From Novice to Expert in Bioinformatics

Linux, Perl and R for Bioinformatics

Shaojun Xie

2017-11-07

Contents

Preface

Who should read this book?

I started to do NGS data analysis as a bench researcher. I started from ZERO. I understand how people feel when they were ZERO. So this book is written for researchers who have no prior knowledge of NGS data analysis. You do NOT need any programming background. If you already have some, that's good. All you need is the strength and the resolution to finish this book.

For those of you who already have some experience but you still don't feel comfortable when dealing with the data, this will be for you. In this book, I will write some skills that I have learned, which will accelerate your analysis.

How to read this book?

This is not a sci-fiction books. This is a technical book. I highly recommend you to practice when you're reading this book.

Don't panic!

The reason why I didn't write in Chinese?

The targets of this book are not only Chinese but also reseasrchers from other countries.

About me

My name is Shaojun Xie. I'm a husband, a father, a son, a brother and a researcher. I started to work on analyzing next generation sequencing (NGS) data in 2010. When I first heard of NGS, I was an undergraduate student at Huazhong Agricultural University in China. When I began to do my PhD in China Agricultural University (CAU), more and more researchers around me started to talk about NGS data. At that time, I had no idea about what NGS was. In the first year of my PhD (2009), my wet experiment was not smooth. I thought I need to make some changes. I think Bioinformatics is a cool area. Then I checked online for job advertisements related to bioinformatics to see what the employers need. I figured out that Linux, Perl and R were the most important things needed. After that I started to read the book (Perl: How to program). Also I started to use Ubuntu which is a variant of Linux. I can understand the contents in the book. But I didn't know what I should do or what I can do with the knowledge I learned from the book.

A few months later (September in 2010), I joined a new lab and started to analyze RNA-seq data. The results of that work was later published in PNAS. I'm one of the co-first authors. Although I was responsible for parts of the project, it helped me a lot. I learned how to use Perl and R in the data analysis. That project opened the door for me. After that I worked on several other projects. Meanwhile, we had several

collaborations with other labs. These different projects broadened my horizon in the area of NGS data analysis. Some of the projects were published in high profile journals, like Genome Research, Plant Cell, etc.

In June 2014, I graduated from China Agricultural University. One of my collaborators recommended me to another lab. I got a position as a research associate. This position also provided me an opportunity to be a visiting scholar at Purdue University. I got Purdue University in August 2014. I worked on several nice projects. Some of them also got published in nice journals. One year later, I started to look for a job as a Bioinformatician. Bioinformatics Core was just recruiting bioinformaticians at that time. I got an offer and then joined the core in April 2014. Currently I'm a Bioinformatician in Bioinformatics Core at Purdue University.

Acknowledgement

First, I'd like to thank my family (my wife, our parents and my two daguthers: Angela and Sarah). Without them, I wouldn't have the time and the confidence to write this book.

I want to thank my current colleages and previous colleages, as well as my superwisors. I learned a lot from them.

Then I'd thank the author of bookdown (Xie (2016)). For a long time I tried to have a nice platform that can help me write this book. Finally, I found bookdown is the best one for me.

Last but not the least, there are many people in my life. I'd like to thank them for the help.

Disclaimer

- The opinions expressed in this book represent my own and not those of my current or previous employers.

https://www.justanswer.com/immigration-law/5f14g-someone-write-book-staying-h1b-h4-f1-visa-earn.html

Hello, welcome to JustAnswer, my name isXXXXX you for this opportunity to answer your questions. Royalty payments are not considered unlawful employment. It is the sale of an intellectual property and as such is not considered providing goods or services in violation of the employment laws. Judith

Chapter 1

Why Linux?

1.1 What is Linux

Before the creation of Linux, Unix was developed by AT&T Bell Labs in the 1960's. It's an operating system. Before the creation of Linux, and before the rise of Windows, the computing world was dominated by Unix (from web). After many years of evolution, Linux was created in early 1990's.

In case you don't know, Mac OS X is also a certified Unix operating system. So most of the Linux skills are applied in Mac OS X.

Linux is a clone of the operating system Unix, written from scratch by Linus Torvalds (Figure 1.1) with assistance from a loosely-knit team of hackers across the Net. It aims towards POSIX and Single UNIX Specification compliance (Torvalds (2015)).

Linux is a clone of the operating system Unix, written from scratch by Linus Torvalds (Figure 1.2) with assistance from a loosely-knit team of hackers across the Net. It aims towards POSIX and Single UNIX Specification compliance (Torvalds (2015)).

It has all the features you would expect in a modern fully-fledged Unix, including true multitasking, virtual memory, shared libraries, demand loading, shared copy-on-write executables, proper memory management, and multistack networking including IPv4 and IPv6. It is distributed under the GNU General Public (Torvalds, 2015).

Maybe it's hard to understand what Linux or to remember the sentences mentioned above. Just know Linux is an operating system like Windows. This is enough for you to start out.

1.2 Linux for bioinformatics

For analysis of NGS data, a large amount of software were developed for using under Linux environment. Among them, a large proportion can be only used under Linux environment.

- Easy to build simple pipelines (awk, bash, piping, bash redirection, texttools)
- Simple to install and use software development tools
- Multiple versions of a program can be installed by the user himself and switched on/off with sourcing some scripts without being administrator
- A lot of good scientific software is written in a non portable way for linux/unix (almost all short read aligners, samtools). This makes it necessary to use Unix for genomics.
- Ability to perform analyses on computer clusters (important for big/long computational jobs)

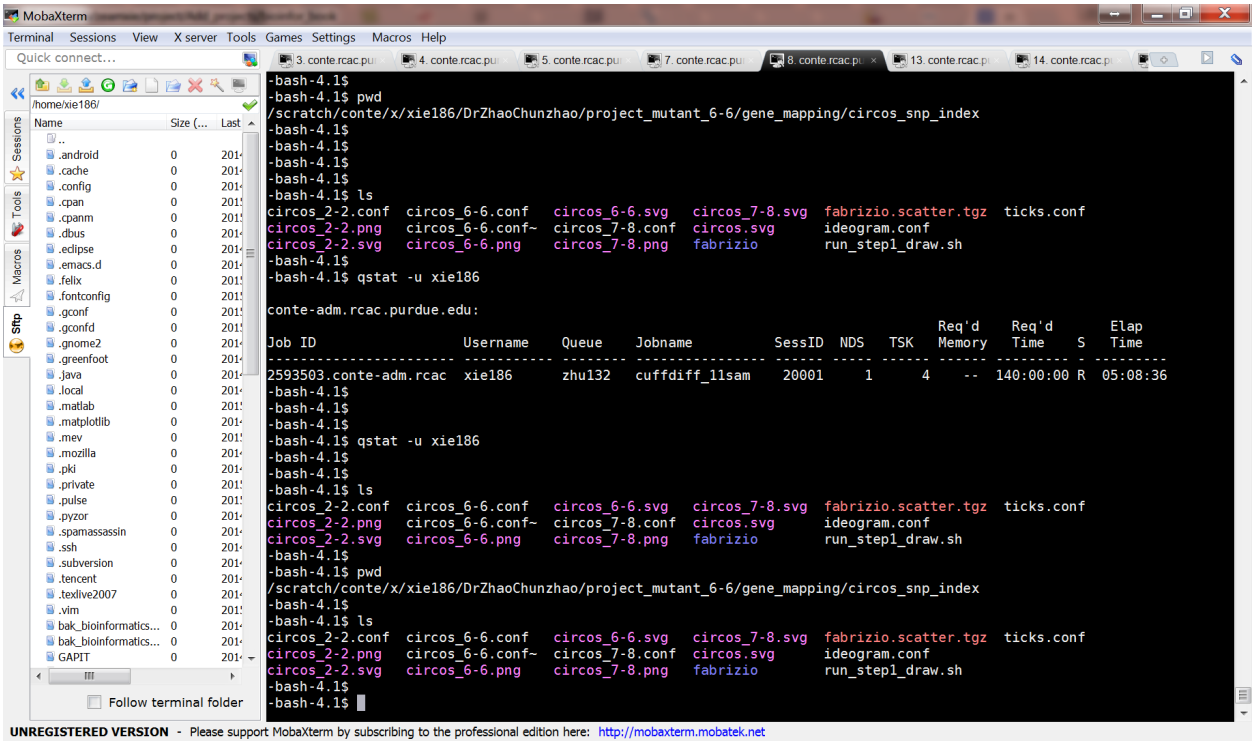


Figure 1.1: An example of Linux terminal.

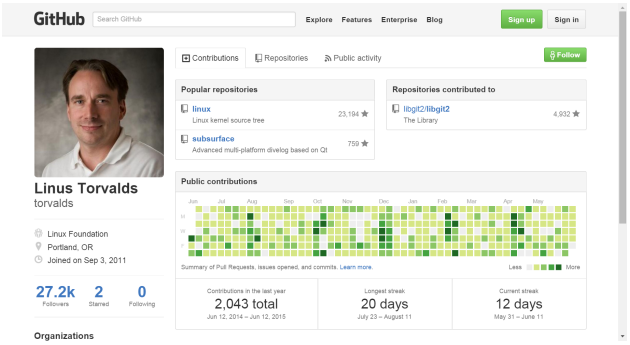


Figure 1.2: Linus Torvalds on GitHub

Chapter 2

Connecting to Linux

2.1 User interfaces

As an operating systemsm, Linux comes with two types of user interfaces: Graphical User Interface (GUI) and command line interface (shell).

GUI means there will be window, buttons, menus, etc. The most popular system with GUI is Windows system (Figure 2.1).

Command line interfaces means that you need to type the command line yourself. Usually the results will be displayed as text (Figure 2.2).

In Bioinformatics analysis, usually you won't operate directly on the physical machine of the Linux server. Usually you need to connect to the Linux server via a tool, like Putty, Mobaxterm, etc.

