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

General Introduction

1

General Introduction to Linux

CONTENTS

“nobreak

1.1 General Introduction to Operating System

“nobreak

1.2 A Brief History of Linux

“nobreak

1.3 Key Features of Linux

“nobreak

1.3.1 Filesystem Hierarchy

“nobreak

1.3.2 Access Control Lists

2

Linux Installation

CONTENTS

“nobreak

2.1 Different Flavours of Linux

“nobreak

2.2 Linux Installation

“nobreak

2.3 Linux Basic Configuration

“nobreak

2.4 Useful APPs and Tools

Part II

Linux Basics

3

Shells

CONTENTS

“nobreak

3.1 Brief Introduction to Linux Shells

“nobreak

3.2 Basic Grammar

“nobreak

3.3 Useful Commands

4

Text File Editing

CONTENTS

“nobreak

4.1 Text File Editing Environment in Linux

“nobreak

4.2 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, such as by using macros.
- *Vim* can be integrated with third-party tools for useful functions such as browsing project folders.

The above reasons have their point, and it is true *Vim* can be come very powerful and convenient for the user if he is very familiar with it and 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 may require a learning curve for beginners.

In this section, *Vim* is introduced as the text editor that will be used for viewing and editing text files, either being configuration files or programming codes.

4.2.1 General Introduction to Vim

Different from other text editors, *Vim* defines different "modes" during the operation, each mode has some unique features. For example, in the *insert* mode, *Vim* takes in the keyboard inputs and put them into the text file. In this concept, many other editors can be taken as a slim version of *Vim* where there is only one mode, the *insert* mode.

In the case of *Vim*, however, there are other equally useful modes that eventually make it unique and powerful. For example, 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 4.1 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.

TABLE 4.1
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.

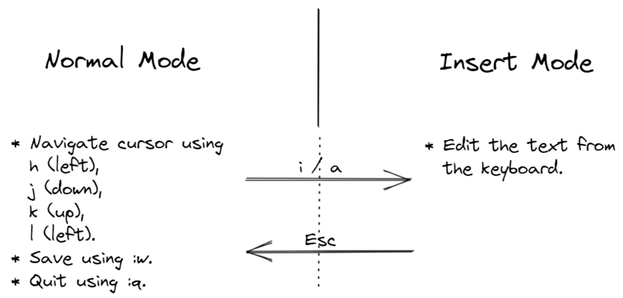


FIGURE 4.1
Mode switching between normal mode and insert mode, and basic functions associated with the modes.

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. 4.1. A flowchart to create/open, edit, save, and quit a text file using the aforementioned commands are given in Fig. 4.2.

4.2.2 Configure Customizable User Profile

With the basic operations introduced in Section 4.2.1, 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 finally useful now.

Before introducing more advanced features of *Vim* for more convenient

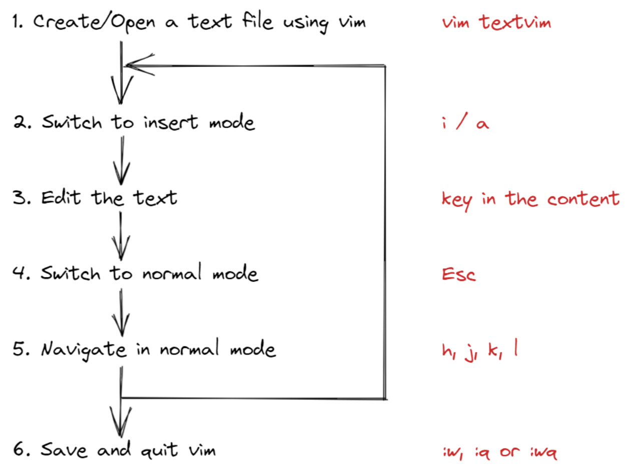


FIGURE 4.2
A flowchart for simple creating, editing and saving of a text file using *Vim*.

user experience, we can now customize 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.

4.2.3 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.

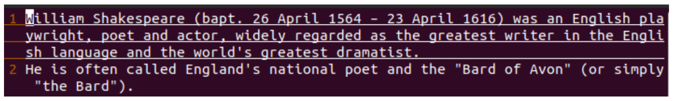
Delete/Cut, Change, Copy and Paste

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 4.3.

> 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



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

FIGURE 4.3
A piece of text of "William Shakespeare", for demonstration.

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 "emulate" doing that 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

TABLE 4.2
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.

"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. 4.2 and 4.3. 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 blank space or tab or line, 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 4.3 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 its associated motion to indicate the range of text. The motions also follow Table 4.3.

TABLE 4.3
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[`, `}`	inner-block for different types of brackets.

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 4.3, 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
```

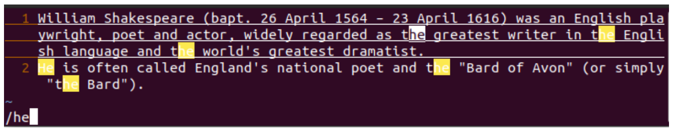
In normal mode, using `f` followed by a character will navigate the cursor to the nearest corresponding character that appears in the text. For example, `fG` will move the cursor to letter “G”. Similarly, `f;` will move the cursor to the “;” between “G” and “H”. Key in `f;` again and the cursor will 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 “hlsearch” 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



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

FIGURE 4.4
Search “he” in the piece of text of “William Shakespeare”.

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. 4.4. From Fig. 4.4, 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 4.2.

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

4.2.4 Advanced Interface

“nobreak

4.2.5 Vim Accessories

5

Files Management

CONTENTS

“nobreak

5.1 Filesystem Hierarchy Standard

“nobreak

5.2 File Management

6

Software Management

CONTENTS

“nobreak

6.1 Linux Kernel Management

“nobreak

6.2 General Introduction to Linux Package Management Tools

“nobreak

6.3 Installation of Software

“nobreak

6.4 Software Upgrade

“nobreak

6.5 Uninstallation of Software

“nobreak

6.6 Software Management Using Git

7

Process Management

CONTENTS

“nobreak

7.1 General Introduction to Process

“nobreak

7.2 Running Process Management on Linux

Part III

Linux Advanced

Part IV

Linux Server Management

Part V

Linux Security

Part VI

Linux on Cloud

Bibliography