merge` VS `git rebase`.

build a remote repository hosting server at home on a regular computer. In this section, only *GitHub* is considered.

An *https* URL is associated with each remote repository on *GitHub*, for example *https://github.com/torvalds/linux.git* for Linux kernel. This URL can be used to download the remote repository to a local machine, or to link (synchronize) a local repository with that remote repository.

Consider the case where there is already a remote repository, either under the user's *GitHub* account or from the community. To download a clone of an existing repository to the local machine, use

```
$ git clone <repository URL> [<local directory>]
```

and to maintain the local repository up-to-date with the remote repository, regularly use

```
$ git remote update
$ git pull
```

to synchronize the local repository with the remote one.

> One may argue whether it is a good idea to use `git pull`. Although convenient, `git pull` is an integration of two commands, `git fetch` and `git merge` (or `git rebase`) executed sequentially. It may be "safer" to manually run the two commands separately.

The local commits can also be pushed to the remote repository using

```
$ git push
```

if the remote repository is under the user's account, or if the owner of the remote repository gives the user the permission. Notice that when pushing updates to the remote repository, *GitHub* may require login credentials for permission checks.

Consider the case where there is already a local repository, and an empty repository has just been created on *GitHub*. To setup the connection, navigate to the local repository and use

```
$ git remote add <remote repository name> <repository URL>
```

which will register the remote repository URL in the local configuration file.

The next step is to map the local current working branch with a branch in the remote repository, which would be used as the default source/target branch when running `git pull` and `git push`. This can be done as follows.

```
$ git branch --set-upstream-to <remote repository name> <remote
    repository branch>
```

Notice that this step is not required if the project is cloned from a remote repository using `git clone` introduced earlier.

From there on, use `git push` and `git pull` normally.

Finally, to revert a commit that has already been pushed to a shared server, using `git reset` in the local repository to erase the faulty commit, then `git push` to push the erased repository to the remote repository would not work. This is because by erasing the latest commit locally, the local commit will be "behind" the remote repository, making *Git* think that the local repository needs to be updated to follow the remote repository.

To tackle this problem, use `git revert` instead as follows.

```
$ git revert <commit ID>
```

Different from `git reset`, this command creates a new commit that mirrors the specified historical commit. The new commit can then be pushed to the shared server.

11.2 Linux Repository Management

...

11.2.1 Brief Introduction to Linux Repository

...

Part III

Server Management

12

Database

CONTENTS

Database and its manage systems are introduced in this chapter. SQL is introduced. Both relational and non-relational databases are used as examples to demonstrate how database can be used on a Linux system.

12.1 Introduction to Database

Database, in a broad view, refers to an organized collection of data of any format. In this sense, any file format that hosts information in an meaningful and explainable way, such as *csv*, *XML*, or *JSON*, is a database. These file formats often work fine when the data is stored in a centralized manner and its size small.

As the data size grows, the robustness and efficiency of data storage and retrieval may become an issue. Therefore, different database models have been proposed for more robust and efficient data storage and retrieval. Dedicated software, namely the database management systems (DBMS), are developed to manage and maintain the database and provide an interface for the users to create, retrieve, update and delete data. Different database models require different database engines and different types of DBMS.

There are hundreds of database models and engines available in the market. In general, the most widely seen database models can be divided into

two categories, namely the relational databases (RDS) and the non-relational databases.

Relational databases proposed in 1980s use rows and fixed columns in a series of fixed tables to store data. The tables are related to each other via shared columns. RDS is intuitive and efficient in most occasions. Structured query language (SQL), a domain-specific language, is used in managing RDS and interfacing relational DBMS.

Non-relational databases proposed in 2000s, on the other hand, usually store data in key-value pairs instead of tables. A widely acknowledged non-relational database storage format is *JSON*.

Non-relational databases can be very fast in query, and are becoming popular in handling big data and in real-time web applications. However, many non-relational databases compromise consistency issue, and can only provide "eventual consistency", which implies that the latest update in the non-relational database may not be reflected in an immediate query. Non-relational databases can be accessed by NoSQL, a collective set of languages dedicated for non-relational database management.

Unlike SQL which applies to almost all RDS, there is no universally adopted language for NoSQL.

In this chapter, both relational and non-relational databases are introduced in further details to illustrate the database service on a Linux system. In particular, SQL is also briefly introduced as it is a very important subject in managing a relational database.

12.2 Relational Database and SQL

Relational database is so far the most commonly used digital database. Details are given in the rest of the section.

12.2.1 Brief Introduction to Relational Database

Relational database is proposed in 1970s by IBM. Some important features of a relational database includes the following.

- Structure the data as "relations", which is a collection of tables, each consisting of a set of rows (also known as tuple/record) and columns (also known as attribute/field).
- Provide relational operations the manipulate the data in the tables, for example, joining tables together and aligning them using an attribute.

Examples of relational databases include *Oracle*, *MySQL*, *Microsoft SQL Server*, *MariaDB*, *IBM Db2*, *Amazon Redshift* and *Amazon Aurora*. Most

TABLE 12.1
An example of a relational database table.

user				
`user_id`*	`user_password_sha256`	`user_email`	`membership`	`referee-id`
sunlu	xxxxxxxx	sunlu@xxx.com	premium	NULL
xingzhe	yyyyyyyy	xingzhe@yyy.com	basic	sunlu
...	...	...	...	...

TABLE 12.2
A second database table in the example.

membership		
`membership_type`	`monthly_price`	`annual_price`
none	0	0
basic	5	50
premium	10	80

commercialized relational databases adopt SQL as the query language. There are alternative languages, but are rarely used compared to SQL.

12.2.2 Tables

An example of a table used in RDS is given in Table 12.1, where the table has a name as the identifier, `user`. In this table, there are 5 attributes, namely `user_id`, `user_password_sha256`, `user_email`, `membership` and `referee_id`.

A table should have an attribute (or a set of attributes) defined as the primary key. In the example given by Table 12.1, `user_id` as denoted by the asterisk. The primary key is used to uniquely identify a row in the table. When a key is made up of multiple attributes, it is called a composite key.

The primary key can be either a surrogate key, which is generated by the table for recording purpose and is not derived from application data (for example, product serial id, etc.), or a natural key, which reflects meaningful information in real world (for example, email, citizenship IC number, etc.).

A foreign key is the attribute(s) that link a table to another table. It is the primary key of another table that in someway connects to this table. For example, the `membership_type` attribute in Table 12.1 could be the foreign key that links to another Table 12.2. Notice that a foreign key doe not necessarily need to have the same name as the primary key in another table (though their links are there), it is a good practice to keep them consistent.

A table can have only one primary key, but can have multiple foreign keys. The foreign key can not only relate a table to another, but can also relate a table to itself. For example, the "referee-id" attribute in Table 12.1 could be

the foreign key that relates to "user-id" of itself, from which we know that user the referee of user "xingzhe" is user "sunlu".

Primary key and foreign keys designs form the database schema. They together defines and describes the tables and relations between them.

Naming conventions shall apply to the databases, tables and columns. Some rules and good practices are concluded as follows.

- Use natural collective terms instead of plurals. For example, "staff" but not "employees".
- Use only letters, numbers, and underscores.
- Begin with a letter and may not end with an underscore.
- Avoid using abbreviations unless commonly understood.
- Avoid using prefixes.
- Table:
  - Do NOT use the same name for a table and one of its columns.
  - Do NOT concatenate two table names to create a third relationship table.
- Column:
  - Use singular name for columns.
  - Avoid using over simplified terms such as "id".
  - Use only lowercase if possible.
- Alias:
  - Use keyword AS to indicate an alias.
  - The correlation name should be the first letter of each word of the object name.
  - If there is already the same correlation name, append a number.
- Stored procedure. Always contain a verb in the name of a stored procedure.
- Uniform suffixes:
  - `_id`: primary key.
  - `_status`: flag values.
  - `_total`: the total number of a collection of values.
  - `_num`: a number.
  - `_date`: a date.
  - `_name`: the name of a person or product.

12.2.3 SQL

SQL is the most widely language for interacting with relational DBMS for data query and maintenance. SQL is very powerful and flexible in its full capability. In this section, only the basic SQL operations are introduced.

Notice that there might be slight differences in the SQL for different DBMS, depending on their associated unique features. Most of the commands, especially the widely used ones, shall be universally consistent.

SQL is a hybrid language consisting of the following 4 types of languages.

- Data query language: query information and metadata of a database.
- Data definition language: define database schemas.
- Data control language: control user access and permission to a database.
- Data manipulation language: insert, update and delete data from a database.

SQL supports variety of data types, and different DBMS may cover slightly different data types. Some of the most commonly used data types are summarized in Table 12.3 and they shall be universally consistent. For the full list of data types that a DBMS supports, check the manuals and documents of that DBMS.

SQL defines reserved keywords for database manipulation. The keywords have specific meanings and cannot be used as user-defined variable names. Commonly used SQL keywords are summarized in Tables 12.4, 12.5 and 12.6.

General Rules

All SQL commands shall end with a semicolon "`;`".

The programming of SQL shall follow the following general rules wherever possible. This helps to maintain the good quality and portability of the code.

- Use standard SQL functions instead of vendor-specific functions for better portability.
- Do NOT use object-oriented design principles in SQL or database schema.
- Use UPPERCASE for keywords.
- Use `/*<comments>*/` to add comments to the code, otherwise precede comments with `-- <comments>` and finish them with a new line.

The naming of database, tables and columns shall follow conventions introduced in Section 12.2.2.

During the coding, follow the following rules.

- Use spaces to align the codes.
- Use a space before and after equals (=), after commas (,).

TABLE 12.3
Widely used SQL data types.

Data Type	Description
INT/INTEGER	Integer, with a range of -2147483648 to 2147483647. When marked "UNSIGNED", the range becomes 0 to 4294967295. Some relevant data types are TINYINT, SMALLINT, MEDIUMINT, and BIGINT, which have a different range.
DEC/DECIMAL(size,d)	An exact fixed-point number. The total number of digits and the number of digits after decimal point are specified by "size" and "d", respectively. Some relevant data types are DOUBLE(size,d), which can also be used to specify a floating point. Notice that DEC/DECIMAL is usually preferable in most occasions.
CHAR(size)	A fixed length string with the specified length in characters.
VARCHAR(size)	A variable length string, with the specified maximum string length in characters.
BOOL/BOOLEAN	This is essentially a 1-digit integer, where 0 stands for "false" and other values stand for "true".
BLOB	A binary large object with maximum 65535 bytes.
DATE	A date by format "YYYY-MM-DD".
TIME	A time by format "hh:mm:ss".
DATETIME	A combination of date and time by format "YYYY-MM-DD hh:mm:ss".
TIMESTAMP	A timestamp that measures the number of seconds since the Unix epoch. The format is "YYYY-MM-DD hh:mm:ss". Unlike DATETIME, TIMESTAMP specifies an exact point in time, thus is not affected by timezone, etc.

TABLE 12.4
Widely used SQL keywords (part 1: names).

Keyword	Description
CONSTRAINT	A constraint that limits the value of a column.
DATABASE	A database.
TABLE	A table.
COLUMN	A column (attribute, field) of a table.
VIEW	A view, which is a virtual table which does not store data by itself and only reflects the base tables data.
INDEX	An index, which is a pre-scan of specific column(s) of a table and can be used to speed up future queries related to the column(s). Notice that unlike a view, an index needs to be stored together with the table.
PRIMARY KEY	The primary key of a table.
FOREIGN KEY	A foreign key defined in a table that links to a (different) table.
PROCEDURE	A procedure that defines a list of database operations to be executed one after another

TABLE 12.5
Widely used SQL keywords (part 2: actions).

Keyword	Description
CREATE	Create a database (CREATE DATABASE), a table (CREATE TABLE), an index (CREATE INDEX), a procedure (CREATE PROCEDURE).
ADD	Add a column in an existing table, or a constraint to an existing column.
ALTER	Modify columns in a table (ALTER TABLE), or a data type of a column (ALTER COLUMN).
SET	Specify the columns and values to be updated in a table.
DROP	Delete a column (DROP COLUMN), a constraint (DROP CONSTRAINT), a database (DROP DATABASE), an index (DROP INDEX), a table (DROP TABLE), or a view (DROP VIEW).
CHECK	Define a constraint that limits the value that can be placed in a column.
DEFAULT	Define a default value for a column.
INSERT INTO	Insert a new row into a table.
UPDATE	Update an existing row (tuple, entity) in a table.
DELETE	Delete a row (tuple, entity) from a table.
EXEC	Executes a stored procedure.

TABLE 12.6
Widely used SQL keywords (part 3: queries).

Keyword	Description
SELECT	Query data from a database. Relevant combinations are SELECT DISTINCT which returns only distinct values; SELECT INTO which copies data from one table into another; SELECT TOP which returns part of the results.
AS	Assign an alias to a column or table.
FROM	Specify the table where the operation is run.
WHERE	Filter results that fulfill a specified condition.
IN	Specify multiple values in a WHERE clause.
AND	Select rows where both conditions are true.
OR	Select rows where either condition is true.
ALL	Return true if all followed sub-query values meet the condition.
ANY	Return true if any followed sub-query value meet the condition.
BETWEEN	Select values within a given range.
ORDER BY	Sort the results in ascending or descending order.
JOIN	Join tables for query. Relevant combinations are OUTER JOIN, INNER JOIN, LEFT JOIN and RIGHT JOIN.
EXISTS	Tests for the existence of any record in a sub-query.
GROUP BY	Groups the result set when using aggregate functions (COUNT, MAX, MIN, SUM, AVG).
UNION	Combines the result sets of multiple select statements.

- Use BETWEEN and IN, instead of combining multiple AND and OR clauses.

When creating a table, follow the following rules.

- Choose standard SQL data types.
- Specify default values and set up constraints, and put them close to the declaration of the associated column name.
- Select primary key carefully and keep it simple.
- Specify the primary key first right after the CREATE TABLE statement.
- Implement validation. For example, for a numerical value, use CHECK to prevent incorrect values.

Database Manipulation

To list down all the databases running on the server, use

```
SHOW DATABASES;
```

To create a database, use

```
CREATE DATABASE <database-name>;
```

To select a database, use

```
USE <database-name>;
```

To delete a database, use

```
DROP DATABASE <database-name>;
```

Table Manipulation

Tables are the fundamental components in an RDS. Some features of a table have been introduced in Section 12.2.2. To create a commonly seen table, use

```
CREATE TABLE <table-name> (
    PRIMARY KEY (<column-name>),
    <column-1>        <data-type>    <constraint>,
    <column-2>        <data-type>    <constraint>,
                      CONSTRAINT <constraint-name-1>
                      CHECK(<constraint-rule>),
                      CONSTRAINT <constraint-name-2>
                      CHECK(<constraint-rule>)
);
```

TABLE 12.7
Commonly used constraints.

Constraint	Description
`NOT NULL`	Not allowed to be NULL.
`UNIQUE`	Not allowed to have duplicated values.
`PRIMARY KEY`	Set as primary key, thus, must be not NULL and must remain unique.
`FOREIGH KEY`	Set as foreign key.
`DEFAULT <value>`	Set a default value.
`AUTO_INCREMENT = <value>`	Each time a new row is inserted and NULL or 0 is set for this column, instead of set the column to NULL or 0, automatically generate the next sequence number. The starting value is defined by `<value>` which by default is 1.

where notice that for demonstration purpose, 2 columns and 2 constraints are defined.

The `<constraint>` that comes after the data type of a column is used to set an additional restriction to the data in the table. When such restriction is violated, an error would raise to stop the operation. For example, if `NOT NULL` is set as a constraint, then when inserting a row to the table later, the user cannot input NULL for that specific column. Notice that the primary key can also be set in this way, as a constraint named `PRIMARY KEY`, although it is a better practice to use `PRIMARY KEY (<column-name>)`.

Commonly used such constraints are summarized into Table 12.7. As shown by Table 12.7, a default value can be assigned to a column by using the `DEFAULT <value>` constraint. When inserting a new row, the column of that row will be assigned to its default value if no other value is assigned. If no such statement is provided for a column, its default value is NULL.

Upon creation of a table, its basic schema can be reviewed using

```
DESCRIBE <table-name>;
```

To list down existing tables, use

```
SHOW TABLES;
```

To delete a table, use

```
DROP TABLE <table-name>;
```

To edit the column of a table, use either of the following

```
ALTER TABLE <table-name>
ADD <column-name> <data-type>; -- add new column
ALTER TABLE <table-name>
DROP COLUMN <column-name>; -- drop column
```

```
ALTER TABLE <table-name>
RENAME COLUMN <old-name> TO <new-name>; -- rename column
ALTER TABLE <table-name>
MODIFY COLUMN <column-name> <data-type>; -- modify column data type (
    depending on DBMS, syntax may differ)
```

and

```
ALTER TABLE <table-name>
ADD CONSTRAINT <constraint-name> CHECK(<constraint-rule>); -- add
    constraint
ALTER TABLE <table-name>
DROP CONSTRAINT <constraint-name>; -- drop constraint
```

Notice that it is possible to change the primary key of a table similarly, because essentially the primary key is treated as a constraint named "primary".

