Lu Sun, and many more.

A Notebook on Linux Operating System

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

Contents

Foreword

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

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

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

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

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

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

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

—

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

This notebook is suitable as:

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

This notebook is NOT suitable as:

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

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

—

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

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

Preface

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

List of Figures

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 comparing with other operating systems.

1.1 Brief Introduction

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

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

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

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

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

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

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

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

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

the source code had to be higher than other OS (such as Microsoft Windows). Other companies, such as SCO and Sun Microsystems, were more successful than AT&T by selling UNIX based PC and workstations for high-end users. Overall, UNIX source code was extremely expensive.

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

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

1.3 Linux Distributions

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

Today, there are hundreds of Linux distributions in the community. The most famous two categories of distributions are as follows.

- Red Hat Distribution
  - Red Hat Enterprise Linux (RHEL)
  - Fedora
  - CentOS
- Debian Distribution
  - Ubuntu
  - Linux Mint
  - Elementary OS
  - Raspberry Pi OS

Some of the main features of Red Hat distributions are as follows. Red Hat

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

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

CentOS is a "recreation" simplified version of RHEL using freely available RHEL source code. Recently, Red Hat took over support of CentOS project.

Fedora, different from RHEL, is a free, cutting-edge Linux distribution sponsored by Red Hat. It is less stable 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.

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

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

1.4 Linux Graphical Desktop

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

Figs. 1.1, 1.2, 1.3 and 1.4 give the flavors of each desktop environment

FIGURE 1.1
GNOME desktop environment.

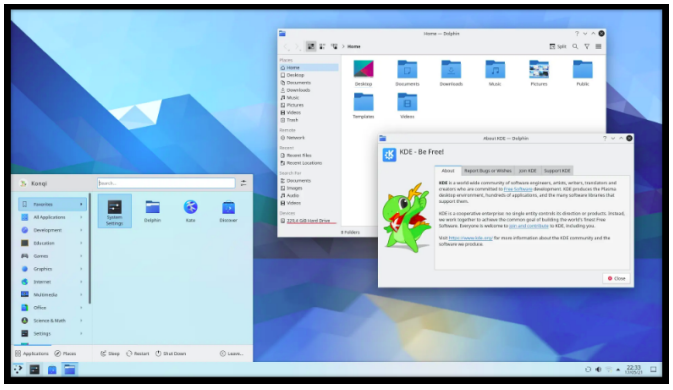

FIGURE 1.2
KDE desktop environment.

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

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

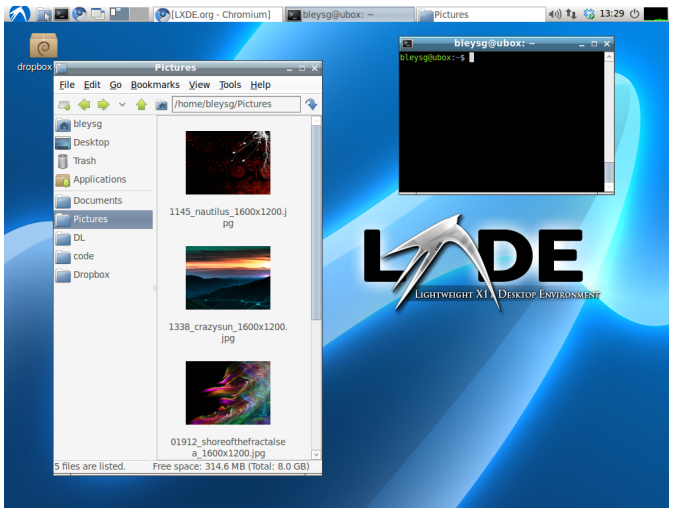

FIGURE 1.3
LXDE desktop environment.

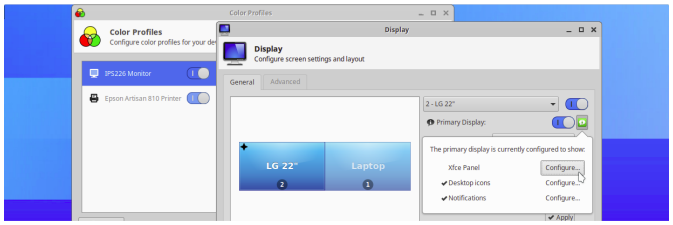

FIGURE 1.4
Xfce desktop environment.

1.5 Linux Installation

Linux can be installed on a local PC hard disk, or on a mobile device such as a thumb drive. The installation of different distributions might differ. Thanks to the graphical installation tools for the popular distributions, the installations can be done easily by just following the instructions on the associated official sites.

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

2

Shell

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 and flexible, and well-supported by the community.

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

2.1 Brief Introduction

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

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.

Some other shells such as "C Shell" and "Korn Shell" are also popular among certain users or certain Linux distributions. In case where your Linux distribution does not have these shells pre-installed, you can install and use these shells just like installing other software.