2.2 How to connect

If you want to connect to a Linux server, what you need to know first is:

1) IP address of your Linux server;

2) User name and password of your account;

If you are a Mac OS X user, you can connect to a Linux server by using `Terminal`, a console program included with the operating system.

For Windows users, I would recommend MobaXterm for remote connection. MobatXterm is an excellent toolbox for remote connection from Windows system. It comes with an X11 server and provides many networking tools and tabbed SSH. It has all the essential UNIX commands in a single portable executable file.

Here I show one example of what you should do When you first open Mobaxterm. You need to follow the numbers in Figure 2.3: click `Session`; then click `SSH`; type the IP or name address of the remote host, check `Specify username` if you need; click `OK`.

Then you need to type the password. It's OK that you don't see anything when you're typing (Figure 2.4). Then click `Enter` on the keyboard.For the first time of log-in, you'll be asked to whether to save the password or not. If you say click `Yes`, you won't need to type the password again next time.

If you can find a Linux server, it'll be very good. If NOT, here I provided a guest account for you. Here are the user name and password:

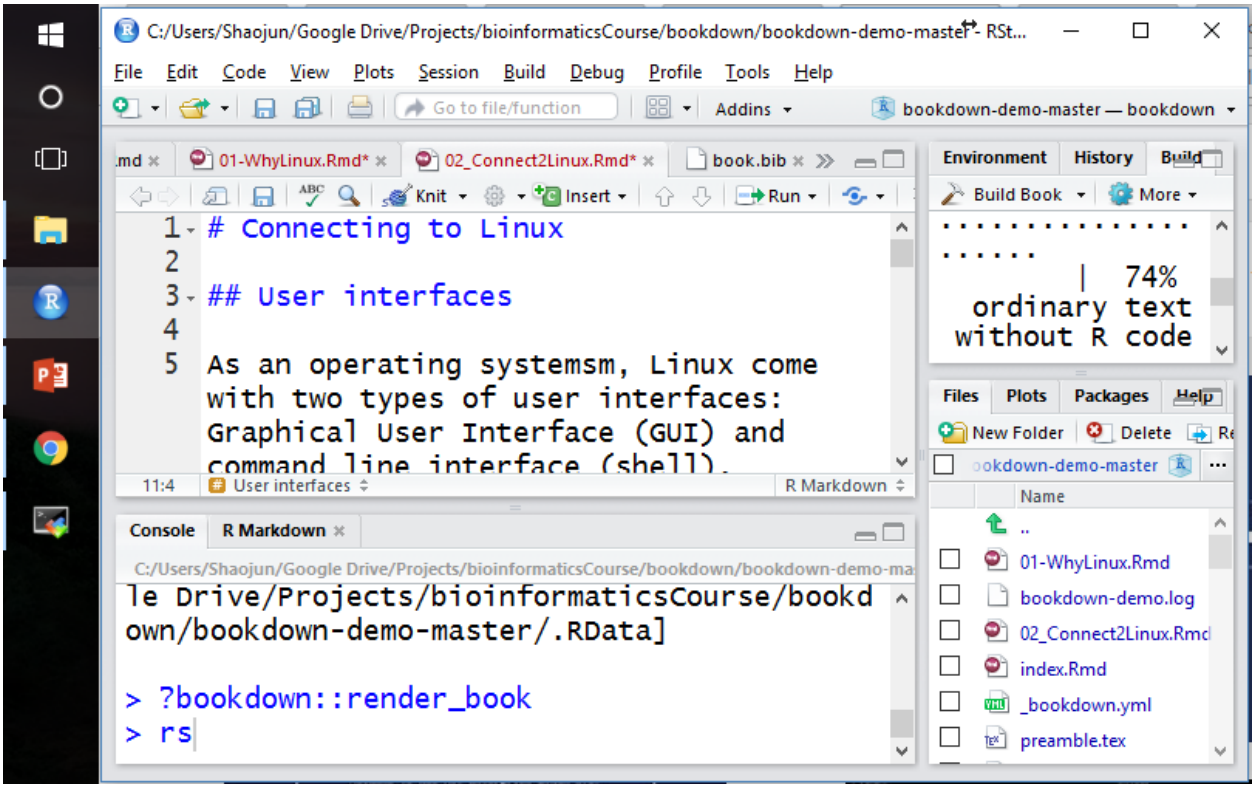


Figure 2.1: Windows GUI.

```
IP address: 198.211.107.37
User name: guest4bioinfo
Password: nobigfile
```

As you can tell from the password, please do NOT upload BIG files (bigger than 2 MB).

2.2.1 Set up Linux Lab Environment

One way to get access to a Linux system is to take advantage of Red Hat Enterprise Linux delivered by Amazon EC2 (Elastic Compute Cloud). Red Hat and Amazon Web Services collaborate to provide official Red Hat Enterprise Linux licensed images through Amazon's on-demand public cloud service at free or low cost.

The guided exercises and labs for this course were written assuming that you will set up an account with Amazon Web Services and use it to start a single, simple system running Red Hat Enterprise Linux 7. You will connect to that system securely over the internet and use it to practice commands.

At the time of writing, Amazon Web Services provides an AWS Free Tier offering, which gives new users free access to certain sizes of cloud instances and operating environments (including Red Hat Enterprise Linux 7) for up to 750 hours per month, for 12 months.

Install `samtools` without root previledges

If you are not eligible for AWS Free Tier or have used up your free tier eligibility, a t2.micro-sized instance (cloud computer) running the same software, is currently estimated to cost between US$0.072 and US$0.08 per hour of compute time, depending on the data center in which the instance is started.

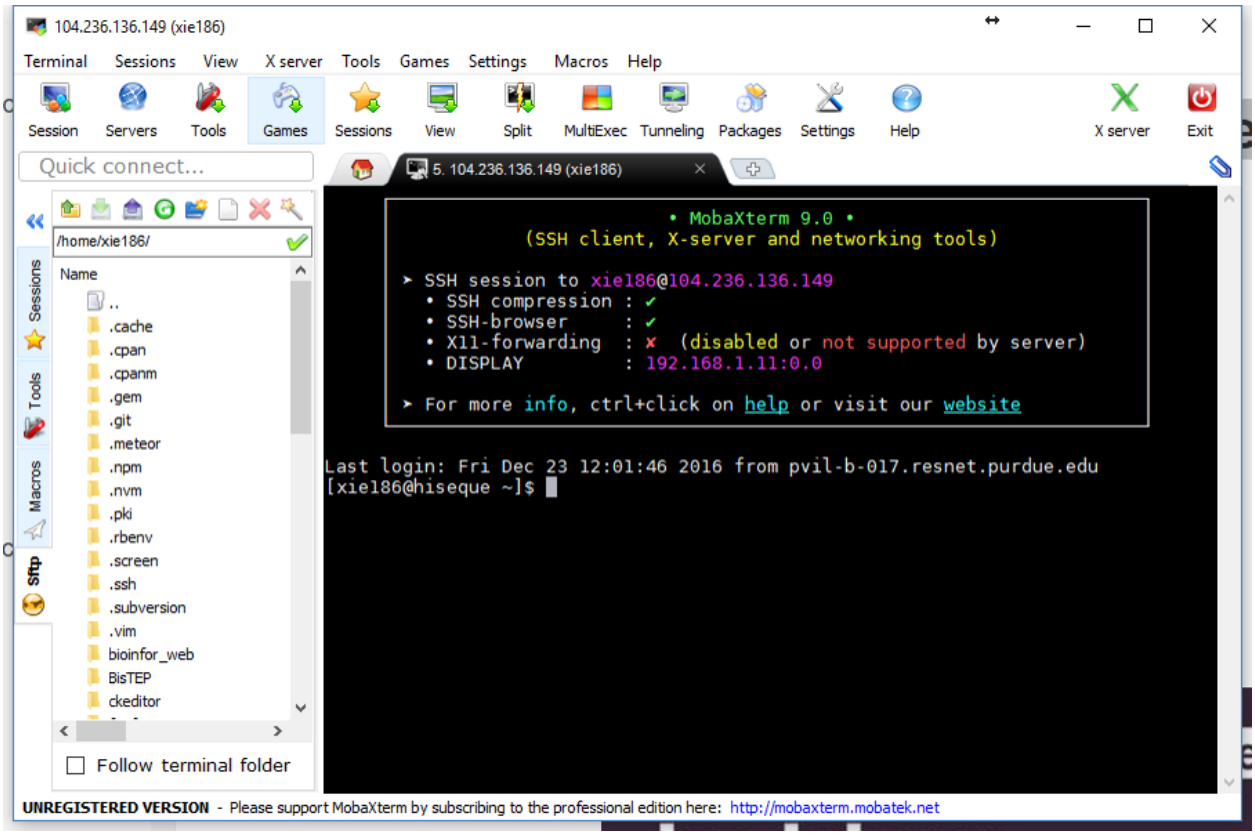


Figure 2.2: An example of Linux terminal.