The foreign key is a key used to point to another table. Therefore, the foreign key can be nominated only after the other table has been created. Declare foreign key upon creation of a table as follows. As mentioned, to do this, the other tables must be created beforehand.

```
CREATE TABLE <table-name> (
   PRIMARY KEY (<column-name>),
   <column-1>        <data-type>   <constraint>,
   <column-2>        <data-type>   <constraint>,
                     CONSTRAINT <constraint-name-1>
                     CHECK(<constraint-rule>),
                     CONSTRAINT <constraint-name-2>
                     CHECK(<constraint-rule>),
   FOREIGN KEY (<column-name>) REFERENCES <referred-table-name>(<
       referred-column-name>), -- one way to define foreign key
                     CONSTRAINT <constraint-name-3>
                     FOREIGN KEY (<column name>)
                     REFERENCES <referred-table-name>(<referred-column
                         -name>) -- another way to define foreign key
);
```

where, as can be seen, foreign key is just treated as a constraint in the table.

To define a foreign key in an existing table, use

```
ALTER TABLE <table-name>
ADD FOREIGN KEY (<column name>) REFERENCES <referred-table-name>(<
    referred-column-name>); -- one way
ALTER TABLE <table-name>
ADD CONSTRAINT <constraint-name> FOREIGN KEY (<column name>) REFERENCES
     <referred-table-name>(<referred-column-name>); -- another way
```

To drop a foreign key, use

```
ALTER TABLE <table-name>
DROP CONSTRAINT <constraint-name>;
```

where put the constraint name associated with the foreign key at `<constraint-name>`. There are variety ways of checking the constraints names of a table. An example is given below.

```
SELECT TABLE_NAME, CONSTRAINT_TYPE, CONSTRAINT_NAME
FROM information_schema.table_constraints
WHERE table_name=<table-name>;
```

One thing to note is that upon creation of a foreign key, the referred column becomes the "parent" and the foreign key becomes a "child". As long as the child exists, the parent cannot be removed from its table. This helps to protect the schema of the database. Should there be any quest to break the schema, this restriction can be overwritten. When defining the foreign key, add an additional claim "ON DELETE SET NULL" or "ON DELETE CASCADE" as follows.

```
FOREIGN KEY (<column name>) REFERENCES <referred-table-name>(<referred-
    column-name>) ON DELETE SET NULL
FOREIGN KEY (<column name>) REFERENCES <referred-table-name>(<referred-
    column-name>) ON DELETE CASCADE
```

in the first scenario, the child foreign key will be set to NULL, while in the second scenario, the child relevant rows will be removed.

Row Manipulation

To insert a row into a table, use

```
INSERT INTO <table-name> VALUES (<content>, <content>, ...);
```

where the contents shall follow the field sequence as shown by the `DESCRIBE <table-name>` command. To specify the column name while inserting a row, use

```
INSERT INTO <table-name>(<column-name>, <column-name>, ...) VALUES (<
    content>, <content>, ...);
```

Notice that it is also possible to populate multiple rows of a table using one command as follows.

```
INSERT INTO <table-name>(<column-name>, <column-name>, ...)
VALUES (<content>, <content>, ...),
       (<content>, <content>, ...),
       (<content>, <content>, ...);
```

where 3 rows are inserted into the table.

Notice that if a foreign key bound exists between two tables, when inserting a row to the child table, the foreign key value of this row must already be defined in the parent table.

Use the following command to query all items in a table, which can be used to check whether the row is added to the table correctly.

```
SELECT * FROM <table-name>;
```

To modify the attributes of specific row(s), use

```
UPDATE <table-name>
SET <column-name> = <value>, ...
WHERE <filter-criteria>;
```

where `<filter-critera>` is used to filter the rows to which the update is carried out. Commonly used filter criteria are a set of `<cloumn-name> = <value>` separated by `AND` and `OR`. The filter criteria can be set very flexibly and more details are given in later sections. Notice that it is possible to change multiple column values together, by stacking multiple `<column-name> = <value>` separated by ",".

To delete rows from a table, use

```
DELETE FROM <table-name>
WHERE <filter-criteria>;
```

Notice that if filter criteria is not specified, i.e., if `WHERE` is missing, all items in the table will be affected.

Query

A typical query looks like the following and it returns the data in a table-like format.

```
SELECT <column-or-statistics>
FROM <table-name-or-combination>
GROUP BY <column-name>
WHERE <filter-criteria>
ORDER BY <column-name>, ...
LIMIT <number>;
```

where

- `<column-or-statistics>` describes the columns to be returned.
- `<table-name-or-combination>` describes the source of the information, either being a table, or a joint of multiple tables.
- `GROUP BY` groups rows with the same value of the specified column into "summary rows".
- `<filter-condition>` defines the filter criteria and only rows meet the criteria are returned.
- `ORDER BY <column-name>` allows the items to be returned in a specific order based on ascending/descending order. It is worth mentioning that the `<column-name>` here does not need to appear in the selected returns, and it can be multiple columns separated by ",". Use `ASC` (default) or `DESC` after each `<column-name>` to specify ascending or descending order.
- `LIMIT <number>` restricts the maximum number of rows to be returned.

Notice that `SELECT` and `FROM` statements are compulsory in all queries, `WHERE` statement very widely used, and other statements optional case by case.

Details to each field are given below.

The statement `<column-or-statistics>` mainly controls the information to be returned. Commonly seen selected items in `<column-or-statistics>` are summarized as follows.

- `*` (asterisk): return all columns.
- `<column-name>, ...`: return selected columns.
- `<table-name>.<column-name>, ...`: return selected columns, and to avoid ambiguity, specify table name with the column name.
- `<column-name> AS <alias>`: return selected columns, and use alias in the returns.
- `DISTINCT <column-name>, ...`: return only distinct rows.
- `COUNT()`, `SUM()`, `MIN()`, `MAX()`, `AVG()`: return aggregate function of a column instead of all the items in that column. They can be used along with DISTINCT, for example, `COUNT(DISTINCT <column-name>, ...)`.
- Simple calculations to the above result, for example `1.5*<column-name>`. Commonly used arithmetic operations are `+`, `-`, `*`, `/`, `%`, `DIV` (integer division).

The statement `<table-name-or-combination>` mainly indicates the source table(s). It can be a single table, a joint of multiple tables, or a nest query. More details about joint of multiple tables are illustrated below.

Consider the following example, where two tables are given as follows.

```
> SELECT * FROM test;
+---------+---------+---------+
| test_id | value_1 | value_2 |
+---------+---------+---------+
|       1 | a       |      10 |
|       2 | a       |      20 |
|       3 | a       |      30 |
|       4 | b       |     100 |
|       5 | b       |     200 |
|       6 | c       |    1000 |
|       7 | c       |    2000 |
+---------+---------+---------+

> SELECT * FROM test_join;
+--------------+---------+---------+---------+
| test_join_id | value_1 | value_2 | value_3 |
+--------------+---------+---------+---------+
```

```
| a           |      10 |     99 | alpha  |
| b           |     100 |    999 | bravo  |
| d           |   10000 |  99999 | delta  |
+-------------+---------+--------+--------+
```

There are different types of joins, namely "inner join" (or "join"), "left join", "right join" and "cross join". They are introduced as follows.

The most intuitive join is the cross join. It returns everything in the two tables like a cartesian product (that explains whey cross join is also called cartesian join), where the total number of columns are the sum of two tables, the number of rows the product of two tables, as shown below.

```
> SELECT * FROM test CROSS JOIN test_join;
+---------+---------+---------+-------------+---------+---------+---------+

| test_id | value_1 | value_2 | test_join_id | value_1 | value_2 |
    value_3 |
+---------+---------+---------+-------------+---------+---------+---------+

|       1 | a       |      10 | a           |      10 |      99 | alpha   |
|       1 | a       |      10 | b           |     100 |     999 | bravo   |
|       1 | a       |      10 | d           |   10000 |   99999 | delta   |
|       2 | a       |      20 | a           |      10 |      99 | alpha   |
|       2 | a       |      20 | b           |     100 |     999 | bravo   |
|       2 | a       |      20 | d           |   10000 |   99999 | delta   |
|       3 | a       |      30 | a           |      10 |      99 | alpha   |
|       3 | a       |      30 | b           |     100 |     999 | bravo   |
|       3 | a       |      30 | d           |   10000 |   99999 | delta   |
|       4 | b       |     100 | a           |      10 |      99 | alpha   |
|       4 | b       |     100 | b           |     100 |     999 | bravo   |
|       4 | b       |     100 | d           |   10000 |   99999 | delta   |
|       5 | b       |     200 | a           |      10 |      99 | alpha   |
|       5 | b       |     200 | b           |     100 |     999 | bravo   |
|       5 | b       |     200 | d           |   10000 |   99999 | delta   |
|       6 | c       |    1000 | a           |      10 |      99 | alpha   |
|       6 | c       |    1000 | b           |     100 |     999 | bravo   |
|       6 | c       |    1000 | d           |   10000 |   99999 | delta   |
|       7 | c       |    2000 | a           |      10 |      99 | alpha   |
|       7 | c       |    2000 | b           |     100 |     999 | bravo   |
|       7 | c       |    2000 | d           |   10000 |   99999 | delta   |
+---------+---------+---------+-------------+---------+---------+---------+
```

where notice that `CROSS JOIN` can be replaced by a comma ",".

It is clear in this table, that the two columns `value_1` from table `test` and `test_join_id` from table `test_join` is probably the logic connection point. In this context, the rows with inconsistent `test.value_1` and `test_join.test_join_id` is meaningless and shall be removed. This can be achieved by the following code.

```
> SELECT * FROM test CROSS JOIN test_join
    -> where test.value_1 = test_join.test_join_id;
+---------+---------+---------+--------------+---------+---------+---------+

| test_id | value_1 | value_2 | test_join_id | value_1 | value_2 |
    value_3 |
+---------+---------+---------+--------------+---------+---------+---------+

|       1 | a       |      10 | a            |      10 |      99 | alpha   |
|       2 | a       |      20 | a            |      10 |      99 | alpha   |
|       3 | a       |      30 | a            |      10 |      99 | alpha   |
|       4 | b       |     100 | b            |     100 |     999 | bravo   |
|       5 | b       |     200 | b            |     100 |     999 | bravo   |
+---------+---------+---------+--------------+---------+---------+---------+
```

and this is equivalent to inner join (or simply, join)

```
> SELECT * FROM test JOIN test_join
    -> ON (test.value_1 = test_join.test_join_id);
+---------+---------+---------+--------------+---------+---------+---------+

| test_id | value_1 | value_2 | test_join_id | value_1 | value_2 |
    value_3 |
+---------+---------+---------+--------------+---------+---------+---------+

|       1 | a       |      10 | a            |      10 |      99 | alpha   |
|       2 | a       |      20 | a            |      10 |      99 | alpha   |
|       3 | a       |      30 | a            |      10 |      99 | alpha   |
|       4 | b       |     100 | b            |     100 |     999 | bravo   |
|       5 | b       |     200 | b            |     100 |     999 | bravo   |
+---------+---------+---------+--------------+---------+---------+---------+
```

where `ON (<table1.column-name> = <table2.column_name>)` is used to indicate the association. The number of columns remain unchanged, but the number of rows depends on the repentance of the associated row with connection in each table. For example, for "a", `test.value_1=a` has 3 relevant rows while `test_join.test_join_id=a` has 1 relevant row. Thus, the total number of regarding "a" is $3 = 3 \times 1$. The same applies to "b".

From table `test` perspective, its rows regarding `value_1` equals to "a" and "b" are fully included in the inner join results. However, the two rows regarding `value_1=c` is omitted. This is because there is no associated row in the other table `test_join`. It is possible to protect information loss from table `test` by adding these two rows back, with all the columns from table `test_join` filled with NULL. Use left join to achieve this goal as follows.

```
> SELECT * FROM test LEFT JOIN test_join
    -> ON (test.value_1 = test_join.test_join_id);
```

```
+---------+---------+---------+-------------+---------+---------+---------+

| test_id | value_1 | value_2 | test_join_id | value_1 | value_2 |
    value_3 |
+---------+---------+---------+-------------+---------+---------+---------+

|       1 | a       |      10 | a           |      10 |      99 | alpha   |
|       2 | a       |      20 | a           |      10 |      99 | alpha   |
|       3 | a       |      30 | a           |      10 |      99 | alpha   |
|       4 | b       |     100 | b           |     100 |     999 | bravo   |
|       5 | b       |     200 | b           |     100 |     999 | bravo   |
|       6 | c       |    1000 | NULL        |    NULL |    NULL | NULL    |
|       7 | c       |    2000 | NULL        |    NULL |    NULL | NULL    |
+---------+---------+---------+-------------+---------+---------+---------+
```

Alternatively, think of this as temporarily adding one row to `test_join` with `test_join_id=c` and everything else NULL, before the joining.

The same idea applies to right join as well, as shown below.

```
> SELECT * FROM test RIGHT JOIN test_join ON (test.value_1 = test_join.
    test_join_id);
+---------+---------+---------+-------------+---------+---------+---------+

| test_id | value_1 | value_2 | test_join_id | value_1 | value_2 |
    value_3 |
+---------+---------+---------+-------------+---------+---------+---------+

|       1 | a       |      10 | a           |      10 |      99 | alpha   |
|       2 | a       |      20 | a           |      10 |      99 | alpha   |
|       3 | a       |      30 | a           |      10 |      99 | alpha   |
|       4 | b       |     100 | b           |     100 |     999 | bravo   |
|       5 | b       |     200 | b           |     100 |     999 | bravo   |
|    NULL | NULL    |    NULL | d           |   10000 |   99999 | delta   |
+---------+---------+---------+-------------+---------+---------+---------+
```

Some DBMS supports "outer join", which is basically a union of the left and right join results. More about union is introduced later.

The statement `<filter-condition>` applies filtering to the results. Commonly seen filter criteria `<filter-condition>` are summarized as follows.

- `<column-name> = <value>`, where `=` can be replaced by `<`, `<=`, `>`, `>=` and `<>`.
- `<column-name> IN (<value>, <value>, ...)`
- `<column-name> BETWEEN <value> AND <value>`
- `<column-name> LIKE <wildcards>`, which compares the column value (usually a string) with a given pattern.

- A combination of the above, with `AND` and `OR` joining everything together.

A bit more about wildcard query is introduced as follows. A wildcard character is a "placeholder" that represents a group of character(s). Most commonly used wildcard characters in the SQL context include

- `_`: any single character.
- `%`: any string of characters (including empty string).
- `[<c1><c2> ...]`: any single character given in the bracket.
- `^[<c1><c2>...]`: any single character not given in the bracket.
- `[<c1>-<c2>]`: any single character given within the range in the bracket.

Wildcard query can be applied to both CHAR and DATE/TIME types, as they can all be characterized as strings.

The `GROUP BY` groups the rows with the same value of the specified column into "summary rows". In each summary row, aggregated information is collected. To further explain this, consider the following example. The table used in this example is given below. Consider the same table `test` as follows.

```
> SELECT * FROM test;
+---------+---------+---------+
| test_id | value_1 | value_2 |
+---------+---------+---------+
|       1 | a       |      10 |
|       2 | a       |      20 |
|       3 | a       |      30 |
|       4 | b       |     100 |
|       5 | b       |     200 |
|       6 | c       |    1000 |
|       7 | c       |    2000 |
+---------+---------+---------+
```

Consider running the following command.

```
> SELECT * FROM test GROUP BY value_1;
+---------+---------+---------+
| test_id | value_1 | value_2 |
+---------+---------+---------+
|       1 | a       |      10 |
|       4 | b       |     100 |
|       6 | c       |    1000 |
+---------+---------+---------+
```

From the result, it can be seen that summary rows have been created using `GROUP BY`. Distinct values from `value_1` have been selected to form each summary row. From the display, only the first appearance of each associated summary row is returned, and a lot of information seems missing. However, it

is worth mentioning that although not displayed, the aggregated information is included.

To verify the presence of the aggregated information, consider running the following command.

```
> SELECT COUNT(*) FROM test GROUP BY value_1;
+----------+
| COUNT(*) |
+----------+
|        3 |
|        2 |
|        2 |
+----------+
```

From the result, we can see that the counted number of each summary row is returned.

Similarly, the following SQL returns other aggregation information associated with each summary row.

```
> SELECT value_1, COUNT(*), SUM(value_2) FROM test GROUP BY value_1;
+---------+----------+--------------+
| value_1 | COUNT(*) | SUM(value_2) |
+---------+----------+--------------+
| a       |        3 |           60 |
| b       |        2 |          300 |
| c       |        2 |         3000 |
+---------+----------+--------------+
```

Finally, `ORDER BY` and `LIMIT` controls the sequence and maximum number of returned rows, respectively.