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

2.2 Basic Concepts

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

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

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

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

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

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

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

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

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

2.3 Useful Commands

Some useful commands are introduced in this section by categories. Notice that 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
```

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

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

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

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

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

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.

Show User Information

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

The following commands show basic information of a user.

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

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

Show System Information

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

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

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

```
$ sudo lshw
```

Navigate Files and Folders

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

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

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

Alias and Shortcuts

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

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

A temporary alias can be added to the shell by using

```
$ alias <shortcut command>='<original command and options>'
```

for example

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

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

2.4 Basic Shell Programming

A truly power feature of the shell is its ability to redirect inputs/outputs of commands, thus to chain the commands together. Meta-characters pipe (`|`), ampersand (`&`), semicolon (`;`), right parenthesis (`)`), left parenthesis (`(`), less than sign (`<`) and greater than sign (`>`) are used for this feature.

3

Text File Editing

CONTENTS

“nobreak

3.1 Text File Editing Environment in Linux

“nobreak

3.2 Vim

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

Some people claim *Vim* to be the most powerful text file editor as well as integrated development environment for programming on a Linux machine (and potentially on all computers and servers). The main reasons are as follows.

- *Vim* is usually built-in to Linux during the operating system installation, making it the most available and cost-effective text editor.
- *Vim* can work on machines where graphical desktop is not supported.

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

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

3.2.1 General Introduction to Vim

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

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

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

```
$ vim testvim
```

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.

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.

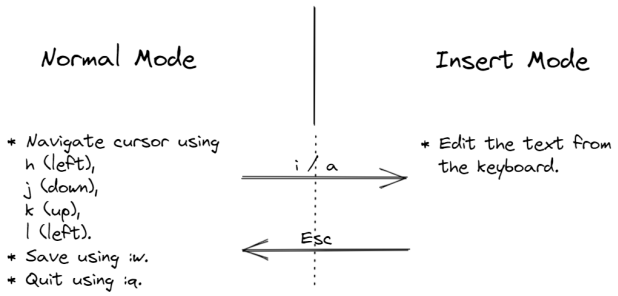

FIGURE 3.1
Mode switching between normal mode and insert mode, and basic functions associated with the modes.

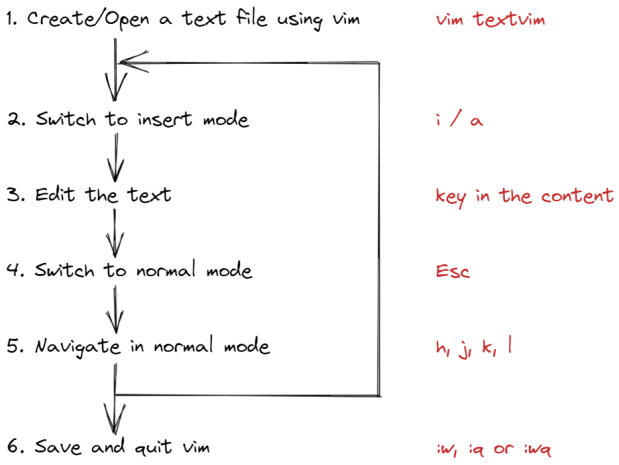

FIGURE 3.2
A flowchart for simple creating, editing and saving of a text file using *Vim*.

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.

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

3.2.2 Configure Customizable User Profile

With the basic operations introduced in Section 3.2.1, we are able to create and edit a text file as we want to, just like using any other text editor. Though at this point the advantages of using *Vim* over other text editors are not obvious yet, the *Vim* editor is at least useable.

Before introducing more advanced features of *Vim* for more convenient user experience, we can now customize the user profile to suit our individual habit. Notice that the customization is completely optional and personal. This section only introduces the ideas and basic methods of such customization,

such as re-mapping keys and create user-defined shortcuts. Everything introduced here are merely examples and it is completely up to the user how to design and implement his own profile.

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

Mapping of Keys

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

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

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

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

The upper case letter `S` and lower case letter —s— in control mode are originally used to delete and substitute texts. They may be not so important in practice as there functions are overlapped by another shortcut key `c`, which is powerful in replacing characters and is more frequently used. We can re-map `S` for saving the text, and disable `s` to prevent mis-touching. Similarly, upper case letter `Q` is mapped to quit *Vim*.

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

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

Syntax Highlight, Color Scheme and Others

By default *Vim* displays white color contents on black background. Use

the following command in *vimrc* to enable syntax highlighting or change color scheme. Use `:colorscheme` in normal mode in *Vim* to check for available color schemes.