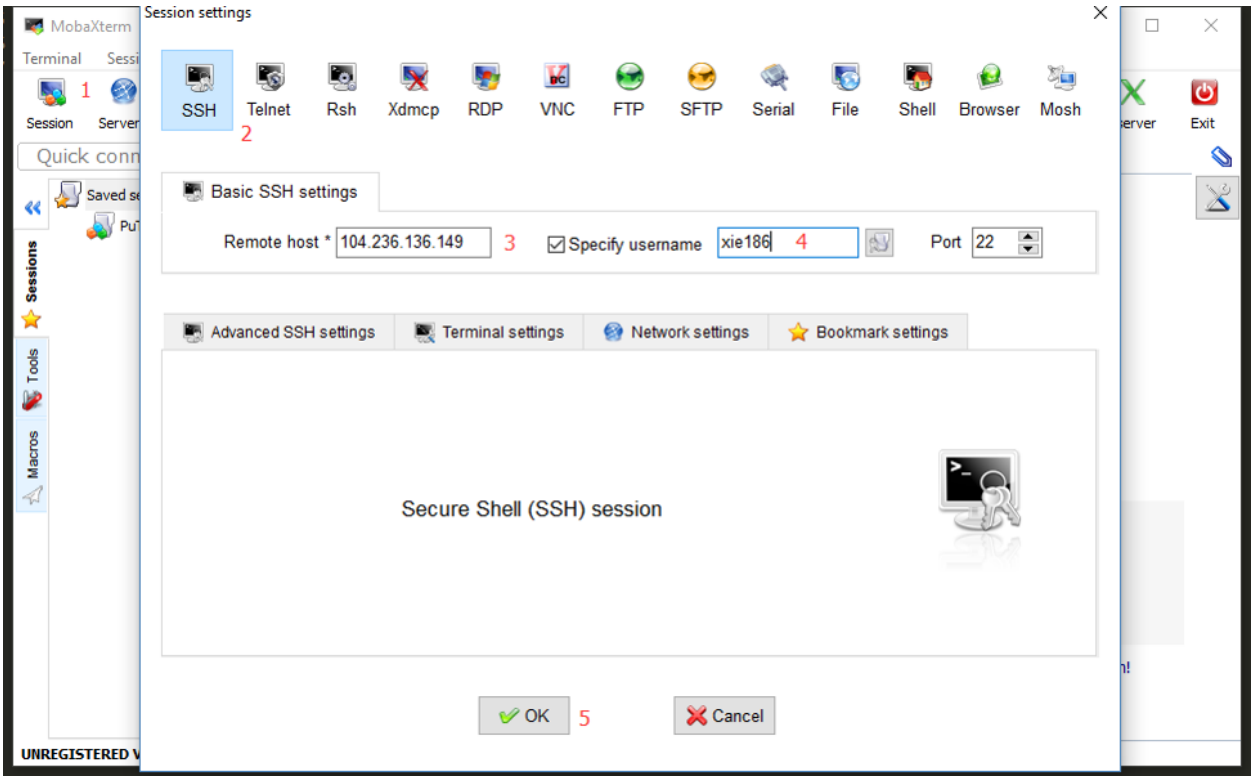


Figure 2.3: First open of Mobaxterm.

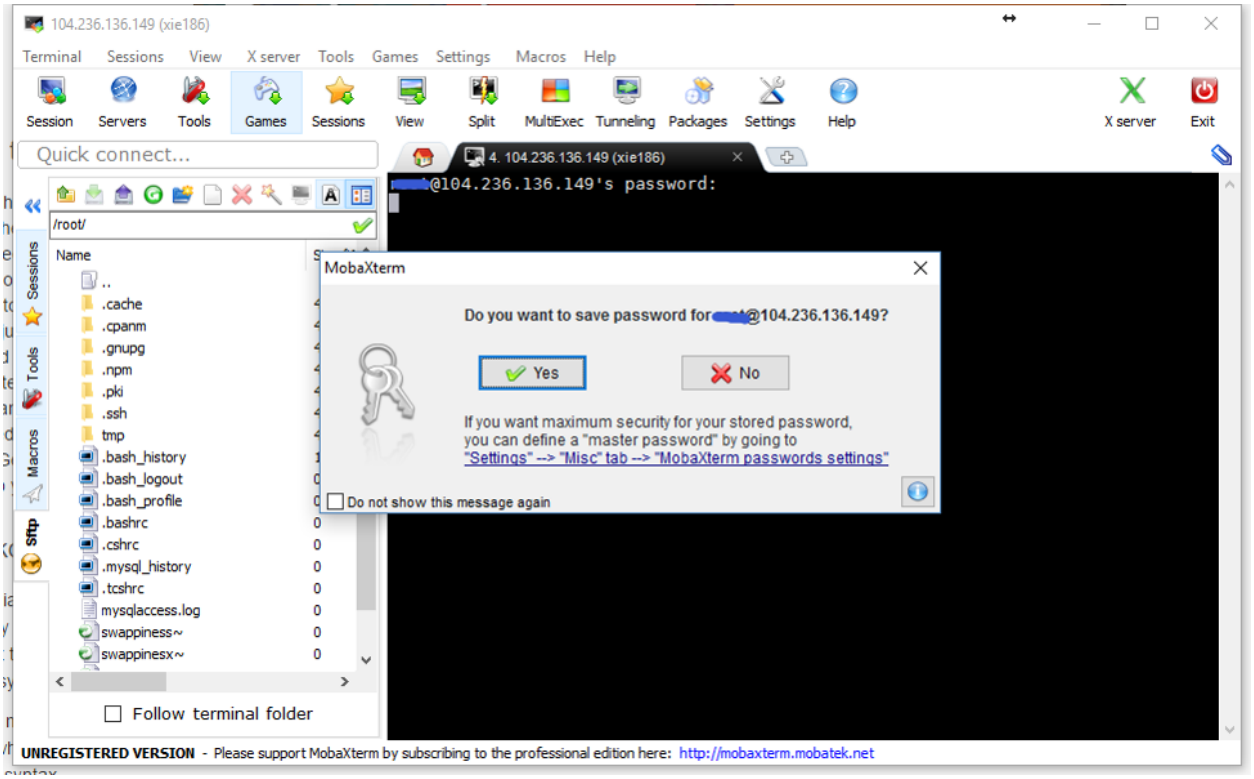


Figure 2.4: Type password and save it in Mobaxterm.

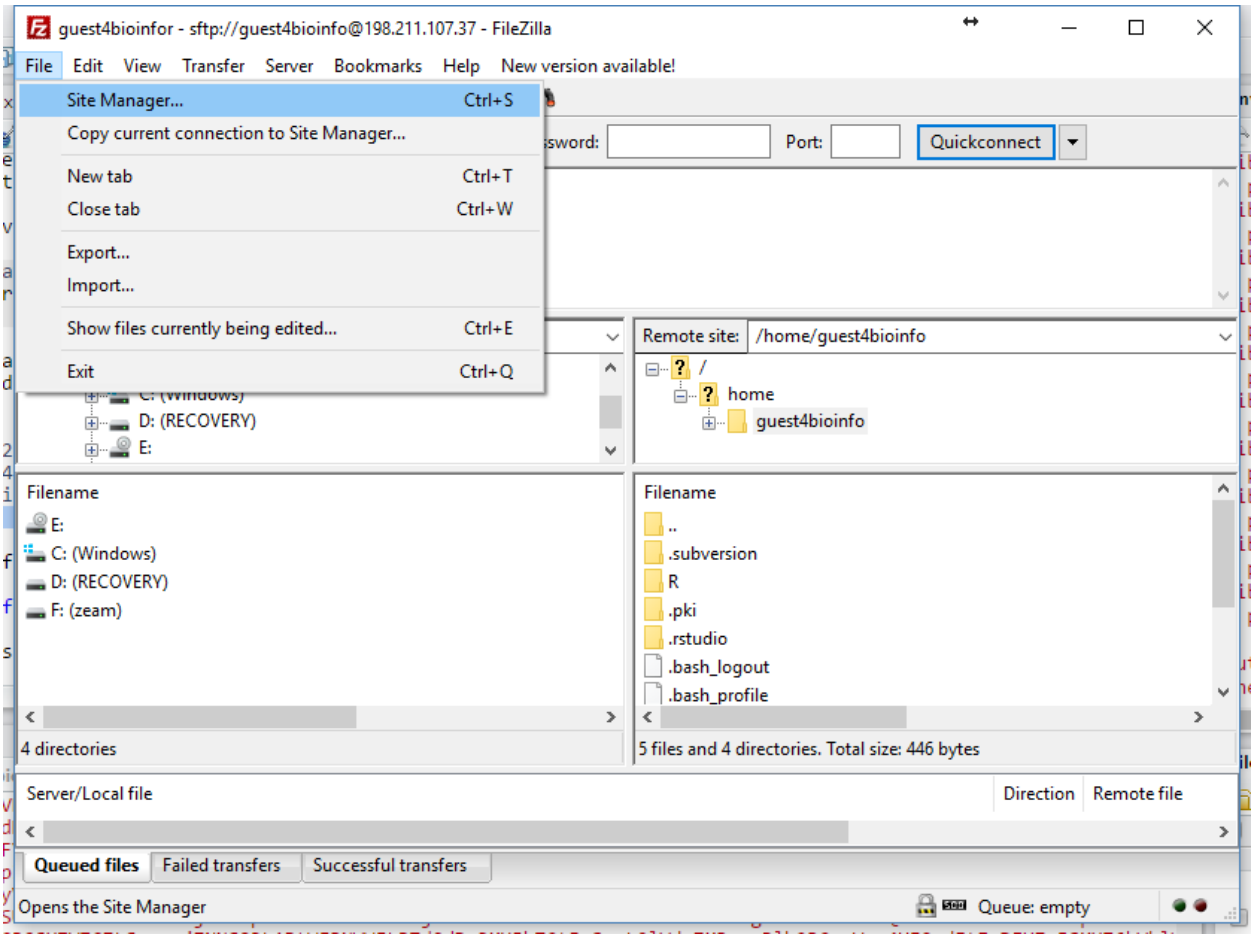


Figure 2.5: (#fig:filezilla_screenshot1)Windows GUI.

> To conserve compute time and any costs, make sure you shut down your cloud instance when you are not using it, and terminate (delete) it when you are completely finished with it.

2.3 Transferring files between local computer and Linux server

To transfer files between local computer and Linux sever, there are two options: 1) GUI application and 2) command line.

- Open FileZilla and then click `File` -> `Site Manager`.

GUI means there will be window, buttons, menus, etc. The most popular system with GUI is Windows system (Figure @ref(fig:filezilla_screenshot1)).

(ref:filezilla_screenshot1) FileZilla application.