The returns of multiple queries might be able to `UNION` together, if they are union-compatible. To union the results, use

```
SELECT <...>
UNION
SELECT <...>
UNION
SELECT <...>
...
SELECT <...>;
```

where inside `<...>` are the original query statements. Notice that for the queries to be union-compatible, they must have the same number of columns with identical data type for the associated column. The names of the column in the returns, if different, follow the first query result. Use alias `AS` to change the names if needed. Duplicated rows in the union will be excluded. If duplications need to be included in the result, certain DBMS provides the `UNION ALL` option.

SQL uses nest queries to add more flexibility. Nest queries plays as the intermediate steps to provide a temporary searching result, from which another

query can be executed. Wherever a table name appears in the query, it can be replaced by a `SELECT` statement nested in a bracket "()". A demonstrative example is given below. Consider the same tables `test` and `test_join` as follows.

```
> SELECT * FROM test;
+---------+---------+---------+
| test_id | value_1 | value_2 |
+---------+---------+---------+
|       1 | a       |      10 |
|       2 | a       |      20 |
|       3 | a       |      30 |
|       4 | b       |     100 |
|       5 | b       |     200 |
|       6 | c       |    1000 |
|       7 | c       |    2000 |
+---------+---------+---------+

> SELECT * FROM test_join;
+--------------+---------+---------+---------+
| test_join_id | value_1 | value_2 | value_3 |
+--------------+---------+---------+---------+
| a            |      10 |      99 | alpha   |
| b            |     100 |     999 | bravo   |
| d            |   10000 |   99999 | delta   |
+--------------+---------+---------+---------+
```

A inner join is provided to the above tables. However, for each `value_1` in the first table, the sum of the associated `value_2`, instead of each individual row, is used. This can be achieved using

```
> SELECT temp.value_1 AS type,
    ->         temp.sum_value_2 AS total_value,
    ->         test_join.value_1 AS minval,
    ->         test_join.value_2 AS maxval,
    ->         test_join.value_3 AS abbrev
    -> FROM (SELECT value_1,
    ->             SUM(value_2) AS sum_value_2
    ->      FROM test GROUP BY value_1) AS temp
    -> JOIN test_join
    -> ON (temp.value_1 = test_join.test_join_id);
+------+-------------+--------+--------+--------+
| type | total_value | minval | maxval | abbrev |
+------+-------------+--------+--------+--------+
| a    |          60 |     10 |     99 | alpha  |
| b    |         300 |    100 |    999 | bravo  |
+------+-------------+--------+--------+--------+
```

where notice that alias are quite some times to clarify the logics.

Nest queries can be popular in table joins as well as filter criteria, where the boundary of a variable can be obtained from a nest query.

Trigger

A trigger defines a set of operations to be carried out automatically when something happens to specified tables. For example, in any case a new row is added to a table, a trigger can automatically insert an associated record into a second table.

There are mainly 3 types of triggers: DML trigger (triggered by `INSERT`, `UPDATE`, `DELETE`, etc.), DDL trigger (triggered by `CREATE`, `ALTER`, `DROP`, `GRANT`, `DENY`, `REVOKE`, etc.), and CLR trigger (triggered by LOGON event).

A quick DML trigger can be defined as follows.

```
CREATE TRIGGER <trigger-name>
    [BEFORE | AFTER] [INSERT | UPDATE | DELETE] ON <table-name>
    FOR EACH ROW <operation>;
```

where `BRFORE` is often used to validate and modify data to be added to `<table-name>`, and `AFTER` is often used to trigger other changes consequent to this change.

In case multiple operations need to be defined, consider using

```
DELIMITER $$
CREATE TRIGGER <trigger-name>
    [BEFORE | AFTER] [INSERT | UPDATE | DELETE] ON <table-name>
    FOR EACH ROW BEGIN
        <operation>;
        ...
        <operation>;
    END$$
DELIMITER ;
```

where `DELIMITER $$` and `DELIMITER ;` is used to temporarily change the delimiter for the `BEGIN...END` statement. It is possible to build slightly complicated logics in the operations, for example to build conditional statements.

Use `NEW` in the operation(s) to represent the rows that is added/updated/deleted from the `table-name`.

Use the following to drop a trigger.

```
DROP TRIGGER <trigger-name>;
```

12.2.4 MariaDB

MariaDB and MySQL are two widely used relational DBMS. MariaDB is initially a fork of MySQL, and in this since they share many similarities. While MySQL moves towards a dual license approach (free community license and paid enterprise license with proprietary code), MariaDB is designed to be fully open-source and plays as a replacement of MySQL.

In general, MariaDB supports a larger varieties of data engines and new features, and it is claimed to be faster, more powerful and advanced. However,

it lacks some of the enterprise features provided by MySQL. The users can gain these features by using open-source plugins.

In this section, MariaDB is used for demonstration. Notice that both of them shall serve fine for this purpose. Most of the operations introduced in this section, if not all, shall work indifferently on both DBMS.

MariaDB Installation

Install MariaDB on a Linux system as follows.

```
$ sudo apt update
$ sudo apt install mariadb-server
```

Notice that MySQL can be installed instead by replacing `mariadb-server` with `mysql-server`. Similarly, replace `mariadb` with `mysql` in the rest of this section, wherever applicable. Both DBMS shall work similarly within the scope of this introduction section.

MariaDB server can be controlled using `systemctl`, which is introduced in Section 10.1. For example, to start MariaDB, use

```
$ sudo systemctl start mariadb.service
```

and to check its status, use

```
$ sudo systemctl status mariadb
```

After installation of and starting MariaDB, use

```
$ sudo mysql_secure_installation
```

to run a quick security-related configuration such as creating password for the root user, and deleting test database.

Login to MariaDB console using

```
$ sudo mariadb
```

Notice that when *sudo* privilege is used, this should brings the user to the root account of the DBMS. Otherwise, the DBMS may ask the user to provide further login credentials, or deny the login attempt. To login as a particular user, use the following command

```
$ mariadb -u <user-name> -p
Enter Password:
```

Notice that in some machines, remote login to the database using the root account is forbidden.

After login to MariaDB console, a prompt that looks like the following would show up.

```
MariaDB [(none)]>
```

from where an admin account can be created as follows.

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO '<user-name>'@'
    localhost' IDENTIFIED BY '<user-password>' WITH GRANT OPTION;
MariaDB [(none)]> FLUSH PRIVILEGES;
```

By creating an admin account, the use of root account can be reduced, thus reducing the operation risks. Notice that when using remote connection from another machine to the database, this command is also required to grant access to the remote user with specified IP address. Wildcard expression can be used for the IP address, if necessary.

To check existing users and their IP addresses to whom access has been granted, use

```
SELECT host, user FROM mysql.user;
```

Finally, use

```
MariaDB [(none)]> exit
```

to quite MariaDB console.

A Database Example: Create Database Schema

In the rest of this section, a database is created from scratch as a demonstration of using SQL to interact with MariaDB. The database is used in the smart home project to trace the resources obtained and consumed by the user. The resources in this context may refer to groceries bought from the supermarket, books purchased online, subscriptions of magazines and services, etc.

For simplicity, the prompt is ignored in the rest of this section.

Check the existing databases using

```
SHOW DATABASES;
```

A database named `smart_home` is created as follows.

```
CREATE DATABASE smart_home;
```

Select the database as follows.

```
USE smart_home;
```

With the above command, `smart_home` is selected as the current database.

Based on the database schema design, a few tables need to be created. We shall start with creating `asset`, `accessory`, `consumable` and `subscription` tables as follows.

The `asset` table is used to trace assets in the home. They are often expensive and comes with a serial number or a warranty number, and shall persist for a long time (a few years, at minimum). Examples of assets include beds, televisions, computers, printers, game consoles. The `accessory` table is used to trace relatively cheaper accessories than assets. Though they are designed to last long, they may not have an serial number. Examples of accessories include books, charging cables, coffee cups. The `consumable` table is used to trace items that is meant to be used up or expire. Examples of consumable items include food, shampoo, A4 printing paper. And finally the `subscription` table is used to trace subscriptions of services. Examples of these services include software license (either permanent license or annual

subscription license), magazine subscriptions, membership subscriptions, and digital procurement of a movie.

The serial number or warranty number for assets are used as the primary key of `asset` table. For the other three tables, surrogate keys are used. Each table has a column `product_type_id` that specifies the type of the item, such as "television", "cooker", "fruit", "software". The types in these tables are given by integer indices. A separate `product_type` relates the indices with their associated meanings. The same applies to `product_brand_id` and `payment_method_id`.

Create `asset` table as follows.

```
CREATE TABLE asset (
    PRIMARY KEY (serial_num),
    serial_num              VARCHAR(50)   NOT NULL,
    product_type_id         INT(5),
    product_brand_id        INT(5),
    product_name            VARCHAR(50)   NOT NULL,
    receipt_num             VARCHAR(50),
    procured_date           DATE          NOT NULL DEFAULT (
        CURRENT_DATE),
    procured_price          DECIMAL(10,2),
    payment_method_id       INT(5),
    warranty_date_1         DATE          NOT NULL DEFAULT (
        CURRENT_DATE),
    warranty_date_2         DATE          NOT NULL DEFAULT (
        CURRENT_DATE),
    expire_date             DATE          NOT NULL DEFAULT
        '9999-12-31',
                            CONSTRAINT warranty_after_procured
                            CHECK(warranty_date_1 >= procured_date AND
                                 warranty_date_2 >= warranty_date_1),
                            CONSTRAINT expire_after_procured
                            CHECK(expire_date >= procured_date)
);
```

where

- `serial_num`: the serial number, MAC number or registration ID that can be used to uniquely identify the asset.
- `product_type_id`: type index.
- `product_brand_id`: brand index.
- `product_name`: full name of the product that can uniquely specify the asset on the market.
- `receipt_num`: receipt and/or warranty number.
- `procured_date`: date of procurement.

- `procured_price`: price of the product as procured.
- `payment_method_id`: payment method.
- `warranty_date_1`: warranty expiration date (free replace or repair); leave it as the procured date if no such warranty is issued.
- `warranty_date_2`: second warranty expiration date (partially covered repair); leave it as the procured date if no such warranty is issued.
- `expire_date`: the date when the asset expires or needs to be returned. For example, in Singapore a car "expires" in 10 years from the day of procurement.

Notice that constraints and default values have been added to the table creation. An SQL script is used contain the code, and

```
$ mariadb -u <user-name> -p < <script-name>
```

is used to execute the script, which is more convinient than typing all the lines in the MariaDB console.

Similarly, create the rest 3 tables for the resources as follows.

```
CREATE TABLE accessory (
    PRIMARY KEY (item_id),
    item_id                 INT(5)         AUTO_INCREMENT,
    product_type_id         INT(5),
    product_brand_id        INT(5),
    product_name            VARCHAR(50)    NOT NULL,
    receipt_num             VARCHAR(50),
    procured_date           DATE           NOT NULL DEFAULT (
        CURRENT_DATE),
    procured_number         DECIMAL(10,2)  NOT NULL DEFAULT 1.00,
    procured_unit_price     DECIMAL(10,2),
    procured_price          DECIMAL(10,2),
    payment_method_id       INT(5),
    expire_date             DATE           NOT NULL DEFAULT
        '9999-12-31',
                            CONSTRAINT expire_after_procured
                            CHECK(expire_date >= procured_date)
);

CREATE TABLE consumable (
    PRIMARY KEY (item_id),
    item_id                 INT(5)         AUTO_INCREMENT,
    product_type_id         INT(5),
    product_brand_id        INT(5),
    product_name            VARCHAR(50)    NOT NULL,
    receipt_num             VARCHAR(50),
    procured_date           DATE           NOT NULL DEFAULT (
        CURRENT_DATE),
```

```
    procured_number          DECIMAL(10,2) NOT NULL DEFAULT 1.00,
    procured_unit_price      DECIMAL(10,2),
    procured_price           DECIMAL(10,2),
    payment_method_id        INT(5),
    expire_date              DATE          NOT NULL DEFAULT (
        CURRENT_DATE),
                             CONSTRAINT expire_after_procured
                             CHECK(expire_date >= procured_date)
);

CREATE TABLE subscription (
    PRIMARY KEY (item_id),
    item_id                  INT(5)        AUTO_INCREMENT,
    product_type_id          INT(5),
    product_brand_id         INT(5),
    product_name             VARCHAR(50)   NOT NULL,
    receipt_num              VARCHAR(50),
    procured_date            DATE          NOT NULL DEFAULT (
        CURRENT_DATE),
    procured_price           DECIMAL(10,2),
    payment_method_id        INT(5),
    expire_date              DATE          NOT NULL DEFAULT (
        CURRENT_DATE),
                             CONSTRAINT expire_after_procured
                             CHECK(expire_date >= procured_date)
);
```

Create the tables for users, product types, product brands and payment methods as follows.

```
CREATE TABLE user (
    PRIMARY KEY (user_id),
    user_id                  INT(5),
    first_name               VARCHAR(50)   NOT NULL,
    last_name                VARCHAR(50)   NOT NULL,
    email                    VARCHAR(50)   NOT NULL UNIQUE
);

CREATE TABLE product_type (
    PRIMARY KEY (product_type_id),
    product_type_id          INT(5)        AUTO_INCREMENT,
    product_type_name        VARCHAR(50)   NOT NULL UNIQUE,
    product_type_name_sub    VARCHAR(50)   NOT NULL DEFAULT ('na')
);

CREATE TABLE product_brand (
    PRIMARY KEY (product_brand_id),
    product_brand_id         INT(5)        AUTO_INCREMENT,
    product_brand_name       VARCHAR(50)   NOT NULL UNIQUE
);
```

```
CREATE TABLE payment_method (
    PRIMARY KEY (payment_method_id),
    payment_method_id        INT(50)        AUTO_INCREMENT,
    user_id                  INT(5),
    payment_method_name      VARCHAR(50)    NOT NULL
);
```

Finally, create foreign keys as follows.

```
ALTER TABLE payment_method
ADD FOREIGN KEY (user_id)
REFERENCES user(user_id);
ALTER TABLE asset
ADD FOREIGN KEY (product_type_id)
REFERENCES product_type(product_type_id);
ALTER TABLE asset
ADD FOREIGN KEY (product_brand_id)
REFERENCES product_brand(product_brand_id);
ALTER TABLE asset
ADD FOREIGN KEY (payment_method_id)
REFERENCES payment_method(payment_method_id);
ALTER TABLE accessory
ADD FOREIGN KEY (product_type_id)
REFERENCES product_type(product_type_id);
ALTER TABLE accessory
ADD FOREIGN KEY (product_brand_id)
REFERENCES product_brand(product_brand_id);
ALTER TABLE accessory
ADD FOREIGN KEY (payment_method_id)
REFERENCES payment_method(payment_method_id);
ALTER TABLE consumable
ADD FOREIGN KEY (product_type_id)
REFERENCES product_type(product_type_id);
ALTER TABLE consumable
ADD FOREIGN KEY (product_brand_id)
REFERENCES product_brand(product_brand_id);
ALTER TABLE consumable
ADD FOREIGN KEY (payment_method_id)
REFERENCES payment_method(payment_method_id);
ALTER TABLE subscription
ADD FOREIGN KEY (product_type_id)
REFERENCES product_type(product_type_id);
ALTER TABLE subscription
ADD FOREIGN KEY (product_brand_id)
REFERENCES product_brand(product_brand_id);
ALTER TABLE subscription
ADD FOREIGN KEY (payment_method_id)
REFERENCES payment_method(payment_method_id);
```

A Database Example: Insert Data

xxx

12.2.5 RDS Access Using Python

Python provides variety of libraries to access RDS, many of which use embedded SQL codes to interact with the DBMS. Depending on the DBMS, different libraries and commands can be used, some of which more general and the other more specific to a particular DBMS.

In this section, both `pandas` and `mariadb` libraries are introduced. The `pandas` library provides data manipulation and analysis tools, and it provides `pandas.io.sql` that allows connecting to a DBMS and embedding SQL commands into the python code. The `mariadb` library, on the other hand, is dedicated for MariaDB connection. Like `pandas.io.sql`, it also allows embedding SQL commands to interface the DMBS.

As pre-requisitions, make sure that the following has been done.

- The DBMS has been configured to allow remote access.
- Make sure that an account has been registered in DBMS that has the privilege of operation from a remote machine.
- Make sure that the firewall configuration is correct.

For DBMS configuration, in the case of MariaDB, use the following code in the shell to check the location of the configuration files.

```
$ mysqld --help --verbose
```

Typical locations of the configuration files are `/etc/my.cnf` and `/etc/mysql/my.cnf`. In the configuration file, use the following to disable binding address.