```
syntax on
colorscheme default
```

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

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

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

```
set wildmenu
```

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

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

3.2.3 Commonly Used Operations in Normal Mode

The operations, such as delete, cut, copy, paste, replace and search, are mostly done in normal mode through shortcut keys. For example, `dd` delete (cut) the entire row at the cursor and `p` paste the row to its new position. For beginners, remembering shortcut keys can be difficult. In such case, it is recommended for us to look for the consistent patterns of the different commands, instead of brute-force remembering the keys only.

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

```
<operator><motion>
```

The operator command tells *Vim* what to do (say, copy), and the motion command tells the applicable range of the operation (say, the entire row, or the single word, or the single character, of the cursor position). Of course for some operator commands, they can be used alone without the motion command.

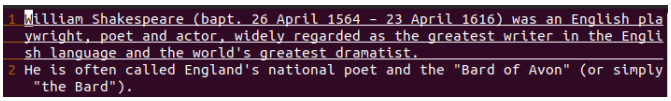

FIGURE 3.3
A piece of text of "William Shakespeare", for demonstration.

Delete/Cut, Change, Copy and Paste

We will use the most commonly used operator commands, delete/cut, change, copy (also known as "yank" in *Vim*) and paste to demonstrate the above idea.

In this demonstration, we will be editing the following lines taken from Wikipedia under "William Shakespeare". In the text file, each sentence takes a new row as given by Figure 3.3.

> William Shakespeare (bapt. 26 April 1564 – 23 April 1616) was an English playwright, poet and actor, widely regarded as the greatest writer in the English language and the world's greatest dramatist.
>
> He is often called England's national poet and the "Bard of Avon" (or simply "the Bard").

To quickly delete/cut a single character, use either `x` or `X`. Each time `x` is input in normal mode, it deletes the current cursor selected character, and automatically select the next character in the text. Each time `X` is input in normal mode, it keeps the current cursor selected character while deleting the previous character in front of the cursor. In this sense, `x` and `X` play like `delete` and `backspace` respectively in other text editors such as *notepad++*.

Operators `x` and `X` do not require consequent motion command, as they simply delete/cut one character immediately each time they are pressed. What if you want to delete multiple characters from the cursor? You can press `x` or `X` multiple times, or alternatively you can ask *Vim* to do that repeatedly for you, as long as you tell *Vim* what actions (key combinations) and how many times you want perform. For example, `20x` is equivalent with physically pressing `x` for 20 times. The same applies for other operators or motions commands. For example, `10l` is equivalent of pressing `l` for 10 times, making the navigation faster.

Operator `d` does similar things as `x` and `X` but requiring a motion command, for more flexible usage. The motion shall tell *Vim* what to delete/cut.

For example, `dl` deletes to the right, i.e. deletes the current cursor selected character, and automatically select the next character. It is the same as if `x` is pressed. Similarly, `dh` deletes to the left, just like `X`. What if you want to delete 20 characters to the right? You can key in `dl` for 20 times. Or alternatively, just like the case for `x`, you can tell *Vim* to do it by using `20dl`. Or, you can

change the motion, by using `d20l`, where "20l" as a whole plays as the motion of "to the right for 20 characters". Or, you can do a combination by using things like `5d4l`, since $20 = 5 \times 4$. All of the above gives you the same result (they will be a difference in the clipboard if later you want to paste them).

Thanks to the "operator-motion" structure, `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 character of next word. Motion `w` can also be used to navigate in the text. Similarly, `b` directs to the beginning character of the current word (if the cursor is at the middle of the current word) or previous word (if the cursor is already at the beginning of the current word). Thus, `db` can be used to delete word to the left. You can use something like `d10b`, `10db`, `d20w`, `5d4w` to delete multiple words at a time.

When in the middle of a word, `dw` will delete the characters from the current cursor position till 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 of from the middle of the word, you can use `b` first to navigate back to the beginning of the word. Alternatively, use "inner-word" motion `iw` to indicate that you want to delete inner word. When the cursor is at "k" in "Shakespeare", use `diw` to delete the entire word.

So far we have introduced the delete/cut operator `d`, and character motion `h` (left), `l` (right), and also word motion `b` (left), `w` (right). There are similar motions for sentence `(` (previous), `)` (next) and paragraph `{` (previous), `}` (next). Finally, there is the inner-word motion `iw` to indicate the current word of cursor, whichever the cursor is inside the word. Similarly, there are inner-sentence motion `is` and inner-paragraph motion `ip`. There are also inner-quotation motion `i'`, `i"`, `` i` `` and inner-block motion `i(`, `i<`, `i{`, and many more. For example, when cursor is at "A" of "26 April 1564", `di(` will delete everything inside "()", i.e. deleting "bapt. 26 April 1564 - 23 April 1616".

To conclude, the operators and motions so far are listed in Tabs. 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. (Notice that *Vim* marks the end of a sentence using ".", "?" or "!" followed by a space, tab or blank row, and the end of a paragraph by an empty row.)

To change a piece of text, operator `c` is used, followed by its associated motion to indicate the range of text to be changed. The same motions as given in Table 3.4 can be used. Effectively, operator `c` deletes/cut the text indicated by the motion first (just like operator `d`), then switch to insert mode.

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

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

Operator	Description
`x`	Delete/Cut the character at cursor.
`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`, `w`	One word to the previous or next.
`(`, `)`	One sentence to the previous or next.
`{`, `}`	One paragraph to the previous or next.
`iw`, `is`, `ip`	inner-word, inner-sentence, inner-paragraph.
`aw`, `as`, `ap`	a word, a sentence, a paragraph (including the end blank).
`i'`, `i"`, `` i` ``	inner-quotation for different types of quotations.
`i(`, `i<`, `i[`, `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.

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 text at the cursor position, use `p`. No motion is required.

In addition to the motions given in Table 3.4, another commonly used method to navigate to a particular position if text 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 typing `f;` twice.

Search in the Text

It is common that we want to search for a particular keywords or phrase, and highlight all of its appearances in the text. To make the searching result highlighted, add the following line to the user profile at *vimrc*.

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

where `hlsearch` highlights all matching results in the text, and `incsearch` allows highlighting texts while keying in the word or phrase. Each time highlight search is enabled, *Vim* will remember the keywords from the previous search and automatically hight them in the text, which can be confusing sometimes. The command `exec "nohlsearch"` (`exec` command in the user profile make *Vim* execute that command when starting a new session) that comes after `set hlsearch` resolves this issue by forcing *Vim* to clear its memory. Finally, `ignorecase` allows case insensitive while searching.

In normal mode, use `/<keyword>` to search keywords or phrase. With the above setup, search for "he" using `/he` will lead 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 the first appearance, i.e. "he" in "and the world's greatest dramatist". Click `Enter` to quit searching. Now `h`, `j`, `k` and `l` can be used to navigate the cursor again, with the searching result maintain highlighted. Use `n` and `N` to navigate the cursor from the highlighted results. Notice that in this case, `n` and `N` can be used as motion together with delete/cut, change and copy as given in Table 3.3.

To disable the existing searching highlight, either start searching a new

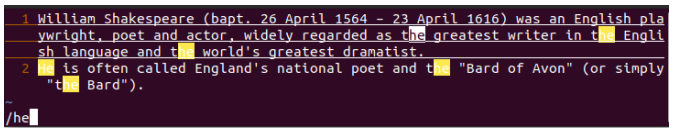

FIGURE 3.4
Search "he" in the piece of text of "William Shakespeare".

keyword, or key in command `:nohlsearch` in normal mode. For convenience, people may prefer to map it with a customized shortcut key as well, for example in *vimrc* key in