2.3.1 Use command line tools

`rsync` compares the files at each end and transfers only the changed parts of changed files. When you transfer files the first timeo it behaves pretty much like scp, but for a second transfer, where most files are

unchanged, it will push a lot less data than scp. It's also a convenient way to restart failed transfers - you just reissue the same command and it will pick up where it left off the time before, whereas scp will start again from scratch.

2.3.1.1 Copy files using `rsync`

2.3.1.2 Copying Files with `scp`

The command `scp` is short for secure copy. It can be used to copy files between hosts on a network. It uses ssh(1) for data transfer, and uses the same authentication and provides the same security as ssh(1).`Scp` will ask for passwords or passphrases if they are needed for authentication.

File names may contain a user and host specification to indicate that the file is to be copied to/from that host. Local file names can be made explicit using absolute or relative pathnames to avoid scp treating file names containing ':' as host specifiers. Copies between two remote hosts are also permitted.

```
# Copy the file test.pl on 198.211.107.37 to the current directory.
scp guest4bioinfor@198.211.107.37:~/test.pl ./
```

To copy files from a server to a client, you need to know where the files are located on the server. For example, to copy a single file `~/test.pl` from the server with IP address of 198.211.107.37 to the current directory.

```
# Copy the file test.pl in the current directory to  198.211.107.37
scp ./test.pl guest4bioinfor@198.211.107.37:~/
```

To copy files from a client to a server, you need to know where the files you want to put on the server. For example, to copy a single file `test.pl` from the current folder to the HOME folder of the server with IP address of 198.211.107.37.

If you want to copy an entire directory recursively, you can use `-r` argument. See the example below:

```
# Copy the file test.pl in the current directory to  198.211.107.37
scp -r guest4bioinfor@198.211.107.37:~/bioinfo/ ./
```

2.3.2 Download files

`wget <url>`

Resume

`wget -c <url>`

Reference:

RH066x Fundamentals of Red Hat Enterprise Linux on edX

Chapter 3

File system and commands of Linux

3.1 Path

To understand Linux file system, you can image it as a tree structure (Figure 3.1).

In Linux, a path is a unique location of a file or a directory in the file system.

For convenience, Linux file system is usually thought of in a tree structure. On a standard Linux system you will find the layout generally follows the scheme presented below.

The tree of the file system starts at the trunk or slash, indicated by a forward slash (/). This directory, containing all underlying directories and files, is also called the root directory or "the root" of the file system.

3.1.1 Relative and absolute path

- **Absolute path**

An absolute path is defined as the location of a file or directory from the root directory(/). An absolute path starts from the `root` of the tree (`/`).

Here are some examples:

```
/home/xie186
/home/xie186/perl5
```

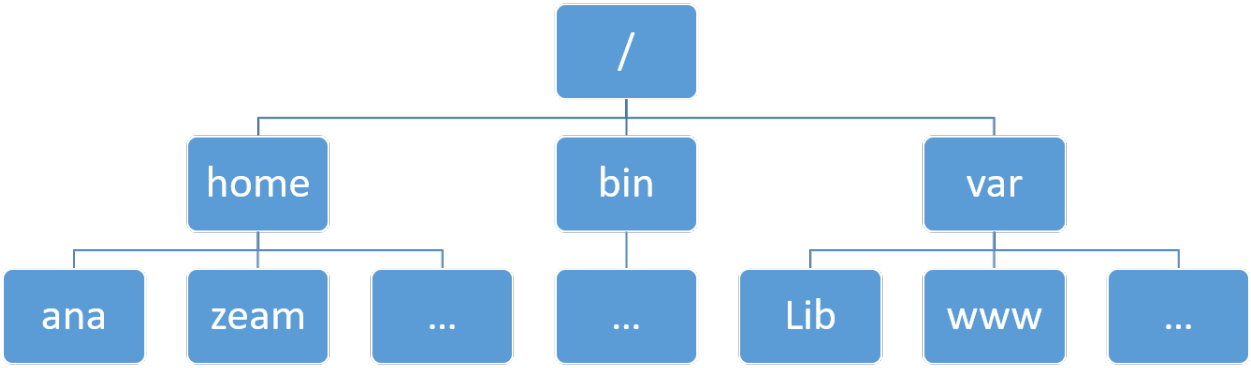


Figure 3.1: Tree structure of Linux system.

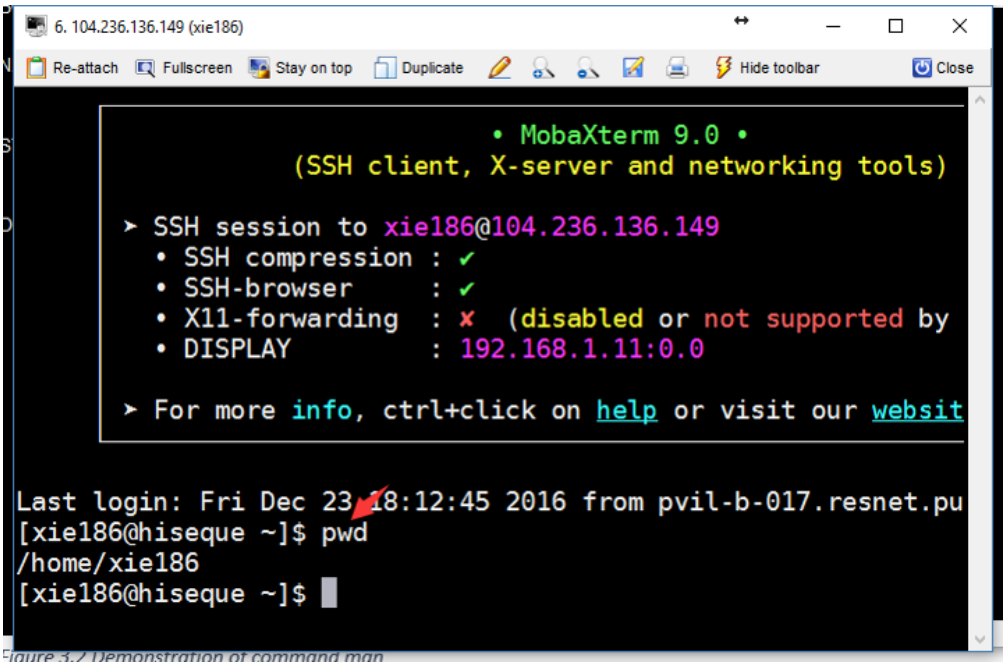


Figure 3.2: ref:linuxCMDpwd

- **Relative path**

Relative path is a path related to the present working directory.

```
data/sample1/
../doc/
```

If you want to get the **absolute path** based on **relative path**, you can use `readlink` with parameter `-f`:

```
pwd
readlink -f ../
```

```
## /home/xie186/bookdown/novice2expert4bioinformatics
## /home/xie186/bookdown
```

3.2 Surfing in Linux file system

Once we enter into a Linux file system, we need to 1) know where we are; 2) how to get where we want; 3) how to know what files or directories we have in a particular path.

3.2.1 Command `pwd`

In order to know where we are, we need to use `pwd` command. The command `pwd` is short for "print name of current/working directory". It will return the full path of current directory.

Command pwd is almost always used by itself. This means you only need to type pwd and press ENTER (Figure 3.2).

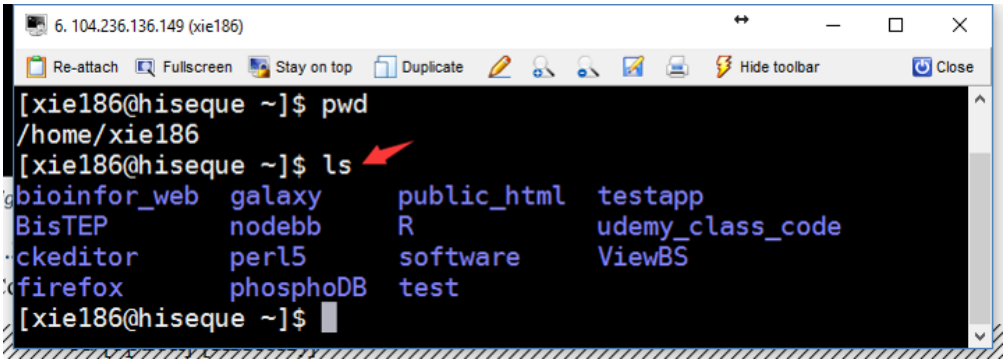


Figure 3.3: ref:linuxCMDls

Table 3.1: Shortcuts of path.

Path	Shortcuts	Description
Single dot	.	The current folder
Double dots	..	The folder above the current folder
Tilde character	~	Home directory (normally the directory:/home/my_login_name)

3.2.2 Comand `ls`

After you know where you are, then you want to know what you have in that directory, we can use command `ls` to list directory contents (Figure 3.3). Its syntax is:

```
ls [option]... [file]...
```

`ls` with no option will list files and directories in bare format. Bare format means the detailed information (type, size, modified date and time, permissions and links etc) won't be viewed. When you use `ls` by itself (Figure 3.3), it will list files and directories in the current directory.

3.2.3 Command `cd`

Command cd is used to change the current directory. It's syntax is:

```
cd [option] [directory]
```

Unlike `pwd`, when you use `cd` you usually need to provide the path (either absolute or relative path) which we want to enter.

If we didn't provide any path information, we will change to home directory by default.

3.3 Path shortcuts

In Linux, there are three commonly used path shortpaths (Table 3.1).