```
[mysqld]
skip-networking=0
skip-bind-address
```

For account setup, in the DBMS console, use something like

```
> GRANT ALL PRIVILEGES ON *.* TO '<user-name>'@'<ip-address>'
    IDENTIFIED BY '<user-password>' WITH GRANT OPTION;
```

where `'<ip-address>'` is the remote machine that runs the Python program. If the Python codes are running locally, simply use `'localhost'`.

It might be necessary to install MariaDB database development files in the DMBS host machine for the Python libraries to be introduced to function properly. Install the development as follows.

```
$ sudo apt install libmariadb-dev
```

PANDAS Library

Python library `pandas` is one of the essential libraries for data analysis. It provides flexible interfaces and tools for data reading and processing and works very well with different data formats and engines including CSV, EXCEL and DBMS. This section focuses mainly on the interaction of `pandas` with DBMS. Therefore, the detailed use of `pandas` for data analysis, etc., are not covered in this section.

A class `pandas.DataFrame` is defined in `pandas` as the backbone to store and process data. The data attribute of `pandas.DataFrame` is a `numpy` array. Many functions are provided to read different data formats into `pandas` data frame, which makes reading data easy and convenient. An example of reading a CSV file is given below.

```
import pandas as pd
df = pd.read_csv(<file-name>)
print(df)
print(df.head(<number>))
print(df.tail(<number>))
print(df.info())
```

where `head()`, `tail()` gives the first and last rows of the data frame, and `info()` checks the data frame basic information including shape and data types of the columns. Check `df.columns` for all the columns of the data frame. Details specific to a column can be accessed via `df.[<column-name>]`. Many functions are provided to further abstract the details, such as grouping and counting. Use `df.loc(<row-index-list>, <column-name-list>)` to check the content of specified rows and columns.

With `pandas` and other relavent libraries, Python can connect to a database and execute a query. An example of using `pandas` to connect to an Microsoft SQL server and implement a query is given below. Notice that different DBMS may require different database connectivity driver standards, and there are mainly two of them, namely open database connectivity (ODBC) and Java database connectivity (JDBC). Microsoft adopts ODBC, and a separate package is required in the Python program to connect to the Microsoft SQL server.

```
import pyodbc
import pandas.io.sql as psql

server = "<server-url>,<port>"
database = "<database>"
uid = "<uid>"
pwd = "<pwd>"
driver = "<driver>" # such as "{ODBC Driver 17 for SQL Server}"

# connect to database
conn = pyodbc.connect(
       server = server,
```

```
        database = database,
        uid = uid,
        pwd = pwd,
        driver = driver
)

# get cursor
cursor = conn.cursor()

# execute sql command
query = """<query>"""
runs = psql.read_sql_query(query, conn)
```

The above codes returns a data frame corresponding to the result set of the query string, which is saved in `runs`.

MARIADB Library

Use the following code to test connectivity from Python to the database.

```
import mariadb
import sys

user = "<user>"
password = "<password>"
host = "<server-url>"
port = "<port>" # MariaDB default: 3306
database = "<database>"

# connect to database
try:
    conn = mariadb.connect(
        user = user,
        password = password,
        host = host,
        port = port,
        database = database
    )
except mariadb.Error as e:
    print(f"Error connecting to MariaDB Platform: {e}")
    sys.exit(1)

# get cursor
cur = conn.cursor()

# execute sql command
cur.execute("<sql-command>")
```

Notice that for query, the result is stored in the cursor object. Use a for loop to view the results.

12.3 Non-relational Database

...

12.3.1 Brief Introduction to Non-relational Database

...

12.3.2 MangoDB

...

13

Virtualization and Containerization

CONTENTS

Virtualization and containerization are widely appreciated techniques for distributing multiple instances of applications on single or multiple physical servers. The primary objective of these technologies is to enhance resource utilization and efficiency, while maintaining isolation between applications. The key challenge for virtualization and containerization is to deploy and manage each instance efficiently and consistently across various environments.

13.1 Introduction

One of the major differences between a server cluster and a PC is that the former is usually shared among multiple users or applications at the same time. Though working on the same machine, a user would usually want a private working environment not interrupted by other users. In other words, a user would want to "virtually" work on an independent machine with his own CPU, RAM, I/O, OS, drivers and hard disk storage, despite that the actual hardware is shared with others. This can be achieved 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 generally consumes a considerably large amount of

time. This is because different VMs on the same server are separated at the OS level, with each VM requiring its own OS installation. Consider a scenario where there are hundreds of small applications (microservices), each requiring a similar but separate environment. Launching VMs for all of them is resource-intensive, particularly when these applications could have shared the same OS kernel and operated within their own isolated workspace.

A more efficient approach would be to deploy a single VM and place each application in a "container" with its own customized drivers and configurations. A container is similar to a VM in the sense that it provides a degree of isolation from others, but it is typically "lighter" than a VM because it doesn't need to virtualize or duplicate the whole OS as VMs do. This makes containers cheaper to launch and manage.

The technique used to deploy and manage containers is known as "containerization". Since a container contains all the configuration and requirement information of an application, running a container on different platforms would consistently generate the same expected result. This has made the sharing and rapid deployment of containers remarkably easy and convenient. The similarities and differences of personal PCs, VMs, and container applications are summarized in Fig. 13.1.

As an analogy, think of running an APP as asking a kitchen to prepare a dish. The hardware is corresponding with the physical resources in kitchen, such as the cooktop, gas, etc. The OS is corresponding with the person who manages the kitchen, say a cook. The OS needs associated drivers and libraries to run the APP correctly. The drivers and libraries correspond with the skill sets or specific cookers for the dish. Finally, the APP is corresponding with the expected dish.

In the most simple configuration, a dedicated machine is used to run an APP. This is like constructing a dedicated kitchen and hiring a dedicated cook for each dish. The cook is trained to master all necessary skills required for that specific dish. This is shown in Fig. 13.2.

In a VM implementation, a larger and more capable kitchen is setup in advance as shown in Fig. 13.3. For each dish, a cook is hired. Each cook is trained with the skills necessary for his assigned dish. All cooks share the same kitchen. This implementation is more efficient than Fig. 13.2, as there is no need to scale up the kitchen for a new dish. By sharing the resources among the cooks, the kitchen can be utilized more effectively.

While Fig. 13.3 might be a popular practice in many restaurants, it is still too costly to hire a new cook for each dish. In a containerization implementation, a cook usually handles a category of dishes, as shown in Fig. 13.4. Of course, each dish will stay in its own fry-pan in an isolated way. For each dish, its receipt is provided from where you can find all information required to prepare the dish consistently. As long as the cook is good at multi-tasking, Fig. 13.4 is a more efficient implementation than Fig. 13.2.

Just like a receipt guaranteeing the consistency of dishes, in containerization, an "image" guarantees the consistent performance of container instances.

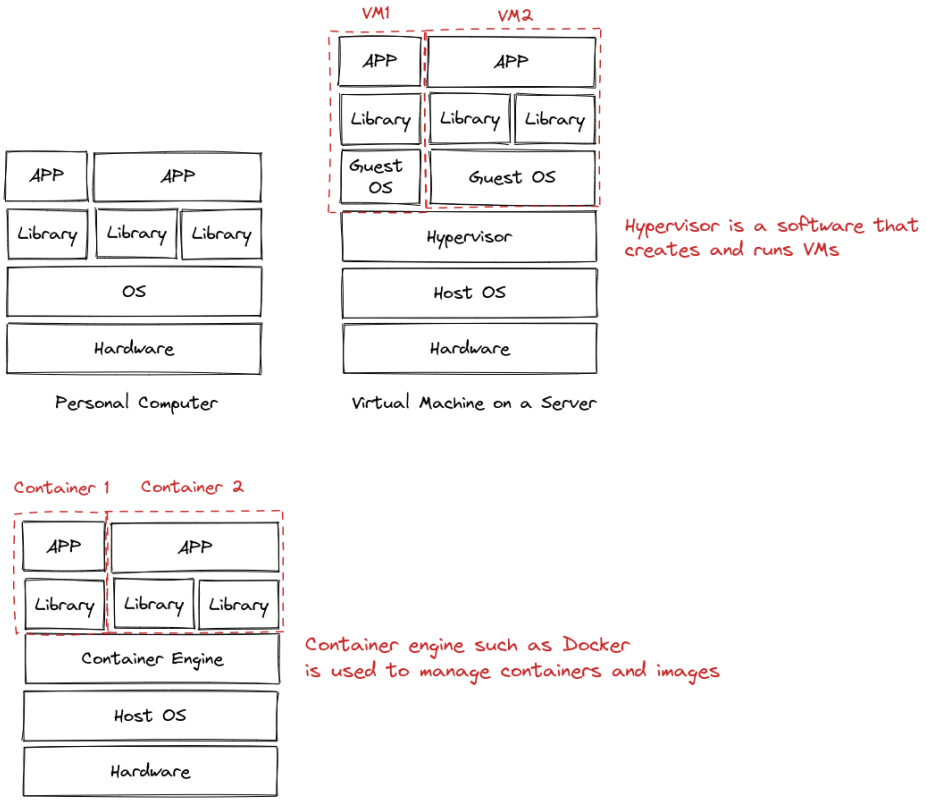

FIGURE 13.1
System architectures of PC, VM and container.

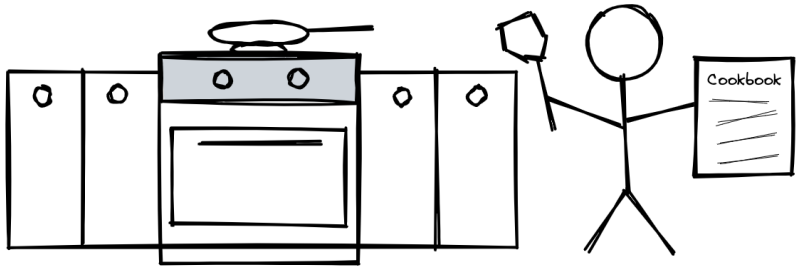

FIGURE 13.2
PC implementation: a cook in a kitchen.

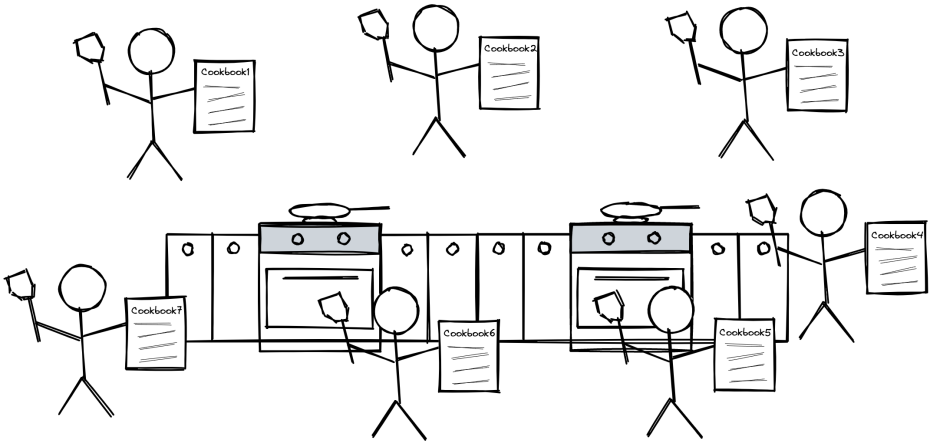

FIGURE 13.3
VM implementation: many cooks in a kitchen, each with a different cookbook.

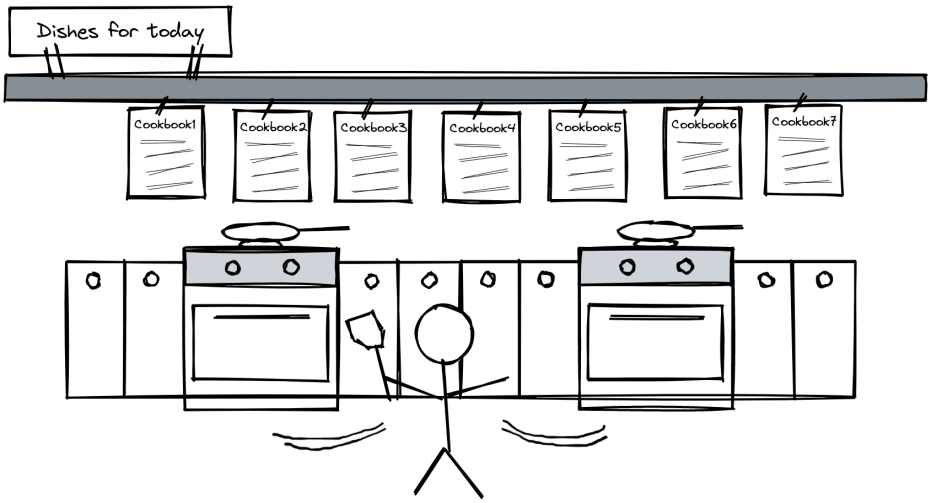

FIGURE 13.4
Container implementation: one in a kitchen, handling multiple dishes, each has a cookbook and stays in its own pan.

An image is basically a collection of prerequisites and configurations to run a container quickly, efficiently and consistently. Images can be shared among machines to replicate the containers even if the machines adopt different underlying infrastructure such as OS.

13.2 Virtualization and Containerization

Virtualization and containerization technologies have been studied for decades.

Before moving forward to the introduction of containerization approaches, it is important to briefly explain the following frequently used concepts: container runtime, container engine, and container orchestration.

Container runtime refers to the backend software that actually runs the containers. Examples of widely used container runtimes are "containerd", "runc" and "cri-o". Container engine is the interface for a user or software to manage images and containers. Examples of container engines include "docker" and "podman". Finally, container orchestration is the software that smartly deploy, monitor, restart, and terminate the containers on servers. It is often able to balance API calls, automatically scale up and down the number of running containers, and restart containers upon failure. One of the most widely used container orchestrations is "kubernetes".

Container runtime is definitely vital in every containerization implementation. Container engines and container orchestrations must have container runtime built-in. However, the users rarely directly talk to the container runtime.

Container engine is the interface and management tool of the container runtime, from where the developer can monitor and control the containers manually. Container engines are widely used especially in small projects or in the development environment.

For massive deployment of containers in enterprise-level applications, container orchestration can become handy. In such a case, the container orchestration calls the container engine via API (nowadays a container orchestration may also talk to container runtime directly), instructing it on how the containers should be deployed.

13.3 Docker

This section introduces docker Notice that although docker is famous for docker container engine, docker as a company or community provides many

revolutionary container related tools and services that go far beyond a container engine, many of which even used in its opponent container engines.

This section focuses mostly on the introduction of docker engine.

13.3.1 Docker Engine VS Alternatives

Docker engine is the most popular container engines available on the market as of 2023, and it is free of charge for open-source, personal and small business usage. More details of docker can be found at *https://docs.docker.com/*.

But does that mean docker engine is the absolutely best and perfect container engine solution?

Docker has surely revolutionized how we use containerization technology in software development and deployment, and it has been one of the most popular and beloved container engine solutions. However, it is worth mentioning that docker engine is not the only available container engine. For example, as explained earlier podman is an alternative to docker. It supports the same interface (as if podman is an alias) as docker, and claims to have better performance and security. As a matter of fact, RHEL already started the transitionary from docker to podman from RHEL 8. Nowadays, installing docker on the latest versions of RHEL is possible but tedious. On the other hand, install podman (it might be built-in to the OS installation) on RHEL can be done by simply using

```
$ sudo yum install podman
```

Kubernetes, a famous container ochestration, is also dropping docker support, as they say "docker support in the kubelet is now deprecated and will be removed in a future release". Some may even argue that "containers are alive, but the role that docker plays is shrinking". Many open-source initiatives such as podman are gaining round.

Docker has some disadvantages indeed. For one thing, docker uses docker server (docker daemon), a single piece of software running in the backend of the system, to support all the services. This creates a single-point-failure of the system. Docker requires root privileges, and it starts a container on behalf of the root user. This means that the program running inside the container, or the users in the docker group can potentially bypass the OS access control and gain root access, which introduces security risk. These shortages are to some extent addressed by other container engines such as podman which is daemon-less and does not necessarily need to run on root user's behalf.

This is not to say that Docker is falling behind as a whole. Some key techniques that Docker introduced are widely used in all different types and brands of container runtimes and engines. It is just that people do not like some of the features and schematics implementation of Docker engine, and new tools are being developed to fix these problems, the latter of which starting drawing more and more attention. Docker still enjoys widespread usage and support due to its massive community, wealth of online resources, and

extensive compatibility with numerous tools and platforms. Nevertheless, for demonstration purpose, for the remaining sections docker is used throughout this notebook. Since podman provides the same interface, it is probable that podman can be used likewise to replicate all the results.

As of 2023, Docker is still the dominating container market engine, with a market share of over 80%.

13.3.2 Docker Installation

To install Docker on a Linux machine, go to *https://www.docker.com/* to look for the instruction. The installation steps differ depending on the host machine. 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
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
    dearmor -o /etc/apt/keyrings/docker.gpg
$ 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. In the rest of the section, `sudo` is neglected for docker commands.