```
noremap <Space> :nohlsearch<CR>
```

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

Visual Mode

The use of a mouse makes selecting a block of text very intuitive. As your eyes move across the text, you can start selecting at any specific character or row by click-and-hold the mouse key, and end selecting at any specific character or row by simply letting it go. The selected text will be highlighted, as if the cursor expands from one character to the entire block of text. You can then perform operations such as delete or copy of the selected block of text.

When using *Vim*, mouse is mostly useless. The visual mode of *Vim* provides users a similar experience when selecting a block of text almost as if using a mouse.

It is possible to select multiple rows of text in visual mode, and then apply an operation all at rows simultaneously.

3.2.4 Advanced Interface

"nobreak

3.2.5 Vim Accessories

4

Files Management

CONTENTS

“nobreak

4.1 Filesystem Hierarchy Standard

“nobreak

4.2 File Management

5

Software Management

CONTENTS

“nobreak

5.1 Linux Kernel Management

“nobreak

5.2 General Introduction to Linux Package Management Tools

“nobreak

5.3 Installation of Software

“nobreak

5.4 Software Upgrade

“nobreak

5.5 Uninstallation of Software

6

Process Management

CONTENTS

“nobreak

6.1 General Introduction to Process

“nobreak

6.2 Running Process Management on Linux

Part II

Linux Advanced

7

Linux Administration

CONTENTS

“nobreak

7.1 Quick Installation Guide

“nobreak

7.2 Useful Tools

8

Linux Account Management

CONTENTS

“nobreak

8.1 Quick Installation Guide

“nobreak

8.2 Useful Tools

9

Linux Disk Management

CONTENTS

“nobreak

9.1 Quick Installation Guide

“nobreak

9.2 Useful Tools

10

Advanced Software Management

CONTENTS

“nobreak

10.1 Quick Installation Guide

“nobreak

10.2 Version Control Software and Git

“nobreak

10.2.1 Brief Introduction to Git

“nobreak

10.2.2 Local Repository Operations

“nobreak

10.2.3 Remote Repository Operations

Part III

Linux Server Management

Part IV

Linux Security

Part V

Linux on Cloud

Bibliography