Here are some examples:

```
###
cd ~
pwd
ls
```

```
/home/xie186
bak
bin
bookdown
cpanm
curl-7.55.0
curl-7.55.0.tar.gz
github_repo
meteor
meteor-window-manager
perl5
R
software
ViewBS
```

```
##

ls ./
```

```
01-WhyLinux.Rmd
02_Connect2Linux.Rmd
03_FileSystemLinux.Rmd
04_Linux_FilteringOutputandFindingThings.Rmd
05-TextEditorInLinux.Rmd
06_InstallationOfSoftwareInLinux.Rmd
07_AchivingAndCompressingFiles.Rmd
07-FirstPerlProgram.Rmd
07-PerlArrat.Rmd
08-PerlVariableOperator.Rmd
09_PerlControlStructure.Rmd
09_StringManipulationRegExp.Rmd
10_PerlInputOutput.Rmd
10_PerlModules.Rmd
10_Rbasics.Rmd
10_SimpleGraphR.Rmd
112_GenerateHeatmap.Rmd
11_FigureManuscript.Rmd
11_GenomeSequence.Rmd
11_IntronNGS.Rmd
11_practicalPerlProgram.Rmd
11_StringManipulationRegExp.Rmd
12_RNA-seq.Rmd
13_BSseq.Rmd
14_WhatLinuxShellIS.Rmd
18_CapstoneProject.Rmd
18_Good_resource.Rmd
19_Python.Rmd
20_R_intro.Rmd
21_Reference.Rmd
bioinfBookXIE186.aux
bioinfBookXIE186_files
bioinfBookXIE186.log
bioinfBookXIE186.Rmd
```

```
_book
book.bib
bookdown-demo_files
bookdown-demo.log
bookdown-demo.Rproj
_bookdown_files
_bookdown.yml
_build.sh
code_perl
code_python
code_R
data
_deploy.sh
DESCRIPTION
figures
images
index.Rmd
LICENSE
_output.yml
packages.bib
preamble_bak.tex
preamble.tex
README.md
rpres.css
style.css
tables
toc.css
```

```
##
pwd
cd ../
pwd
cd ./
pwd
```

```
/home/xie186/bookdown/novice2expert4bioinformatics
/home/xie186/bookdown
/home/xie186/bookdown
```

Each directory has two entries in it at the start, with names . (a link to itself) and .. (a link to its parent directory). The exception, of course, is the root directory, where the .. directory also refers to the root directory.

3.4 Manipulations of files and directories

In Linux, manipulations of files and directories are the most frequent work. In this section, you will learn how to copy, rename, remove, and create files and directories.

3.4.1 Command `cp`

In Linux, command `cp` can help you copy files and directories into a target directory.

3.4.2 Command `mv`

The command `mv` is short for move (or rename) files.

3.4.2.1 Move one file

Here is one common example of `mv`.

```
mv file1 directory1/
```

3.4.2.2 Move multiple files into a directory

```
mv file1 file2 file3 target_direcotry/
```

3.4.2.3 Move a directory

```
mv dir1
```

3.4.2.4 Rename a file or a directory

3.4.3 Command `mkdir`

Command `mkdir` is short for make directory.

The syntax is shown as below:

`mkdir [OPTION ...] DIRECTORY ...`

```
mkdir directory
```

Multiple directories can be specified when calling `mkdir`.

```
mkdir directory1 directory2
```

3.4.3.1 How to create a

`mkdir -p foo/bar/baz`

How to defining complex directory trees with one command

`mkdir -p project/{software,results,doc/{html,info,pdf},scripts}`

This will create a direcotry trees as shown below:

```
$ tree project/
project/
   doc
       html
       info
       pdf
   results
   scripts
   software

7 directories, 0 files
```

The command line above will directories `foo`, `foo/bar`, and `foo/bar/baz` if they don't exist.

3.5 Viewing text files in Linux

3.5.1 Command `cat`

The command `cat` is short for concatenate files and print on the standard output.

The syntax is shown as below:

`cat [OPTION]... [FILE]...`

For small text file, `cat` can be used to view the files on the standard output.

```
cat data/testdata4linux_cmd.txt
```

```
## gene1
## gene2
## gene3
## gene4
## gene5
## gene6
## gene7
## gene8
## gene9
## gene10
## gene11
## gene12
## gene13
## gene14
## gene15
## gene16
```

You can also use `cat` to merge two text files.

```
cat file1 file2 > merged_file
```

3.5.2 Command `more` and `less`

The command `more` is old utility. When the text passed to it is too large to fit on one screen, it pages it. You can scroll down but not up.

The syntaxt of `more` is shown below:

```
more [options] file [...]
```

The command `less` was written by a man who was fed up with more's inability to scroll backwards through a file. He turned less into an open source project and over time, various individuals added new features to it. less is massive now. That's why some small embedded systems have more but not less. For comparison, less's source is over 27000 lines long. more implementations are generally only a little over 2000 lines long.

The syntaxt of `less` is shown below:

```
less [options] file [...]
```

3.5.3 Command `head` and `tail`

The command `head` is used to output the first part of files. By default, it outputs the first 10 lines of the file.

```
head [OPTION]... [FILE]...
```

Here is an exmaple of printing the first 5 files of the file:

```
head -n 5 code_perl/variable_assign.pl
```

```
## #!/usr/bin/perl
## use warnings;
## use strict;
##
## #assign two strings to two variables
```

In fact, the letter n does not even need to be used at all. Just the hyphen and the integer (with no intervening space) are sufficient to tell head how many lines to return. Thus, the following would produce the same result as the above commands:

```
head -5 data/testdata4linux_cmd.txt
```

```
## gene1
## gene2
## gene3
## gene4
## gene5
```

The command `tail` is used to output the last part of files. By default, it prints the last 10 lines of the file to standard output.

The syntax is shown below:

```
tail [OPTION]... [FILE]...
```

Here is an exmaple of printing the last 5 files of the file:

```
tail -5 data/testdata4linux_cmd.txt
```

```
## gene12
## gene13
## gene14
## gene15
## gene16
```

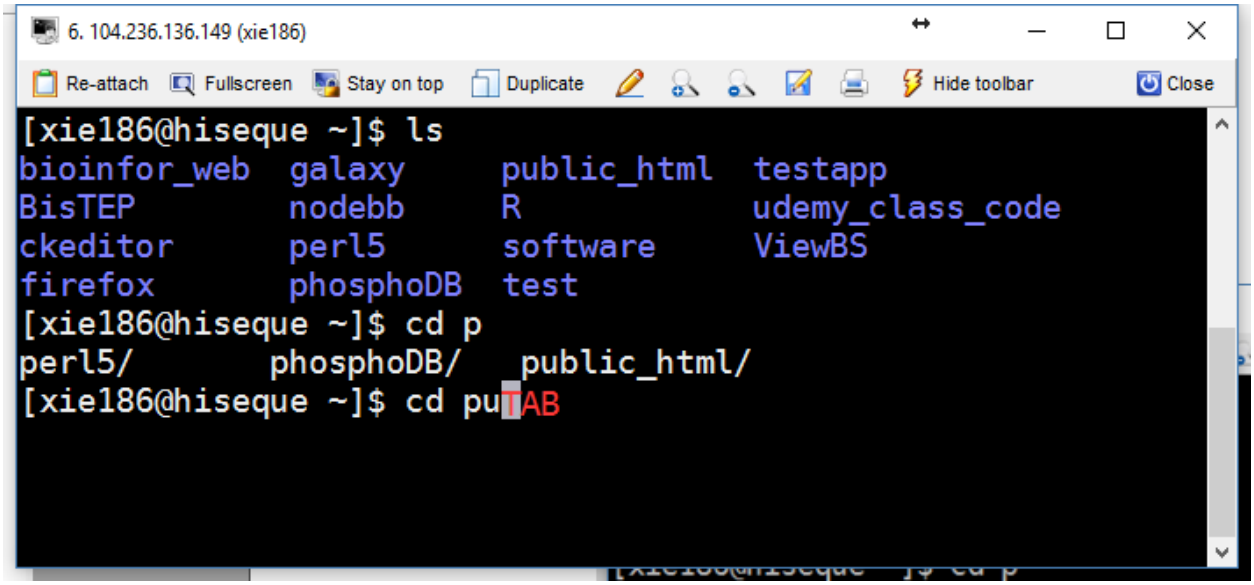


Figure 3.4: Demonstration of programmable completion feature.

To view lines from a specific point in a file, you can use `-n +NUMBER` with the `tail` command. For example, here is an example of viewing the file from the 2nd line of the line.

```
tail -n +2 data/testdata4linux_cmd.txt
```

```
## gene2
## gene3
## gene4
## gene5
## gene6
## gene7
## gene8
## gene9
## gene10
## gene11
## gene12
## gene13
## gene14
## gene15
## gene16
```

3.5.4 Auto-completion

In most Shell environment, programmable completion feature will also improve your speed of typing. It permits typing a partial name of command or a partial file (or directory), then pressing `TAB` key to auto-complete the command (Figure 3.4). If there are more than one possible completions, then TAB will list all of them (Figure 3.4).

3.6 Understand standard input and stardard output

In the Linux environment, input and output is distributed across three streams: standard input (STDIN), standard output (STDOUT), standard error (STDERR). These three streams are also numbered: STDIN (0), STDOUT (1), STDERR (2).

3.6.1 STDIN

… The standard input stream typically carries data from a user to a program. Programs that expect standard input usually receive input from a device, such as a keyboard. Standard input is terminated by reaching EOF (end-of-file). As described by its name, EOF indicates that there is no more data to be read.

To see standard input in action, run the cat program. Cat stands for concatenate, which means to link or combine something. It is commonly used to combine the contents of two files. When run on its own, cat opens a looping prompt. …

```
tail
1
2
3
`CTRL+D`
1
2
3
```

3.6.2 STDOUT

Data that is generated by a program will be written by STDOUT. If the STDOUT is not redirected, it will output the data on to the terminal.

```
stdout="Hello world"
echo $stdout
```

```
## Hello world
```

The STDOUT can be redirected to a file. See the example below:

```
stdout="Hello world"
echo $stdout > data/test_output.txt
# cat the data
cat data/test_output.txt
```

```
## Hello world
```

3.6.3 STDERR

During a program's execution, some errors may be generated when the program fails at some parts. STDERR will help you write the errors. By default, the STDERR will be outputed onto the terminal.