```
$ sudo usermod <user name> -aG docker
```

Docker installs at least two piece of software on the machine, namely Docker CLI and docker server (also known as docker daemon). The CLI is the interface to the user, and the server is the actual tool that manages images and containers. Docker runs natively on Linux OS. If docker is installed on non-Linux system such as Windows or macOS, "docker desktop" is used, which includes a Linux VM to host the docker daemon and run Linux-based containers.

13.3.3 Docker Container Management

To run a container from an image, simply use

```
$ docker run <image>
```

Docker will search the local and remote repositories for the image, download the image if necessary, and start a container from that image. By default, after successful execution, the container will enter "Exited" status. Use

```
$ docker container ls
```

to check the list of running containers, and

```
$ docker container ls -a
```

the list of all containers, running or exited. Alternatively, `docker ps`, `docker ps -a` can also be used to list down containers just like `docker container ls`, `docker container ls -a`.

For example, consider running a container of *alpine* as follows. A screen shot is given in Fig. 13.5.

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

where `-i` stands for "interactive", which keeps the container's standard input (i.e., the console in this example) open so that the user can actively interact with the container. Option `-t` allocates a pseudo-TTY to the container. TTY stands for "TeleTYpewriter", which enforces the I/O of the container following the typical terminal format and allows the user to interact with the container like a traditional terminal, hence making the interactive interface a bit more user-friendly. Finally, `--name` assigns a name to the container. Without an assigned name, docker will assign a random name to the container.

It can be seen from Fig. 13.5 that once the container is started, the user can interact with the container via shell, and perform actions such as listing items in the current directory in the container. While keeping the container running, open another terminal and use `docker container ls`. The container `test-alpine` shall appear in the list, as shown in Fig. 13.6.

After exiting from Fig. 13.5 (by using `exit` in *alpine*), the container will transfer its status from "running" to "exited", as shown in Fig. 13.7.

It is also possible to launch a container and let it run in the backend using `-d` flag which stands for "detached mode". An example is given below.

```
sunlu@sunlu-laptop-ubuntu:~$ docker run -it --name test-alpine alpine
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # 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]
OK: 17030 distinct packages available
/ # apk upgrade
(1/6) Upgrading alpine-baselayout-data (3.2.0-r20 -> 3.2.0-r22)
(2/6) Upgrading busybox (1.35.0-r13 -> 1.35.0-r14)
Executing busybox-1.35.0-r14.post-upgrade
(3/6) Upgrading alpine-baselayout (3.2.0-r20 -> 3.2.0-r22)
Executing alpine-baselayout-3.2.0-r22.pre-upgrade
Executing alpine-baselayout-3.2.0-r22.post-upgrade
(4/6) Upgrading libcrypto1.1 (1.1.1o-r0 -> 1.1.1q-r0)
(5/6) Upgrading libssl1.1 (1.1.1o-r0 -> 1.1.1q-r0)
(6/6) Upgrading ssl_client (1.35.0-r13 -> 1.35.0-r14)
Executing busybox-1.35.0-r14.trigger
OK: 6 MiB in 14 packages
/ #
```

FIGURE 13.5
An example of running *apline* container, with interactive TTY and name *test-apline*.

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

FIGURE 13.6
List the running container *test-apline*.

```
sunlu@sunlu-laptop-ubuntu:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
sunlu@sunlu-laptop-ubuntu:~$ docker container ls -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS                      PORTS     NAMES
03b1039d4f4d   alpine    "/bin/sh"   33 minutes ago   Exited (0) 7 seconds ago              test-alpine
```

FIGURE 13.7
List the exited container *test-apline*.

TABLE 13.1
Commonly used docker commands to launch a container.

Command	Flag	Description
`docker run`	—	Launch a container of the image followed 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.
`docker run`	`-d`	Launch the container in the backend and keep it running.
`docker run`	`--rm`	Automatically remove the container when exiting. The removed 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. The flag usually comes with the flags `-i` or `-d`, to form `-it` or `-dt`.
`docker run`	`--restart`	Enforce restart of the container upon exiting. This is usually used on containers running in the backend. Commonly used restart configurations include `--restart no` (do not restart), `--restart on-failure[:<max retries>]` (restart if exits with an error flag), `--restart always` (always restart when exists).
`docker run`	`--name`	Assign a name to the container.

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

By changing `-i` to `-d`, the container runs in the backend silently. The status of the container, after executing the above command, will stay running and can be displayed by `docker container ls`.

Commonly used commands regarding launching a container are given in Tables 13.1, and 13.2.

To re-start an existing but exited container, use

```
$ docker start <container>
```

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

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

```
$ docker exec <container> <command>
```

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

TABLE 13.2
Commonly used docker commands to display local images and containers.

Command	Flag	Description
`docker image ls`	—	List local images.
`docker container ls`	—	List running containers.
`docker container ls`	`-a`	List all containers.

```
$ docker exec -it <container> <shell name>
```

Notice that the shell used by the application running inside the container may differ from the one used in the host machine. In the case of an *alpine* image based container, `ash` is the default shell. For a *ubuntu* image based container, `bash` is often used. To exit from the TTY shell while keep the container running in the backend, use shortcut key `Ctrl+p+q`.

An alternative way to interact with containers running in the backend is to use

```
$ docker attach <container>
```

to attach local standard input, output, and error streams to a running container. Similar with the previously introduced `docker exec -it` command, `docker attach` also starts the shell of the application running in the container. Use `Ctrl-C` to quite the shell.

To check the processes that is running in the container, use

```
$ docker top <container>
```

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

```
$ docker cp <container>:<source> <destination>
```

and from host machine to container:

```
$ docker cp <source> <container>:<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, kill (force stop) or restart a container running in the backend, use

```
$ docker stop <container>
$ docker kill <container>
$ docker restart <container>
```

respectively. When a container is stopped, it enters exited status. To remove an exited container or all exited containers, use

```
$ docker container rm <container>
```

or

```
$ docker container prune
```

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

```
$ docker rename <container-old-name> <container-new-name>
```

To quickly check container status including resource consumption (CPU, memory usage, etc.), use

```
$ docker stats [<container>]
```

where the user can choose to list down all containers or a specified container. To show more detailed information of a container, including its status, gateway, IP address, etc., use

```
$ docker inspect <container>
```

Finally, to check the logs of a container (e.g., its standard output to the console), use

```
$ docker logs <container>
```

A container is usually generated from an image. It is also possible to do vise versa, i.e., packaging a container into an image. To create an image from a container, use

```
$ docker commit <container> <image>
```

where `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, and it is not the only way an image is created.

A container can be configured accessible from not only the host machine but also other computers in LAN or the Internet. In a typical web service application, the host machine and the containers are often designed following the architecture similar with Fig. 13.8. Notice that in practice, the load balancer and the containers may or may not run on the same physical machine.

The load balancer is a container orchestration that monitors the status of the containers, manages the data flow, and scales up and down the number of containers depending on the total load. In case where there are multiple physical servers, the load balancer is run on a "master" server, and the containers are distributed on multiple "slave" servers, each of which is called a "worker node", or "node" for simplicity. Container engines are installed on each and every node. The load balancer talks with the container engines on each node to deploy containers. The APP instances such as *apache* or *nginx* shall run in the containers distributively.

An example of setting up a web server in containers from scratch is given in this section. For simplicity, everything happens on a single physical server.

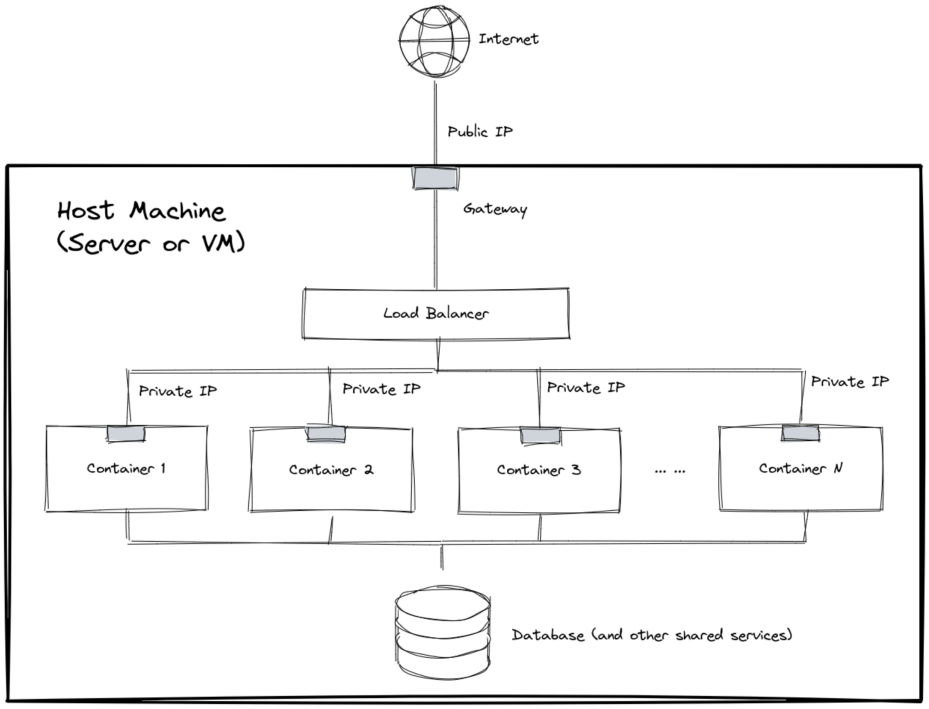

FIGURE 13.8
A simplified architecture where containers are used to host the web service.

Only one container is used and the load balancer and the shared services are not included in the example.

As a first step, create a container from the official *nginx* image as follows. Notice that it is also possible to create a container from *apline*, and install *nginx* on *apline*.

```
$ docker run -dt --name simple-web nginx
```

Next, create the configuration file for *nginx*, and also the *html* files to be used as the static web page. For convenience, the files are created and edited in the host machine, then copied to the container. The following *default.conf* and *index.html* have been created, respectively. The configuration file *default.conf* is given below.

```
server {
       listen 80 default_server;
       listen [::]:80 default_server;
       root /var/www/html/;
}
```

The *html* file *index.html* is given below.

```
<html>
       <body>
              <h1>Hello World!</h1>
       </body>
</html>
```

Use `docker copy` to copy the two files to the designed locations in the container as follows.

```
$ docker exec simple-web mkdir -p /var/www/html
$ docker cp default.conf simple-web:/etc/nginx/conf.d/default.conf
$ docker cp index.html simple-web:/var/www/html/index.html
```

where `mkdir -p` creates the directories along the given path, if not exist. Notice that the file name in the destination can be ignored if it is the same with the source, i.e., the copy commands can be replaced by

```
$ docker cp default.conf simple-web:/etc/nginx/conf.d/
$ docker cp index.html simple-web:/var/www/html/
```

Change the ownership of the *html* file as follows, so that the current user *nginx* is able to access that file.

```
$ docker exec simple-web chown -R nginx:nginx /var/www/html
```

Finally, reload and configuration file and restart the web server as follows.

```
$ docker exec simple-web nginx -s reload
```

To test the web server running inside the container, obtain the IP address of the container using

```
$ docker inspect simple-web | grep IPAddress
```

and open a browser to key in the obtained IP address. If everything is done correctly, the browser should try to access port 80 of the container, and the "Hello World!" web page shall show up.

For easy sharing and populating of the container, commit the container into a new image using `docker commit` as follows. The new image can be used to populate the web server, just like "web01" container given below.

```
$ docker commit simple-web simple-web-image
$ docker run -dt --name web01 -p 80:80 simple-web-image
```

where `-p <host machine port>:<container port>` is used to map ports. Notice that different from the previous container "*simple-web*", the new container "*web01*" IP address port 80 is mapped with the port 80 of the host machine. Therefore, the web page hosted in "*web01*" can be accessed not only by the host machine, but also by other machines in the same network with the host machine.

13.3.4 Docker Volume Configuration

As introduced earlier, `docker cp` can be used to transfer data into and out of a container. An alternative way of accessing docker container data from the host machine is to use docker volume. Docker volume is used to mount host machine hard drive to container storage. Details are introduced below.

To create a docker volume, use

```
$ docker volume create <volume>
```

To list down volumes and to inspect a volume, use

```
$ docker volume ls
$ docker volume inspect <volume>
```

respectively. Finally to remove a volume or all volumes, use

```
$ docker volume rm <volume>
$ docker volume prune
```

respectively.

When starting a container from an image, volumes can be mapped with the internal storage inside the container by using

```
$ docker run -v <volume>:<container-internal-path>[:ro] <image>
```

which should synchronize `<volumn name>` with `<container internal path>`. The optional `:ro` can be specified if it is a read-only volume. Instead of using a volume name, the path to a directory in the host machine can also be used, in which case the specified directories in the host machine and in the container should be synchronized.

Docker volume guarantees data persistence in containerized applications. When a docker container is removed, any data written to the container's

writable layer is lost. Docker volumes, however, are stored outside of the container's writable layer, allowing data to persist even after the container is removed. This persistence is particularly important for applications that require permanent storage, such as databases.

Moreover, Docker volumes can be shared among multiple containers, which facilitates data exchange and allows containers to work on the same dataset. Therefore, Docker volumes are not just a tool for data persistence, but also an effective mechanism for data sharing and collaboration among containers.

The data persists in the Docker host, and from the container's perspective, it is treated as a mounted volume. Hence, the data is not duplicated physically.

13.3.5 Docker Image Management

In earlier Section 13.3, images were used to create containers. An image performs like a blueprint that encapsulates all the necessary information needed to spawn a container. It includes initial configurations, requisite libraries, and other pertinent metadata. Docker images are highly portable and can be shared across various machines and platforms. This section delves deeper into the construction and functionality of Docker images.

An image shall contain everything needed to create and initialize a container. This include but not limited to:

- Necessary steps (also known as "layers") to create a container.
- Application code that is to run in the containers
- Files to support the application
- Libraries, tools and dependencies

In addition, an image shall be designed and organized in such a way that it is migratable, reusable and light, and can be used to easily populate large number of containers. For better inheritability, an image might be based on another existing image, which is called its parent image. An image with no parent, such as the official *hello-world* image from Docker Hub, is called a base image.

The Dockerfile is a text document that serves as the blueprint for constructing a Docker image, as illustrated in Fig. 13.9. Notably, the Dockerfile itself isn't included within the resulting image.

Since a Docker image's primary role is to serve as a template for containers, many Dockerfile commands appear like a "step-by-step" recipe for container creation. Each instruction corresponds to an "image layer". An image is, in essence, an amalgamation of these layers. It is stored and distributed in this format. If images share layers (for instance, different versions of the same app), these shared layers aren't saved or transferred redundantly, resulting in significantly reduced image sizes. More details can be found at *https://docs.docker.com/storage/storagedriver/*.

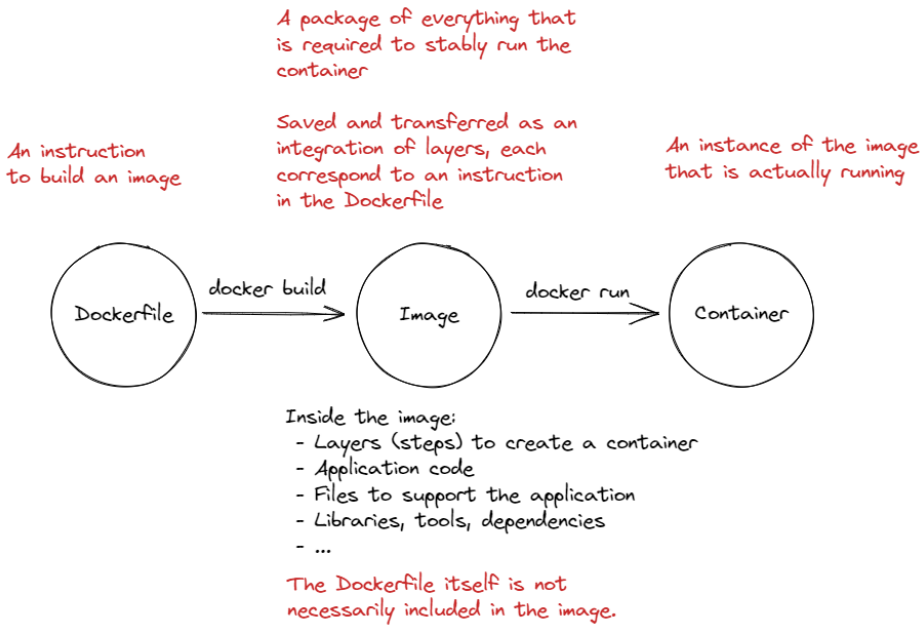

FIGURE 13.9
A demonstration of how Dockerfile, image and container link to each other.

Docker containers employ a special file system known as the Union File System (UFS), which is well suited to the "layer" concept. UFS facilitates file sharing between the container and the host machine, along with combining read-only upper layers and writable lower layers, among other functions. A demonstration to illustrate this concept is given later in Fig. 13.10.

Just as a quick example, the Dockerfile to build the official *hello-world* image from Docker Hub looks like the following.

```
FROM scratch
COPY hello /
CMD ["/hello"]
```

Like other computer languages, Dockerfiles have reserved keywords, environment variables, and syntax rules. Only the basics of constructing a Dockerfile are introduced in this section. More details can be found in the Docker reference on the official website. In the example above, `FROM scratch` signifies that this image is a base image without a parent. The `COPY hello /` instruction copies the *hello* binary script from the image to the root directory of the container. Lastly, `CMD ["/hello"]` runs the *hello* binary script.

In general, a typical Dockerfile includes the following instructions to build an image. These instructions allow the image to know how to create a container, automatically construct the file system directory structure, install necessary packages, and run the app:

(1) Define parent image.

(2) Create filesystem directory.

(3) Set working directory.

(4) Copy files.

(5) Configure registry.

(6) Install packages.

(7) Copy more files after package installation.

(8) Switch to the correct user.

(9) Expose port.

(10) Run the APP.

The keywords to be used in a Dockerfile to realize the above instructions, such as `FROM`, `RUN`, and many more, are explained in Table 13.3.

Besides Table 13.3, there are other Dockerfile keywords that can significantly simplify the design and maintenance of the image. For example, `ENV <key>=<value>` assign a value to an environmental variable; `LABEL <key>="<value>"` assigns a tag to the image, which can be displayed when `docker inspect <container>` is used.

TABLE 13.3
Critical keywords used in a Dockerfile.

Syntax	Description
`FROM <image>`	Define the parent image. A Dockerfile must start with a `FROM` instruction. A Dockerfile can contain multiple `FROM` instructions, in which case the last `FROM` statement is the final base image and the earlier `FROM` instructions creates intermediate images that can be used in the final image. An optional `:<tag>` following `<image>` can be used to specify the version of the image to use as the base. By default, the latest version of the image is used.
`RUN <command>`	Execute a shell command using `/bin/sh -c`.
`WORKDIR <path>`	Set the working directory from the point onward. This prepares the working directory for the upcoming `RUN`, `COPY`, etc., commands.
`ADD <src> <dest>`	Add (Copy) `<src>`, either a directory/-file or URL, to `<dest>`. An optional `[--chown=<user>:<group>]` can be used to specify the owner and group of the added files.
`COPY <src> <dest>`	Copy `<src>`, a directory/file, to `<dest>`. An optional `[--chown=<user>:<group>]` can be used to specify the owner and group of the added files. Notice that `COPY` is similar with `ADD`. `COPY` is easier but less powerful than `ADD`. It cannot handle tar or URL.
`USER <user>`	Switch user for the instructions beyond this point.
`EXPOSE <port>`	Specifies the ports that the container shall listen to. An optional `/<protocol>` following `<port>` can be used to specify the protocol for communication.
`CMD ["<exe>", "p1", ...]`	As the last instruction of a Dockerfile, run the APP. Notice that a Dockerfile can only contain one `CMD` instruction. The executable command name and the parameters are put into a list.

Examples of Dockerfiles are given below, one from *docs.docker.com* and the other from Linux Academy.

The docker image layer structure of the second example is given in Fig. 13.10 as a demonstration. Notice that in Fig. 13.10, `bootfs` refers to the "boot file system", including the bootloader and the Linux kernel. Upon run, a container layer will be added to the image, as shown by the blue dashed box in Fig. 13.10. In the container, all the changes made is saved into the container layer.

To generate a new image to include the changes made in the container, use `docker commit`, which essentially commits the container layer as the latest image layer in the new image, as shown by the green dashed box in Fig. 13.10.

```
# First Example
FROM golang:1.16
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go ./
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/app ./
CMD ["./app"]

# Second Example
FROM node:10-alpine
RUN mkdir -p /home/node/app/node_modules && chown -R node:node /home/
    node/app
WORKDIR /home/node/app
COPY package*.json ./
RUN npm config set registry http://registry.npmjs.org/
RUN npm install
COPY --chown=node:node . .
USER node
EXPOSE 8080
CMD ["node", "index.js"]
```

With the Dockerfile ready, use `docker build` to build an image. An example is given as follows.

```
$ docker build <path/url> -t <image name>
```

where `<path/url>` is the path or URL to the directory where the Dockerfile locates (does not need to contain "`/Dockerfile`" in its end), and `-t` gives a tag, in this case an image name, to the image to build.

The most commonly used image operations can be categorized as follows.

- Create an image.

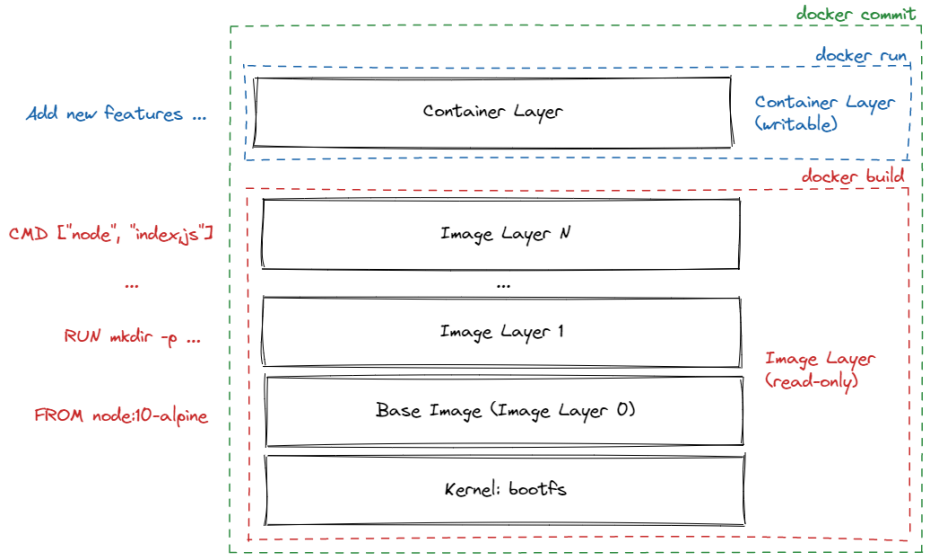

FIGURE 13.10
A demonstration of docker image layer structure using the aforementioned example.

- Create a container from an image.
- Upload and download an image from a remote server.
- Manage local images, such as listing down all images, deleting an image, etc.

The first two operations have been introduced in earlier sections. The third and last ones are introduced below. Use the following command to search for an image on the default remote repository server (Docker Hub).

```
$ docker search <image name>
```

Use the following command to download or update an image from the default remote repository server as follows. Notice that different from `docker run`, this command will not start a container from the image.

```
$ docker pull <image name>
```

Notice that since images are stored by layers, if two images share common layers, it is unnecessary to pull the shared layers repeatedly when downloading the second image, if the first image already exists in the host machine. Command `docker pull` is smart enough to automatically detect shared layers, and avoid duplicating download of layers.

Use the following commands to list down or remove images.

```
$ docker image ls
$ docker image rm <image name>
$ docker image prune # remove all problematic images
$ docker image prune -a # remove all unused images
```

where `prune` removes all problematic images, and `prune -a` removes all unused images from local.

Use the following command to inspect an image, and list down its metadata details.

```
$ docker image inspect <image name>
```

13.4 Portainer

As introduced earlier, docker engine can be used to build and share images as well as start, monitor, and stop containers. It can be difficult for a user to manage containers manually when a lot of them are deployed. Container management tools such as Protainer and Kubernetes are helpful with managing containers. Many of these tools are able to automatically adjust the number of containers and balance their loads.

Portainer is an open-source container management tool. It has a web-based dashboard user interface. Notice that Portainer itself also runs in a container.

Before starting a Portainer container, it is a good practice to first create a Docker volume for Portainer to store the database. Use the following command to create such Docker volume.

```
$ docker volume create portainer_data
```

Then run a Portainer container using

```
$ docker run -d -p 8000:8000 -p 9000:9000 -p 9443:9443 --name portainer
    --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v
   portainer_data:/data portainer/portainer-ce
```

where ports 8000, 9000 and 9443 are used for hosting HTTP traffic in development environments, hosting web interface, and hosting HTTPS or SSL-secured services, respectively. The `docker.sock` is the socket that enables the docker server-side daemon to communicate with its command-line interface. The image name for Portainer community edition (distinguished form the business edition) is `portainer/portainer-ce`.

Use `https://localhost:9443` to login to the container. The following page in Fig. 13.11 should pop up in the first-time login, asking the user to create and administration user.

After creating the admin user and logging in, the status of images, containers and many more can be monitored via the dashboard, as shown in Figs. 13.12, 13.13 and 13.14. Notice that in Fig. 13.14, using the "quick action"

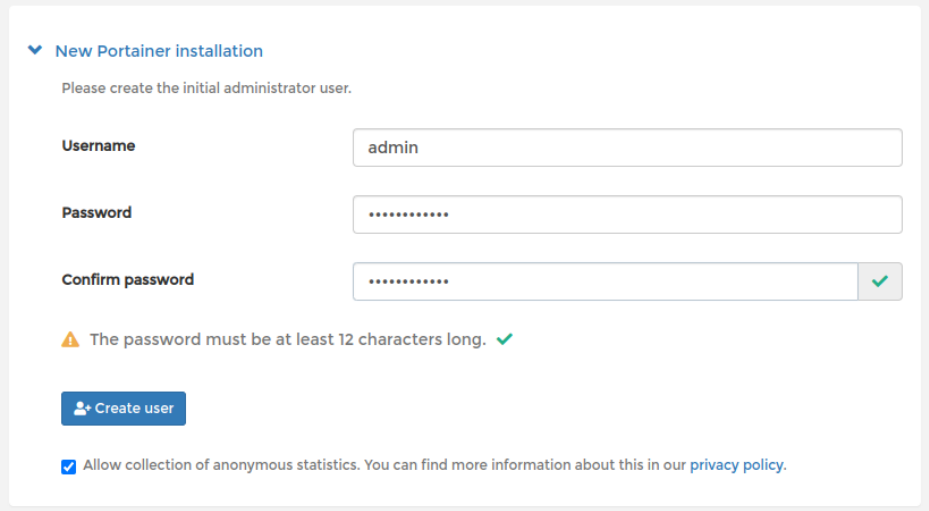

FIGURE 13.11
Portainer login page to create admin user.

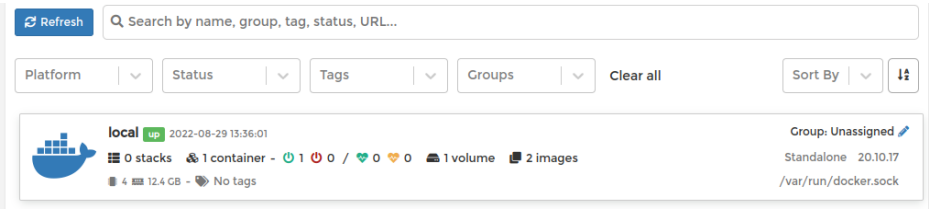

FIGURE 13.12
Portainer dashboard overview of docker servers.

buttons, the user can check the specifics of the container and interact with its console, just like using `docker container inspect` and `docker exec`

In summary, Portainer is an easy-to-use container management tool with clean graphical interface that a user can quickly get used to without a steep learning curve.

13.5 Docker Hub

Docker Hub is a commonly used server for storing and sharing docker images. It is also the default remote repository server of Docker engine. However, do notice that Docker Hub is not the only remote docker image server. Some

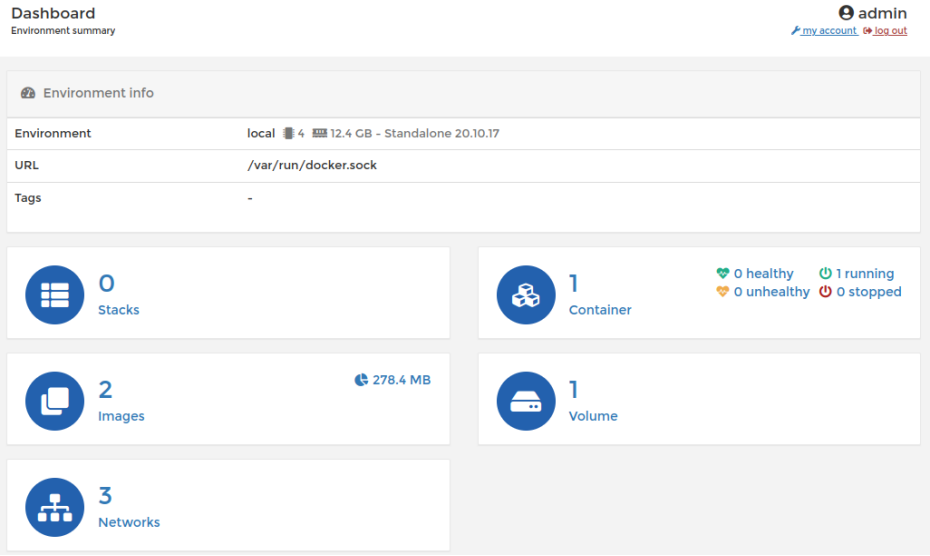

FIGURE 13.13
Portainer dashboard overview in a docker server.

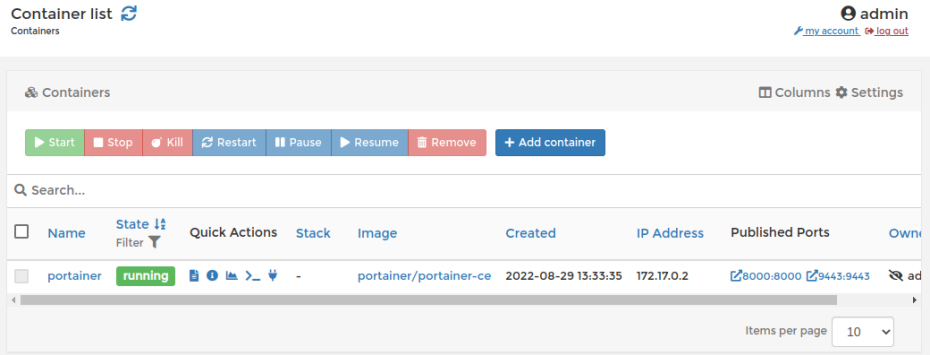

FIGURE 13.14
Portainer dashboard list down of all running containers.

alternatives are Amazon Elastic Container Registry, Red hat Quay, Azure Container Registry, Google Container Registry, etc.

After registering an account on Docker Hub, use the following command to login to the docker hub from your local machine.

```
$ docker login --username=<user name>
Passowrd:
```

Assume that there is an image in the local machine, and an empty repository on Docker Hub. In order to push the local image to the Docker Hub, the first step is to add the remote repository and the "*RepoTags*" in the local image as follows.

```
$ docker tag <image name> <username>/<repository name>:<version>
```

where `<version>` is a tag usually used to distinguish the different branches or versions of the images on Docker Hub. For the first image upload, it can simply be `latest`.

Use the following command to push the image to Docker Hub.

```
$ docker push <user name>/<repository name>
```

14

Kubernetes

CONTENTS

Kubernetes is introduced cross sections. This section focuses on a brief introduction to Kubernetes, its basic schematic architecture, and how it can be installed on a local machine.

14.1 Basic Kubernetes

Kubernetes, also known as *k8s*, is an open-source container orchestration system originally developed by Google. It automates the deployment, scaling, and management of containerized applications. While Kubernetes is more flexible than Portainer, it's also more complex and has a steeper learning curve.