Here is an example of STDERR

```
ls NOTAFILE
```

```
## ls: cannot access NOTAFILE: No such file or directory
```

3.7 Auto-completion

A handy autocomplete feature also exists. Type one or more letters, press the Tab key twice, and then a list of functions starting with these letters appears. For example: type so, press the Tab key twice, and then you get the list as: sort sort! sortby sortby! sortcols sortperm sortrows.

3.8

Filtering Output and Finding Things

https://www.youtube.com/playlist?list=PLtK75qxsQaMLZSo7KL-PmiRarU7hrpnwK

3.9 Check you job status

```
$ ps aux
USER       PID  %CPU %MEM  VSZ RSS     TTY   STAT START   TIME COMMAND
timothy  29217  0.0  0.0 11916 4560 pts/21   S+   08:15   0:00 pine
root     29505  0.0  0.0 38196 2728 ?        Ss   Mar07   0:00 sshd: can [priv]
can      29529  0.0  0.0 38332 1904 ?        S    Mar07   0:00 sshd: can@notty
```

USER = user owning the process PID = process ID of the process %CPU = It is the CPU time used divided by the time the process has been running. %MEM = ratio of the process's resident set size to the physical memory on the machine VSZ = virtual memory usage of entire process (in KiB) RSS = resident set size, the non-swapped physical memory that a task has used (in KiB) TTY = controlling tty (terminal) STAT = multi-character process state START = starting time or date of the process TIME = cumulative CPU time COMMAND = command with all its arguments

3.9.1 List files bigger than filesize specified

```
# To find files larger than 100MB:
find . -type f -size +100M
# If you want the current dir only:
find . -maxdepth 1 -type f -size +100M
```

References:

https://superuser.com/questions/117913/ps-aux-output-meaning

Chapter 4

Text editor in Linux

In Linux, we sometimes need to create or edit a text file like writing a new R program. So we need to use text editor.

As a newbie, someone would prefer a basic, GUI-based text editor with menus and traditional CUA key bindings. Here we recommend Sublime and Notepad++.

But GUI-based text editor is not always available in Linux. A powerful screen text editor vi (pronounced "vee-eye") is available on nearly all Linux system. We highly recommend vi as a text editor, because something we'll have to edit a text file on a system without a friendlier text editor. Once we get familiar with vi, we'll find that it's very fast and powerful.

But remember, it's OK if you think this part is too difficult at the beginning. You can use either Sublime or Notepad++. If you are connecting to a Linux system without Sublime and Notepad++, you can write the file in a local computer and then upload the file onto Linux system (See).

4.1 Basic vi skills

As vi uses a lot of combination of keystrokes, it may be not easy for newbies to remember all the combinations in one fell swoop. Considering this, we'll first introduce the basic skills someone needs to know to use vi. We need to first understand how three modes of vi work and then try to remember a few basic vi commonds. Then we can use these skills to write Perl or R scripts in the following chaptors for Perl and R (Figure 4.1).

4.2 An example for using editor R

```
qnorm(.975)
```

```
## [1] 1.959964
```

```
xval<-seq(-3.2,3.2, length=1000)
yval<-dnorm(xval)
plot(xval, yval, type="l",axes=T,lwd=3,xlab="",ylab="")
x<-seq(qnorm(.975), 3.2, length = 100)
polygon(c(x,rev(x)), c(dnorm(x), rep(0,length(x))), col="salmon")
text(mean(x),mean(dnorm(x))+0.02, "2.5%", cex=2)
```

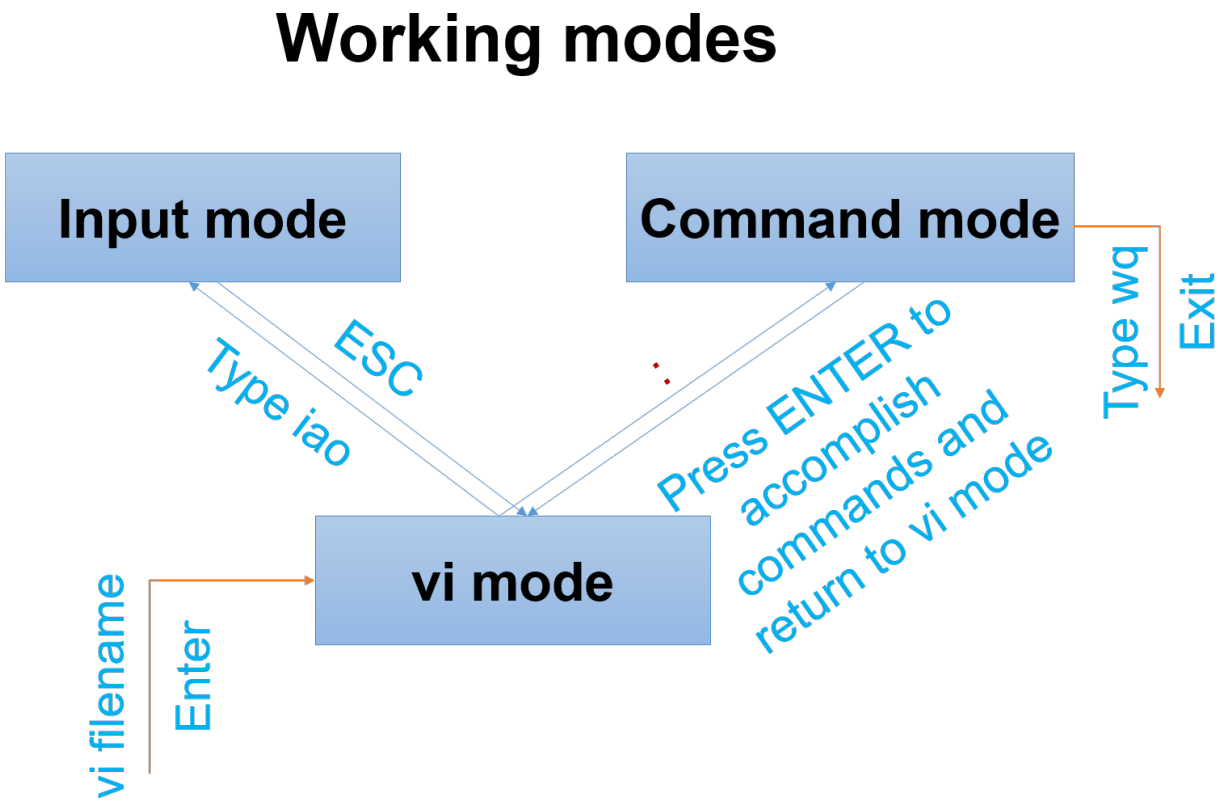
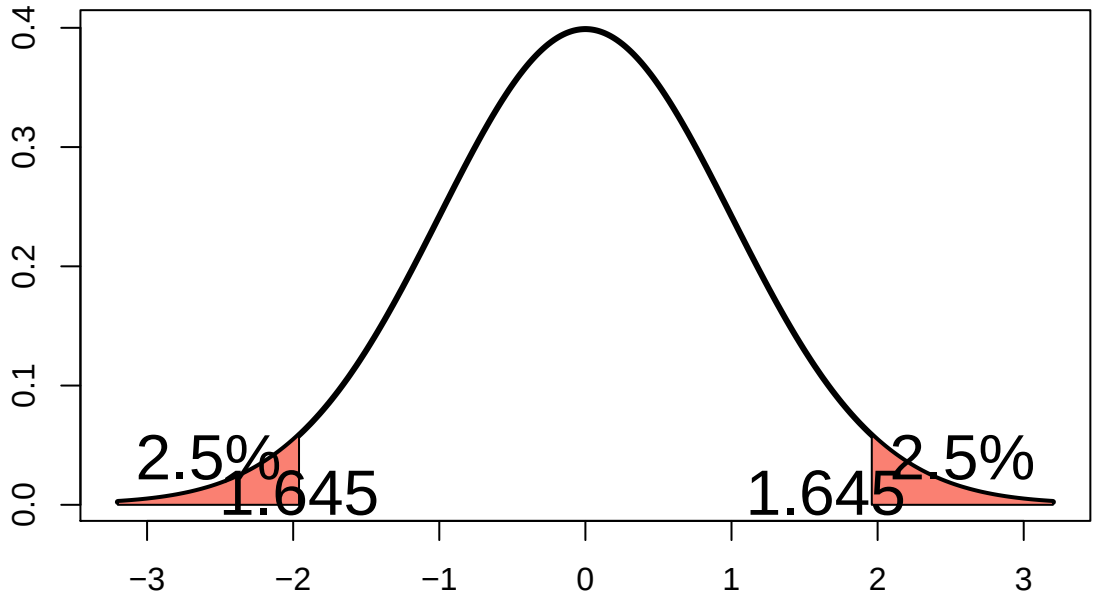


Figure 4.1: Three modes of vi.

```
text(qnorm(.95), 0.01, "1.645",cex=2)

x<-seq(-3.2, qnorm(.025), length =100)
polygon(c(x,rev(x)), c(dnorm(x), rep(0,length(x))), col="salmon")
text(mean(x),mean(dnorm(x))+0.02, "2.5%", cex=2)
text(qnorm(.025), 0.01, "1.645",cex=2)
```



4.2.1 Shell scipt

https://wikis.utexas.edu/display/bioiteam/Example+BWA+alignment+script

4.3 How to exit VIM

https://stackoverflow.com/questions/11828270/how-to-exit-the-vim-editor

split Vim window to view multiple files at once?

###

“‘

“‘

###

https://stackoverflow.com/questions/1269603/to-switch-from-vertical-split-to-horizontal-split-fast-in-vim

https://vi.stackexchange.com/questions/64/is-it-possible-to-split-vim-window-to-view-multiple-files-at-once
https://askubuntu.com/questions/418396/what-is-the-difference-between-vi-and-vim
https://unix.stackexchange.com/questions/93144/exit-vim-more-quickly
https://stackoverflow.com/questions/53664/how-to-effectively-work-with-multiple-files-in-vim?rq=1

Chapter 5

Install Bioinformatics software in Linux

5.1 Installation from source code

Nearly all of the Bioinformatics softwares will be downloaded as a compressed files. So the first thing you need to do is to uncompress the file. Then the source codes will be included in a folder. You can `cd` to the folder and `ls` the files/directories. Mostly you will find either a file named `README` or `INSTALL` or both. If you read this file to know how to install the software.

5.1.1 Install `bwa`

```
wget https://sourceforge.net/projects/bio-bwa/files/bwa-0.7.15.tar.bz2
tar xjvf bwa-0.7.15.tar.bz2
cd bwa-0.7.15
make
```

5.1.2 Install `samtools`

Installation of `Samtools` is one of the best representatives of how to instsall a Bioinformatics tool.

```
# Download the source code
wget https://iweb.dl.sourceforge.net/project/samtools/samtools/1.3.1/samtools-1.3.1.tar.bz2
# Uncompress the source code
tar xjvf samtools-1.3.1.tar.bz2
# Enter the source code directory.
cd samtools-1.3.1
# Configure the build system
./configure
# Build samtools
make
# Become a `root` user for system-wide install:
su root
```

```
# Install `Samtools`
make install
```

Install samtools without root previledges

By default, 'make install' installs samtools and the utilities under /usr/local/bin and manual pages under /usr/local/share/man.

You can specify a different location to install Samtools by configuring with –prefix=DIR or specify locations for particular parts of HTSlib by configuring with –bindir=DIR and so on. Type './configure –help' for the full list of such install directory options.

Alternatively you can specify different locations at install time by typing 'make prefix=DIR install' or 'make bindir=DIR install' and so on. Consult the list of prefix/exec_prefix/etc variables near the top of the Makefile for the full list of such variables that can be overridden.

You can also specify a staging area by typing 'make DESTDIR=DIR install', possibly in conjunction with other –prefix or prefix=DIR settings. For example,

```
make DESTDIR=/tmp/staging prefix=/opt
```

would install into bin and share/man subdirectories under /tmp/staging/opt.

5.1.3 Align reads to genome using bwa and store the alignment results in SAM/BAM files

```
./bwa index ref.fa
./bwa mem ref.fa read-se.fq.gz | gzip -3 > aln-se.sam.gz
./bwa mem ref.fa read1.fq read2.fq | gzip -3 > aln-pe.sam.gz
```

5.2 Installing a precompiled binary (executable)

For programs that are already compiled (converted from high level source code in a language like C into machine specific code), you are often given some choices and need to determine how to download the version that has the correct CPU architecture for your machine.

5.2.1 Install bwa

```
wget https://downloads.sourceforge.net/project/bio-bwa/bwakit/bwakit-0.7.15_x64-linux.tar.bz2
tar xjvf bwakit-0.7.15_x64-linux.tar.bz2
cd bwa.kit/
./bwa
Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.15-r1140
Contact: Heng Li <lh3@sanger.ac.uk>

Usage:   bwa <command> [options]

Command: index         index sequences in the FASTA format
         mem           BWA-MEM algorithm
         fastmap       identify super-maximal exact matches
         pemerge       merge overlapping paired ends (EXPERIMENTAL)
```

```
         aln           gapped/ungapped alignment
         samse         generate alignment (single ended)
         sampe         generate alignment (paired ended)
         bwasw         BWA-SW for long queries

         shm           manage indices in shared memory
         fa2pac        convert FASTA to PAC format
         pac2bwt       generate BWT from PAC
         pac2bwtgen    alternative algorithm for generating BWT
         bwtupdate     update .bwt to the new format
         bwt2sa        generate SA from BWT and Occ

Note: To use BWA, you need to first index the genome with `bwa index'.
      There are three alignment algorithms in BWA: `mem', `bwasw', and
      `aln/samse/sampe'. If you are not sure which to use, try `bwa mem'
      first. Please `man ./bwa.1' for the manual.
```

5.2.2 Achiving and compressing files

5.2.3

Linux: https://www.youtube.com/watch?v=tSRlNwaUgPQ&list=PLtK75qxsQaMLZSo7KL-PmiRarU7hrpnwK&index=27

Chapter 6

First Perl Program

Scripting languages, like Perl, are very commonly used in bioinformatics. As a generous scripting language, Perl have many advantages: easy to use, free for all operating systems like Linux, designed for working with text files (tab-delimited files). It's one of the most popular language in bioinformatics. Moreover there are many scripts and modules available. Additionally, there are a lot of resource on Internet. http://stan.cropsci.uiuc.edu/courses/cpsc565/Lecture_3.ppt

Example: Deitel et al. (2001)

6.1 First Program

As all other programming books, we begin with a "Hello world" program.

```
#!/usr/bin/perl
#Printing a line of text "Hello, Bioinformatics"
print "Hello, Bioinformatics!\n";
```

This program show how to display a line a text in Perl. It have several features. We go through each line in detail.

Line 1 is what we call shebang line. This line starts with shebang construct (`#!`). `/usr/bin/perl` indicates the path of the Perl interpreter.

Line 3 shows how to print a line of text in Perl. Nearly all programming language use print to display texts on the screen. Here, print is a built-in function in Perl. It print the string of characters (its arguments) between quotation marks ("" or '').

```
perl code_perl/hello_bioinfor.pl
```

```
Hello, Bioinformatics!
```

```
ls -l code_perl/hello_bioinfor.pl
code_perl/hello_bioinfor.pl
```

```
-rwxr-xr-x 1 xie186 xie186 103 May 24 23:15 code_perl/hello_bioinfor.pl
Hello, Bioinformatics!
```

```
chmod 755 code_perl/hello_bioinfor.pl
code_perl/hello_bioinfor.pl
```

```
Hello, Bioinformatics!
```

```
ls -l code_perl/hello_bioinfo.pl
code_perl/hello_bioinfo.pl
```

```
Error in running command sh
```

After we press ENTER on the keyboard, the program will execute and display the output in Figure 5.1 if the program is syntactically correct. However the characters `\n` are not displayed. Here backslash `\` is a start of an escape sequence. It changes the meaning of the character after it. The backslash `\` and `n` together (`\n`) form an escape sequence and signify a newline. Other examples are `\t` (tab) or `\$` (= print an actual dollar sign, normally a dollar sign has a special meaning). We'll see more escape sequences in 7.1. You can try to remove `\n` in the program to see what will happen. This will give you a more understanding of the program.

6.1.1 How to read extract a set of sequences from the reference genome

Chapter 7

Arrays in Perl

7.1 Array index

7.2 Sort Arrays in Perl

7.2.1 Sort alphebetically

7.2.2 Sort numerically

To sort an array numerically, we use `spaceship operator: <=>`.

```
my @genome_coor = (100, 300, 200, 500);
my @sorted_coor = sort {$a <=> $b} @genome_coor;

print "Array before sorted: ", "@genome_coor", "\n";
print "Array after sorted: ", "@sorted_coor", "\n";
```

```
## Array before sorted: 100 300 200 500
## Array after sorted: 100 200 300 500
```

Similarly, array can be also sorted numerically in decreasing order.

```
my @genome_coor = (100, 300, 200, 500);
## {$a <=> $b} is modified as {$b <=> $a}
my @sorted_coor = sort {$b <=> $a} @genome_coor;

print "Array before sorted: ", "@genome_coor", "\n";
print "Array after sorted: ", "@sorted_coor", "\n";
```

```
## Array before sorted: 100 300 200 500
## Array after sorted: 500 300 200 100
```

7.2.2.1 Practical exersise

https://perlmaven.com/sorting-arrays-in-perl

////////////////////////////////

Scripting languages, like Perl, are very commonly used in bioinformatics. As a generous scripting language, Perl have many advantages: easy to use, free for all operating systems like Linux, designed for working with text files (tab-delimited files). It's one of the most popular language in bioinformatics. Moreover there are many scripts and modules available. Additionally, there are a lot of resource on Internet. http://stan.cropsci.uiuc.edu/courses/cpsc565/Lecture_3.ppt

Example: Deitel et al. (2001)

7.3 First Program

As all other programming books, we begin with a "Hello world" program.

```
#!/usr/bin/perl
#Printing a line of text "Hello, Bioinformatics"
print "Hello, Bioinformatics!\n";
```

This program show how to display a line a text in Perl. It have several features. We go through each line in detail.

Line 1 is what we call shebang line. This line starts with shebang construct (`#!`). `/usr/bin/perl` indicates the path of the Perl interpreter.

Line 3 shows how to print a line of text in Perl. Nearly all programming language use print to display texts on the screen. Here, print is a built-in function in Perl. It print the string of characters (its arguments) between quotation marks ("" or '').

```
perl code_perl/hello_bioinfor.pl
```

```
Hello, Bioinformatics!
```

```
ls -l code_perl/hello_bioinfor.pl
code_perl/hello_bioinfor.pl
```

```
-rwxr-xr-x 1 xie186 xie186 103 May 24 23:15 code_perl/hello_bioinfor.pl
Hello, Bioinformatics!
```

```
chmod 755 code_perl/hello_bioinfor.pl
code_perl/hello_bioinfor.pl
```

```
Hello, Bioinformatics!
```

```
ls -l code_perl/hello_bioinfo.pl
code_perl/hello_bioinfo.pl
```

```
Error in running command sh
```

After we press ENTER on the keyboard, the program will execute and display the output in Figure 5.1 if the program is syntactically correct. However the characters `\n` are not displayed. Here backslash `\` is a start of an escape sequence. It changes the meaning of the character after it. The backslash `\` and `n` together (`\n`) form an escape sequence and signify a newline. Other examples are `\t` (tab) or `\$` (= print an actual dollar sign, normally a dollar sign has a special meaning). We'll see more escape sequences in 7.1. You can try to remove `\n` in the program to see what will happen. This will give you a more understanding of the program.