Note that running Kubernetes on a local server can differ significantly from

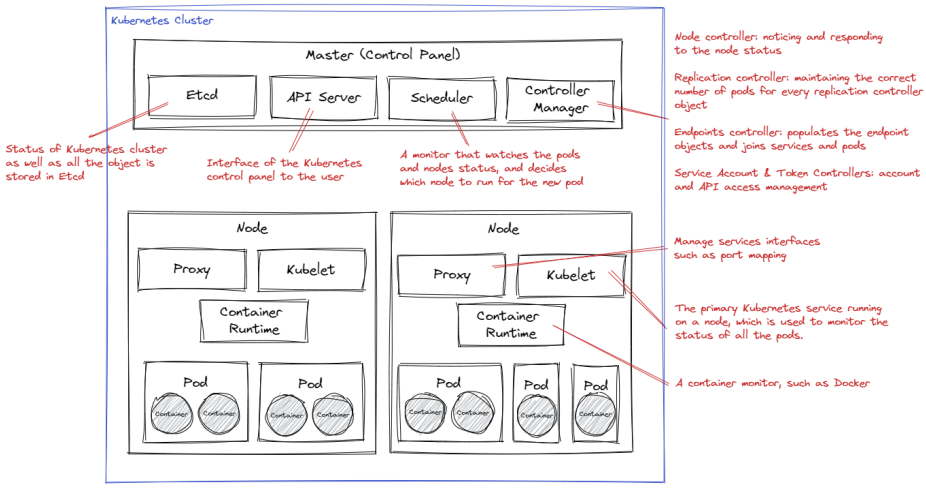

FIGURE 14.1
Kubernetes cluster and its key components.

running it on a cloud platform such as AWS or Google Cloud, which offer managed Kubernetes services with additional features and simplified management. These platforms, AWS with its Elastic Kubernetes Service (EKS) and Google Cloud with Google Kubernetes Engine (GKE), have developed their own tools and interfaces for interacting with Kubernetes. So, while learning Kubernetes can be beneficial, one might not need to learn all the low-level details if he is running Kubernetes using one of the above management tools instead of DIY everything.

This notebook focuses introducing running Kubernetes on a local server in a DIY manner. In this scenario, the following common practice is adopted: *kubectl* is used as the user interface to Kubernetes and *minikube* the software to manage the host machine. Minikube is an open-source software developed by the Kubernetes community to run a single-node Kubernetes cluster on a local machine, which is suitable for developers to learn and test different things in development environment.

Notice that in the past days, docker is the default container engine built-in to Kubernetes, and Kubernetes uses a special program "dockershim" that talks to docker engine. As of now, docker support is deprecated in Kubernetes and dockershim is removed from the installation.

14.1.1 Infrastructure

Figure 14.1 demonstrates what key components Kubernetes has inside its cluster. As shown in Fig. 14.1, Kubernetes manages containers in a centralized "master-worker" mode, where the master plays as the control panel, API

gateway (with static IP address) and load balancer to interact with a user, and the worker nodes (also known as nodes, for short) actually process the data. Each node can host multiple pods, inside each pod is a container or a group of containers that work closely together. Notice that in Kubernetes, containers never run directly in a node. They are always grouped into pods. A pod should be the minimal unit or group of container(s) that can deliver a basic function.

In practice, the master and nodes can run on cross servers or VMs. Kubernetes packages need to be installed on each and every server or VM for the system to work properly. Kubernetes provides variety of tools to distribute the loads to different servers or VMs, or to add redundancy to the system for high availability.

14.1.2 Installation

The installation guidance of Kubernetes can be found at its official website *kubernetes.io*. Notice that different OS adopts different ways of installing and using Kubernetes. The installation procedures introduced in this section applies to Linux OS only. For Windows users, Kubernetes can be installed from Docker desktop. For macOS users, other tools are used to install the tools to be introduced below.

As introduced earlier, *kubectl* is used to interact with Kubernetes. In addition, since we are in development environment, we will also install *minikube* which is used to setup a small Kubernetes cluster in the local machine. They can be installed separately. See following links for more details.

```
https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/
https://minikube.sigs.k8s.io/docs/start/
```

When running *minikube* for the first time, it would try to install *kubectl* as a built-in. Start *minikube* using `minikube start`, which will start a VM using Vertual Box, and setup a single Kubernetes node in the VM.

If *kubectl* and *minikube* are installed separately, run the following command to verify the successful installation of both software.

```
$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/
    namespaces/kube-system/services/kube-dns:dns/proxy
```

If *minikube* is installed, and *kubectl* is considered as a built-in installation to *minikube*, use the following command

```
$ minikube kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/
    namespaces/kube-system/services/kube-dns:dns/proxy
```

in which case `alias kubectl="minikube kubectl --"` may make things easier.

It is indeed possible to run Docker and Kubernetes directly on a host machine if that machine is running a Linux OS. However, this is typically not recommended for reasons pertaining to access control, security, and isolation. It is generally better practice, particularly in a development environment, to deploy your Kubernetes cluster within a virtual machine.

The `kubectl` command is a command-line interface (CLI) for interacting with Kubernetes clusters. Its behavior is governed by a configuration file, typically located at `~/.kube/config`, which determines which Kubernetes cluster `kubectl` communicates with. This could be a cluster running on the host machine, or a cluster running in a VM managed by tools like Minikube. It's worth noting that, for better isolation and control, it is recommended to deploy the Kubernetes cluster in a VM, instead of directly on the host machine.

14.1.3 Kubernetes Configuration Files

Managing containers via container orchestration such as Kubernetes is different from doing so using container engines such as docker engine. Unlike docker engine where we started from building an image from a dockerfile, Kubernetes requires that all images to be used are already available. Instead of having one configuration file where each entry in that file corresponds with a container, when using Kubernetes, multiple configuration files are required, each file corresponding with an object to be created. Notice that an object is not necessarily a container. It can be a pod, a replica controller, a service, or any other "thing" in the Kubernetes framework. There are more involving manual setups of networking when using Kubernetes as well. Details are introduced in the remaining of this section.

The following two configuration files are given as examples [1] to demonstrate how these files look like. This example comes from Udemy course *Docker and Kubernetes: The Complete Guide* by Stephen Grinder. They are both written in YAML. Configuration file to setup a pod:

```
apiVersion: v1
kind: Pod
metadata:
name: client-pod
labels:
component: web
spec:
containers:
- name: client
image: <image-name>
ports:
- containerPort: 3000
```

Configuration file to setup the networking service:

```
apiVersion: v1
kind: Service
metadata:
name: client-node-port
spec:
type: NodePort
ports:
- port: 3050
targetPort: 3000
nodePort: 31515
selector:
component: web
```

Commonly used Kubernetes object types are summarized in Table 14.1. Some highlights of the above configuration files are as follows.

- `apiVersion` plays as the prefix that decides what configuration types are supported. For example, under the scope of `v1`, `Pod`, `Service`, `configMap`, `Namespace`, etc., are supported. In a different `apps/v1`, a different set of configuration types `ControllerRevision`, `StatefulSet`, etc., are supported. It's important to choose the correct apiVersion for the Kubernetes API version you are working with to ensure the compatibility and availability of the desired configuration options.

- `kind` This indicates the type of the object that the configuration file describes. For example, `Pod` represents a pod that is used to host containers, and `Service` the primary object type that defines networking, with subtypes `NodePort` (see example above), `ClusterIP`, `LoadBalancer` and `Ingress`.

- `metadata` indicates the name and labels of the object. For example, `component: web` is defined as a label of the pod. This information is passed to the networking service under `selector`, so that the networking service knows which object it should link to.

- `port`, `targetPort` and `nodePort` are used to specify ports used in the service. The `targetPort` indicates which port in the pod should be exposed to the service, and it is consistent with `containerPort` defined in the pod. Assume that there is another pod in the node who needs to talk to this pod via the service. The other pod's port that communicates with the service is fed into `port`. Finally, `nodePort` is the port with value between 30000 and 32767 that is exposed from the service to outside the node. If `nodePort` is not assigned, a random number within the range will be assigned. This is shown in Fig. 14.2.

Notice that Kubernetes networking using `kind: Service` is more complicated than shown in the above example. More details of it is given later in a dedicated Section 14.2.2.

TABLE 14.1
Commonly used Kubernetes object types.

Object Type	Description
Pod	The smallest and most basic unit in the Kubernetes object model. It represents a single instance of a process running on the cluster.
Deployment	Manages the deployment and scaling of a set of identical pods, ensuring the desired number of replicas are running and providing rolling updates for seamless application upgrades.
Service	Enables network access to a set of pods using a stable IP address and DNS name. It provides load balancing across multiple pod replicas and allows external traffic to be directed to the appropriate pods.
ConfigMap	Stores configuration data in key-value pairs, which can be consumed by pods as environment variables, command-line arguments, or mounted as files.
Secret	Similar to a ConfigMap, but specifically designed to store sensitive data, such as passwords, API keys, and TLS certificates. Secrets are encrypted at rest and can be mounted into pods as files or exposed as environment variables.
PersistentVolume	Provides a way to provision and manage persistent storage resources in a cluster. It decouples the storage from the underlying infrastructure and allows data to persist beyond the lifecycle of individual pods.
PersistentVolumeClaim	Requests a specific amount of storage from a PersistentVolume. It acts as a request for a specific storage resource and provides an abstraction layer for managing persistent storage in a cluster.
Ingress	Manages external access to services within a cluster. It acts as a reverse proxy and exposes HTTP and HTTPS routes to route traffic to the appropriate services based on hostnames, paths, or other rules.

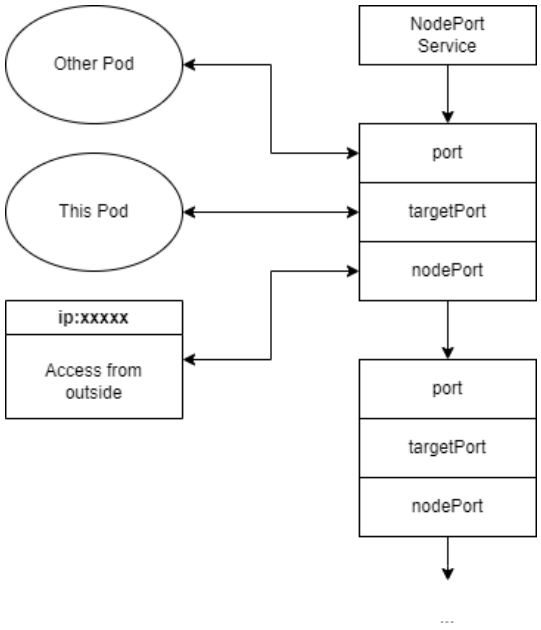

FIGURE 14.2
The NodePort networking service.

14.1.4 Cluster Deployment

With the image and the configuration files ready, the next step is to deploy the nodes, pods, and containers. *kubectl* command line interface is used to instruct Kubernetes to deploy the objects as follows.

```
$ kubectl apply -f <configuration file>
```

This essentially asks the master node in the Kubernetes cluster to start taking actions according to the configuration files, such as to inform the nodes to start creating pods and containers. The master node also keeps monitoring the status of each work node, to make sure that everything is running as planned. If there is a container failure, etc., the master node will guide the associated note to restart the container.

It is worth mentioning here that by default Kubernetes uses declarative deployment instead of imperative deployment, meaning that the developer does not need to specifically tell Kubernetes what to do in each step. The developer only tells the overall objectives, and Kubernetes master node will try to figure out the steps to realize that goal. It is possible to enforce Kubernetes master node to practice specific details via configuration files, but it is almost always recommended to use the default declarative approach with Kubernetes.

To retrieve information, such as the status, of a group of objects, use

```
$ kubectl get <object type>
```

where `<object type>` can be `pods`, `services`, etc. For more details of a specific object, use

```
$ kubectl describe <object type> <object name>
```

for example, to check the containers running in a pod. If `<object name>` is neglected, Kubernetes returns detailed information of all objects of the given object type. For a running object, use

```
$ kubectl logs <object name>
```

to check the log file of that object.

With the above been done, open a browser and use `<ip>:<port>` to access the application running in the container, where `<ip>` is the IP address of the VM (not `localhost`) that *minikube* created, and `<port>` the port configured in NodePort service under `nodePort`. The IP address can be found by running `minikube ip`.

To apply a group of configuration files all together, provide the directory name of all the configuration files to Kubernetes instead of feeding each configuration file one at a time.

```
$ kubectl apply -f <directory>
```

When directory is given instead of a file, Kubernetes will try to apply all the configuration files in that directory.

It is possible to consolidate the configuration files of objects into one conjunctive configuration file. To do that, use `---` to split the configurations for

each object in the conjunctive configuration file as follows. It is of personal preference whether to use conjunctive configuration files or separate configuration files for all objects.

```
<configurations-for-object-1>
---
<configurations-for-object-2>
---
<...>
```

14.1.5 Cluster Update

Without container orchestration such as Kubernetes, one of the most challenging tasks is to update the container for a different configuration, for example, changing the underlying image. With the help of Kubernetes declarative approach, it is possible update the cluster simply by revising the configuration files, and pass them to Kubernetes as if the cluster is to be deployed for the first time. Kubernetes automatically checks the names and kinds of the revised configuration files, comparing them with existing running objects, and update them if necessary.

Check the status of the pods using `kubectl get pods`. After updating, the pods are often restarted, hence it is expected to see increment in the "RESTARTS" tag. To double confirm that updates have been made, use `kubectl describe` to check the details of the relevant objects.

However, there is a limitation to the updating of the Kubernetes deployment. For an existing object, only certain fields in the configuration files can be changed. For example, for a pod that runs containers, the image can be changed, but the container port cannot. Sometimes there can be a walk around. For example, in the case of changing container port of pods, consider using a new object type `Deployment` instead of `Pod`, which allows more flexible updating. The `Deployment` in its backend is consist of one or more monitored and managed identical pods.

To revert `kubectl apply`, i.e., to remove a configuration file, use

```
$ kubectl delete -f <configuration file>
```

Kubernetes treats the above delete command as a specific type of update to the cluster, and will action accordingly.

14.2 Advanced Kubernetes

This section introduces advanced commonly used Kubernetes objects, tools and techniques.

14.2.1 Kubernetes Object: Deployment

As introduced in Section 14.1.5, updating pods has some limitations. It is practically more convenient to setup pods using "Deployment" object instead of "pod". The Deployment object servers as an additional layer of Kubernetes infrastructure that manages identical pods. More details of Deployment object is introduced in this section.

As an example, here is a configuration file from Kubernetes manual that deploys a Deployment object.

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-deployment
labels:
app: nginx
spec:
replicas: 3
selector:
matchLabels:
app: nginx
template:
metadata:
labels:
app: nginx
spec:
containers:
- name: nginx
image: nginx
ports:
- containerPort: 80
```

Some highlights are as follows.

- `replicas` gives the expected number of pods that the Deployment object manages.
- `matchLabels` specifies the pods with which label are to be managed by the Deployment object. In this example, the label is `app: nginx`. When populating pods, the pods would have the same label, as the same label is assigned under `template`, `metadata`, `labels`.
- `template` specifies the template that is used to create the pods.

When a new version of an image becomes available, we may want to update the containers accordingly. Re-apply the same configuration file would not help, as Kubernetes would reject apply request if no change is detected in the configuration file. It would not check whether the image is in its latest version. Kubernetes uses the following imperial command to update images as a walk around, and the developer needs to run this command manually.

```
$ kubectl set image deployment/<Deployment name> <container name
    >=<image name>
```

For example,

```
$ kubectl set image deployment/nginx-deployment nginx=nginx
    :1.25.1
```

14.2.2 Kubernetes Object: Service

There are 4 service types defined in Kubernetes. So far "NodePort" service type has been introduced in earlier examples. More types are introduced here.

A summary of different service types are given below.

- ClusterIP: Exposes the service object on a cluster-internal IP. Objects in the cluster can access to the object that a ClusterIP is pointing at.
- NodePort: Assigns a static port with the cluster IP, and exposes the Service object to the internet. This is used mostly used in development environment, not in production environment.
- LoadBalancer: Exposes the service object externally using an external load balancer. Kubernetes does not provide built-in load balancer.
- ExternalName: Maps the service object to the contents of the `externalName` field, such as a host name. This is related to cluster DNS server.

ClusterIP

A ClusterIP configuration file may look like the following.

```
apiVersion: v1
kind: Service
metadata:
name: client-cluster-ip-service
spec:
type: ClusterIP
selector:
<target tag>
ports:
- port: <port for internal comm>
targetPort: <port for internal comm>
```

where the first `port for internal comm` is the port in the ClusterIP service object that opens to other objects in the cluster, and the second the port the target object opens to the ClusterIP service object. They can be set differently, but usually they are just set to the same value.

NodePort

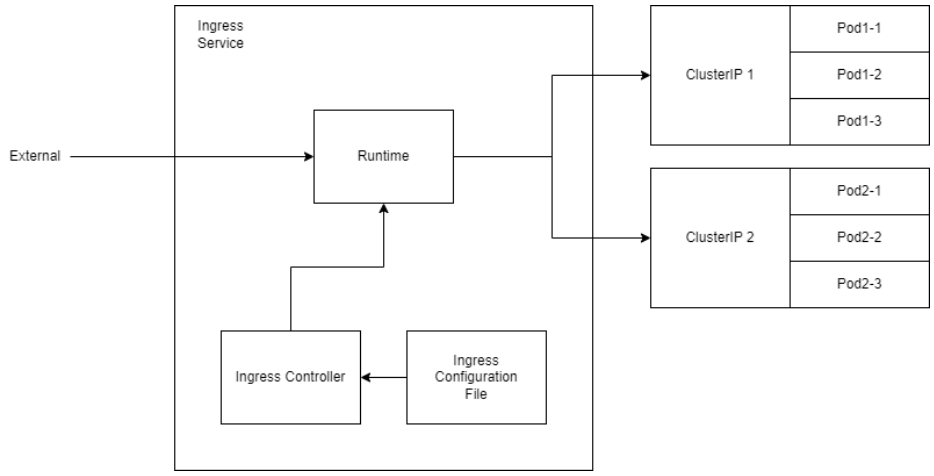

FIGURE 14.3
An example of ingress service framework.

NodePort has already been introduced earlier. It exposes the object to the internet with a static port, and it is used more in a development environment than a production environment.

LoadBalancer

LoadBalancer is a legacy way of getting network traffic into the pods. It is essentially an interface or a tool to bridge an Kubernetes Deployment with an external load balancer. It will try to automatically configure the external load balancer using the configuration provided by the developer, while compromising the external load balancer rule.

Ingress

Ingress is a more commonly used Service type than LoadBalancer to get traffic into the Kubernetes containers. There are different types of Ingress, for example, Nginx Ingress by *github.com/kubernetes/ingress-nginx*. An demonstrative example of ingress service realization is given in Fig. 14.3. In this implementation framework, the configuration file (mainly a set of routing rules) of the object is used to define an "Ingress Controller" which manages the runtime that controls inbound traffic. In some applications such as *kubernetes/ingress-nginx*, the ingress controller and the runtime are integrated together.

The ingress configuration differs depending on the ingress service type and the platform. Details are not given here.

14.2.3 Kubernetes Object: Persistent Volume Claim, Persistent Volume, and Volume

Docker engine uses volumes to maintain persistent data and share data among containers. Details have been introduced in Section 13.3.4. Kubernetes volume framework is similar in the sense that it makes sure that the data is saved and managed by the host machine, so that when the pods or containers are shutdown or restarted, the data can be restored safely.

Do notice that when comes to data sharing using volume, it is dangerous to have multiple containers or the host machine accessing the same files simultaneously, without knowing the existence of each other. Usually additional steps need to be setup to ensure data consistency.

It is worth emphasizing the differences of "volume" technology in containerization and Kubernetes volume-related objects: Persistent Volume Claim (PVC). Persistent Volume (PV), and Volume. As a matter of fact, Kubernetes Volume object is usually not what we want. Kubernetes Volume creates the volume tied to a pod, not to the host machine. It survives container failure in the pod, but not pod failure. In summary:

- Kubernetes Volume: A volume tied to the pod. It survives container failure inside the pod, but not pod failure.
- Kubernetes PV: A volume tied to the host machine. It survives pod failure. It can be provisioned either automatically by a StorageClass, or manually by the developer and administrator.
- Kubernetes PVC: It is essentially a request sent from a pod or a container, asking for specific amount of storage from a PV. Kubernetes will find that amount of PV from either existing provisioned static PV, or dynamically provision new ones for the pod or container.

Notice that it is not necessary to claim PV in order to use PVC, as PVC can provision PV dynamically. There is a one-to-one relationship between the provisioned PV and the PVC. If there are multiple pods, each requiring a dedicated PV, then multiple PVCs must be used. The developer can either create those PVCs manually, or use a Volume Claim Template to claim them if they are similar.

An example of claiming Kubernetes PV and PVC is given below. In the remaining part of this section, we will be mostly using PVC instead of PV.

```
# persistent-volume.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
name: my-pv
spec:
storageClassName: standard
```

```
capacity:
storage: 10Gi
accessModes:
- ReadWriteOnce
hostPath:
path: /data/my-pv
---
# persistent-volume-claim.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: my-pvc
spec:
storageClassName: standard
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 5Gi
```

To check the PV objects, use `$ kubectl get pv` and `$ kubectl get pvc`.

To add the above Kubernetes PVC to a Kubernetes Deployment, add volumes information to the specs as given in the following example.

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: my-app-deployment
spec:
replicas: 1
selector:
matchLabels:
app: my-app
template:
metadata:
labels:
app: my-app
spec:
containers:
- name: my-app-container
image: my-app-image
ports:
- containerPort: 8080
volumeMounts:
- name: data-volume
mountPath: /data
subPath: data-from-container
volumes:
- name: data-volume
```

```
persistentVolumeClaim:
claimName: my-pvc
```

where `volumes` defines which Kubernetes PVC is used, and `volumeMounts` tells how it is mounted in the container. The `mountPath` is the path in the container whose data is mounted by the volume. If a `subPath` is given, a subfolder of its specified name will be created in the host machine in the volume to contain the data.

There are different types of access modes:

- `ReadWriteOnce`: Allow one node to read and write at a time.
- `ReadOnlyMany`: Allow many nodes to read at a time.
- `ReadWriteMany`: Allow many nodes to read and write at a time.

The developer can specify the place for Kubernetes PVs. This is usually the hard drive on a local server, a virtual storage space in the VM. Use the following command to check Kubernetes possible choice of storage.

```
$ kubectl get storageclass
```

and

```
$ kubectl describe storageclass
```

When deploying Kubernetes on the Cloud, the developer needs to do additional configurations as there would be many storage options. Usually, each Cloud provider will have its own default storage space for Kubernetes, such as AWS Elastic Block Store for AWS.

14.2.4 Kubernetes Object: Secrets

Kubernetes Secrets object is used store confidential information, such as the database password, API key, etc. It is often a piece of information that is necessary for the containers, but the developer does not want to present as plain text in the configuration file.

Secrets are not created from configuration files, which is the recommended way of creating other Kubernetes objects. Instead, it is created from one-time imperative command, inside which the confidential information needs to be told to Kubernetes. Use the following command to create a Secret object.

```
$ kubectl create secret <type-of-secret> <secret-name> --from-
    literal <key>=<value>
```

There are 3 types of Secrets: `generic`, `docker-registry` and `tls`.

14.2.5 Kubernetes Environment Variables

Kubernetes environment variables are used to pass or share information among Deployments. Depending on the features of the information, such as whether

it is a constant global configuration or a dynamic value, whether it is plain text or confidential encoding, etc., it might be handled differently.

To define constant environment variables in containers, simply specify them in the Deployment configuration file as given in the example below.

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: my-app-deployment
spec:
replicas: 1
selector:
matchLabels:
app: my-app
template:
metadata:
labels:
app: my-app
spec:
containers:
- name: my-app-container
image: my-app-image
ports:
- containerPort: 8080
volumeMounts:
- name: data-volume
mountPath: /data
subPath: data-from-container
env:
- name: <name1>
value: <value1>
- name: <name2>
value: <value2>
- name: <name3>
valueFrom:
secretKeyRef:
name: <secret-name>
key: <key>
volumes:
- name: data-volume
persistentVolumeClaim:
claimName: my-pvc
```

where a new tag `env` is defined under template, specifications, containers. Under the `env` tag, a list is defined containing names and values of the environment variables. The value must be a string, not a numerical number.

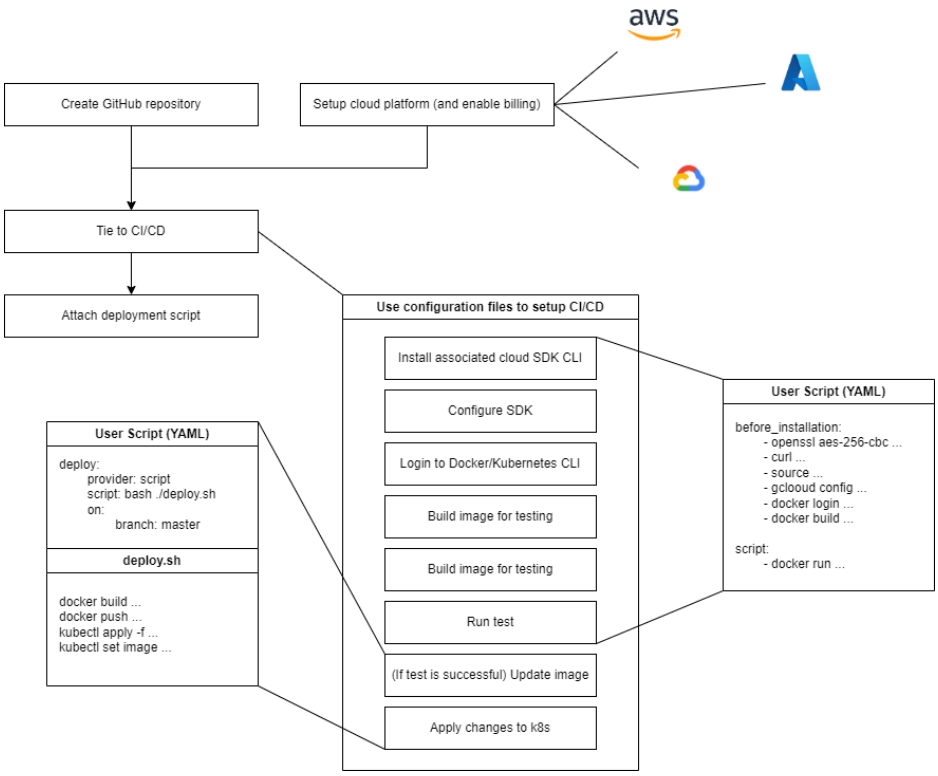

FIGURE 14.4
An example workflow of creating a production environment with Kubernetes.

14.3 Container Deployment in Production Environment

With the tools and methodologies introduced so far, we are able to deploy containers in development environment. This is good enough for testing purpose or for small individual projects. However, when comes to enterprise tier projects or collaborative projects, there is often a CI/CD pipeline that standardize the integration and delivery of the containers in production environment. Container orchestrations such as Kubernetes is often a must have.

This section briefly introduces the steps to develop and deploy containers in production environment with Kubernetes. Figure 14.4 gives an example of overview of what such deployment may look like. Notice that this example is more towards a community project but not an enterprise project.

The example used in this section to demonstrate Kubernetes CI/CD on a cloud platform in production environment is taken from [1]. Following [1], Google Cloud Platform is used as the cloud platform provider.

14.3.1 Setup Cloud Account

Many cloud platforms nowadays have a very good support of Kubernetes. The deployment of containers using Kubernetes can be done via their UIs easily. The developer needs to decide the resources to be used for the deployment. The more nodes and more power machine, the higher the charge. In most cases, the developer does not need to start a VM and install Kubernetes on it by himself. The cloud provider shall have dedicated Kubernetes engine service that would automatically configure the VM per required.

14.3.2 Configure CI/CD

Travis CI is a continuous integration service tool written. It can be deployed on the cloud and linked to a Github repository and a CI/CD platform such as a Kubernetes cluster on Google Cloud Platform.

To run Travis, a machine supporting Ruby programming language is required. For that, a separate container with Ruby installation is deployed to run Travis. Use Github credentials to login to Travis, so that it can link to the Github repositories.

Travis has built-in file encryption function. This function is mainly used to encrypt login credentials and service account credentials (in this example, the service account information to link to Kubernetes clusters on Google Cloud Platform) locally, so that later the unencrypted original credential files can be deleted, and only the encrypted credentials uploaded to the Github repository. When encrypting a file, Travis will also guide the user on how to call the encrypt information in the build script.

Travis uses configuration files to setup CI/CD pipeline.

14.3.3 Deploy Containers

In the Travis configuration file, `deploy` is used to specify the script to run when the testing is successful. A separate bash script `deploy.sh` is defined for this purpose, inside which is a sequence of commands that builds and publishes images, and configures Kubernetes using `kubectl`.

It is particularly worth mention that in `deploy.sh`, when building and applying the latest version of the docker image, tagging using `<image-name>:latest` alone is not going to work for the same reason explained earlier in Section 14.2.1: when the same configuration with `<image-name>:latest` is applied, the system would simply acknowledge it as "no change" and would not actually download the latest version of the image.

The walk around introduced in Section 14.2.1 was to use version number in the configuration file and/or as an imperial command as follows.

```
$ kubectl set image deployment/<Deployment name> <container name
    >=<image name>:<version>
```

so what when the version name changes, Kubernetes would notice the differences and apply the new image. When working with CI/CD using *Git*, this can be further automated. Just use the `$GIT_SHA` as part of the tag as follows.

```
docker build -t <docker-id>/<image-name>:latest -t <docker-id>/<
    image-name>:$GIT_SHA -f <dockerfile> <save-directory>
docker push <docker-id>/<image-name>:latest
docker push <docker-id>/<image-name>:$GIT_SHA
```

Notice that in addition to `:latest`, `:$GIT_SHA` is used as a secondary tag. When pushing the built image to Docker hub, both `:latest` and `:$GIT_SHA` are pushed (although they have identical content). When setting image, the `$GIT_SHA` is used to identify the image just like the version number.

It is recommended not to remove `:latest` in the building command. This is because if someone wants to pull and test the latest image in his server (without knowing the value of `$GIT_SHA` for the latest commit), he is still able to do so using only the image name.

Notice that `$GIT_SHA` is not a built-in environment variable. The developer needs to set that environmental variable manually in the configuration YAML file as follows. It is possible to replace `$GIT_SHA` with a different name.

```
env:
global:
- GIT_SHA=$(git rev-parse HEAD)
```

With the above setup, `$GIT_SHA` can be used in `deploy.sh` as an environmental variable.

14.3.4 Manage Secrets

Notice that when CI/CD tool is tied to the cloud platform provider, service account authentication is required. It is a good habit to NOT to put the authentication information in the CI/CD configuration file as plain text, or to upload the unencrypted file that contains the authentication information to the public workspace. It is possible that some CI/CD tools provide encryption tools that can be used to encrypt the authentication file. In such case, the developer may need to install the required CLI for that CI/CD tool.

In Section 14.2.4, it has been introduced that Kubernetes uses Secrets object to encrypt secret files. The encrypted secret files can then be safely published online. In the Kubernetes configuration file, an environmental variable can be created to call the secret information.

Many cloud platform providers including Google Cloud Platform provides services to manage secrets.

14.3.5 Helm

To install a software in a Kubernetes pod, the most intuitive way is to commit the installation in the image, and call the image in the Kubernetes configura-

tion file. For commonly used services such as `ingress-nginx`, its installation configuration file is available online as part of the manual. It essentially starts and initializes a branch of Kubernetes objects to enable the service.

Helm is designed as an alternative to manage software installation in Kubernetes clusters. In many occasions, it is more convenient (or even only possible) to use Helm to install a software. More details are given in *github.com/helm/helm*.

We need to first install Helm from script. Helm installation used to contain two parts, the CLI (referred as Helm client) and the server (referred as Tiller server). We could then use Helm CLI to install other third-party software and tools.

Access control is important on cloud platforms. In practice, user accounts are used to identify users, and service accounts to identify pods and programs. Associated role bindings are used to manage what resources can be accessed by a user or program. For example, administrative role over the entire account can be used to bind with the administrative user. The same applies to Helm. The Tiller server required some extent of administrative control over the resources in an account. In many occasions, Tiller server was given the administrative permission to access the entire account, which introduced security risks.

As of Helm version 3, a major change was carried out where Tiller server was removed completely. Helm architecture is more secure and simpler today. The concerns related to Tiller's permission in the Kubernetes cluster are no longer relevant. Helm 3 of course requires permissions to use the resources, which is now managed by Kubernetes role-based access control mechanisms.

Part IV

Linux Security

Part V

Linux on Cloud

15

Cloud Computing

CONTENTS

This chapter gives a general introduction to cloud computing and how Linux is used as the backend of most of these cloud computing services.

16

Amazon Web Service

CONTENTS

As an example, this chapter introduces the solution architecture and basic development procedure when using Amazon Web Service (AWS), one of the biggest cloud computing service providers.

Part VI

Appendix

17

Scripts

CONTENTS

This appendix chapter gives the scripts used in this notebook.

17.1 *Vim* Configuration `vimrc` Used in Section 3

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

inoremap jj <Esc>
noremap j h
noremap k j
noremap i k
noremap h i

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

syntax on
colorscheme onedark

set number
set cursorline
set wrap
set wildmenu

set hlsearch
exec "nohlsearch"
set incsearch
```

```
set ignorecase
noremap <Space> :nohlsearch<CR>
noremap - Nzz
noremap = nzz

noremap sj :set nosplitright<CR>:vsplit<CR>
noremap sl :set splitright<CR>:vsplit<CR>
noremap si :set nosplitbelow<CR>:split<CR>
noremap sk :set splitbelow<CR>:split<CR>
noremap <C-j> <C-w>h
noremap <C-l> <C-w>l
noremap <C-i> <C-w>k
noremap <C-k> <C-w>j
noremap J :vertical resize-2<CR>
noremap L :vertical resize+2<CR>
noremap I :res+2<CR>
noremap K :res-2<CR>

set scrolloff=3
noremap sc :set spell!<CR>
```

18

Conclusions and Future Work

CONTENTS

...

Bibliography

[1] S. Grinder, "Docker and kubernetes: The complete guide," 2023.