7.4 Vector

7.5 Dataframe

Chapter 8

Varible and arithmetic operators

8.1 Varaible

Perl provides three kinds of variables: scalars, arrays, and associative arrays. The initial character of the name identifies the particular type of variable and, hence, its functionality.

8.1.1 scalar variable

either a number or string; Perl does not differentiate between the two, nor does it differentiate between integers and reals. In order to tell the computer what to print, we need to use variables. In Perl, the name of a variable starts with the dollar sign $. You can assign either a number or a string to it.

```
#!/usr/bin/perl
use warnings;
use strict;

#assign two strings to two variables
my $dna_seq1 = "ACCTCGGTACAGTGAATGGGAAACGTAGCTGAT";
my $dna_seq2 = "TGCCGATCGTAATAGCTCGCTATCTAGCTCGATCGTCGTA";

#Returns the length in characters of the value of EXPR
my $dna_length1 = length $dna_seq1;
my $dna_length2 = length $dna_seq2;

print "First DNA: $dna_seq1\n";
print "Second DNA: $dna_seq2\n";

#calculate the total length of two DNA sequences
my $tot_length = $dna_length1 + $dna_length2;
print "The total length of two DNA sequences is $tot_length \n";

# calculate the average length of two DNA sequences.
my $average_length = ($dna_length1 + $dna_length2) / 2;
print "The average length of two DNA sequences is $average_length \n";

#calulate the differences between the two DNA sequences
my $diff_length = $dna_length2 - $dna_length1;
```

```
print "The length difference of two DNA sequences is ", $diff_length, "\n";

#Imaging the CC content of first DNA sequence is 0.5.
#How many CC do we have in first DNA?
my $cg_content = 0.5;
my $cg_number = $dna_length1 * 0.5;

#Imaging we have a 10 bps DNA sequnces,
#how many possible DNA sequences do we have?
my $dna_nucleotide = "ATCG";
my $dna_nucleo_number = length $dna_nucleotide;
my $dna_length = 10;
my $possible_number = $dna_nucleo_number ** $dna_length;
print "We have $possible_number possibilities.\n";
```

Here are the explanations of this script.

```
# assign two DNA sequences to two variables
my $dna1 = "CTCGACCAGGACGATGAATGGGCGATGAAAATCT";
my $dna2 = "CGCTAAACGCTAAACCCTAAACGCTAAACCTCTGAATCCTTAATCGCT";
```

The first line here is a comment. In Perl, # (pound sign) is the comment character. The comments can be used to document the program and improve the readability. During the execution, the comments will be ignored.

Mutilple lines comments

The quick-and-dirty method only works well when you don't plan to leave the commented code in the source. If a Pod parser comes along, your multiline comment is going to show up in the Pod translation. A better way hides it from Pod parsers as well.

The =begin directive can mark a section for a particular purpose. If the Pod parser doesn't want to handle it, it just ignores it. Label the comments with comment. End the comment using =end with the same label. You still need the =cut to go back to Perl code from the Pod comment:

The following two

8.1.1.1 use strict; user warnings, my

```
#!/usr/bin/perl

#assign 2 numbers to 2 variable
$dna1 = "CTCGACCAGGACGATGAATGGGCGATGAAAATCT";
$dna2 = "CGCTAAACGCTAAACCCTAAACGCTAAACCTCTGAATCCTTAATCGCT";

#Returns the length in characters of the value of EXPR
$dna_length1 = length $dna1;
$dna_length2 = length $dna2;

print "First DNA: $dna1; Length: $dna_length1\n";
print "Second DNA: $dna2; Length: $dna_length2\n";
```

```
#calculate the total length of two DNA sequences
$tot_length = $dna_length1 + $dna_lenght2;
print "The total length of two DNA sequences is $tot_length \n";
```

In the script above, `$dna_lenght2` is a typo. If you run this script, it will give you the output without error message, although it's not the right output.

```
perl code_perl/var_assign_no_strict_warnings.pl
```

```
First DNA: CTCGACCAGGACGATGAATGGGCGATGAAAATCT; Length: 34
Second DNA: CGCTAAACGCTAAACCCTAAACGCTAAACCTCTGAATCCTTAATCGCT; Length: 48
The total length of two DNA sequences is 34
```

```
#!/usr/bin/perl
use warnings;
use strict;

#assign 2 numbers to 2 variable
my $dna1 = "CTCGACCAGGACGATGAATGGGCGATGAAAATCT";
my $dna2 = "CGCTAAACGCTAAACCCTAAACGCTAAACCTCTGAATCCTTAATCGCT";

#Returns the length in characters of the value of EXPR
my $dna_length1 = length $dna1;
my $dna_length2 = length $dna2;

print "First DNA: $dna1; Length: $dna_length1\n";
print "Second DNA: $dna2; Length: $dna_length2\n";

#calculate the total length of two DNA sequences
my $tot_length = $dna_length1 + $dna_lenght2;
print "The total length of two DNA sequences is $tot_length \n";
```

If we add `use strict` and `use warnings`,

```
Global symbol "$dna_lenght2" requires explicit package name at code_perl/var_assign_strict_warnings.pl lin
Execution of code_perl/var_assign_strict_warnings.pl aborted due to compilation errors.
```

```
Global symbol "$dna_lenght2" requires explicit package name
at code_perl/var_assign_strict_warnings.pl line 17.
Execution of code_perl/var_assign_strict_warnings.pl aborted
 due to compilation errors.
```

8.1.2 Array

8.1.3 Hash

Chapter 9

Control structure

…. Perl is an iterative language in which control flows from the first statement in the program to the last statement unless something interrupts. Some of the things that can interrupt this linear flow are conditional branches and loop structures. Perl offers approximately a dozen such constructs, which are described below. The basic form will be shown for each followed by a partial example. ….

9.1 `while` loop

9.2 `for` loop

9.3 `foreach` loop

9.4 Statement if-else

```
#!/usr/bin/perl
use warnings;
use strict;

my @gene_exp_lev = (1, 5, 3, 4, 9, 10);

# for (my $i = 0; $i < @gene_exp_lev; $i++) {
for (my $i = 0; $i < scalar @gene_exp_lev; $i++) {
    if ($gene_exp_lev[$i] > 4) {
        print "Index $i: $gene_exp_lev[$i]\n";
    }
}
```

```
Index 1: 5
Index 4: 9
Index 5: 10
```

9.5 Operator `last` and `next`

9.6 Operator `redo`

Chapter 10

String manipulation

10.1 String concatenation

Dot (.) can be used to concatenate two strings together.

```
# concatenate two strings together and assing to $z
$z = $x . $y;
# Append $y to $x
$x = $x . $y;
# Append $y to $x
$x .= $y;
```

A more convenient way is to use operator .= to append one variable to another.

As is any other assignments in Perl, if you see an assignment written this way $x = $x op expr, where op stands for any operator and expr stands for the rest of the statement, you can make a shorter version by moving the op to the front of the assignment, e.g., $x op= expr. The string concatenation operator . is just one possible op among many others such as +, -, * and /.

```
$x = 5;
$y = 6;
# Append $y to $x
$x = $x + $y;
# Append $y to $x
$x += $y;
```

10.2 Substring extraction

10.3 Substring search

Index

10.4 Split String

split

join

10.5 Regular expression

10.5.1 Pattern matching

10.5.2 Pattern substitution

10.5.3 Modifiers to pattern matching and substitution

10.5.4 Greedy or non-greedy

Chapter 11

Input and output in Perl

11.1 Input

11.1.1 Standard input

11.1.2 Input from a file on the disk

```
#!/usr/bin/perl
use warnings;
use strict;

my $fasta = "./data/test_ref.fa";

open FASTA, $fasta or die "$!: $fasta";
while(my $line = <FASTA>){
    chomp $line;
    print "$line\n";
}
close FASTA;
```

```
perl code_perl/open_file.pl
```

```
## >chr1
## ACGCTAGCTAGTCAGTCGATCGT
## CGTAGCTAGCTAG
## >chr2
## CTGCGGGCTAAATCGATCGATCG
## GTACGTACGAGCTAGCTA
## >chr3
## CTGCGGGCTAAATCGATCGATCG
## GTACGTACGAGCTAGCTA
```

11.1.3 Special variable $_

```
#!/usr/bin/perl
use warnings;
use strict;

my $fasta = "./data/test_ref.fa";

open FASTA, $fasta or die "$!: $fasta";
while(my $line = <FASTA>){
    chomp $line;
    print "$line\n";
}
close FASTA;
```

When you are using `while` and `foreach`, the content will be assigned to `$_` if you don't assign it explicitly to any variable.

For some build-in functions (`chomp`, `length`, `split` and `print`),

I do NOT suggest you to use `$_` because `$_` will reduce the readerbility of the code.

In Perl, $_ is a powerful In Perl, several functions and operators use this variable as a default, in case no parameter is explicitly used. In general, I'd say you should NOT see $_ in real code. I think the whole point of $_ is that you don't have to write it explicitly.

11.1.4 The input record separator: $/

`$/` is the input record separator. By default, `$/` equals newline (`\n`).

You could actually change the value of `$/`. For example, by assigning the greate-than '>' like this: `$/ = ">";`. Then every call to the read-line operator `$chunk = <FILEHANDLE>` will read in all the characters up-to and including the first '>'.

We could also assign longer strings to `$/` and then that would be the input record separator.

11.1.4.1 How to process `fasta` file by changing `$/`

FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes (A, T, G, C, etc.).

In `fasta` format, each sequence begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol in the first column.

Here is an example of fasta file (Figure 11.1).

```
#!/usr/bin/perl
use warnings;
use strict;

my $fasta = "./data/test_ref.fa";

open FASTA, $fasta or die "$!: $fasta";
# Set the input record separator
$/ = "\n>";
```

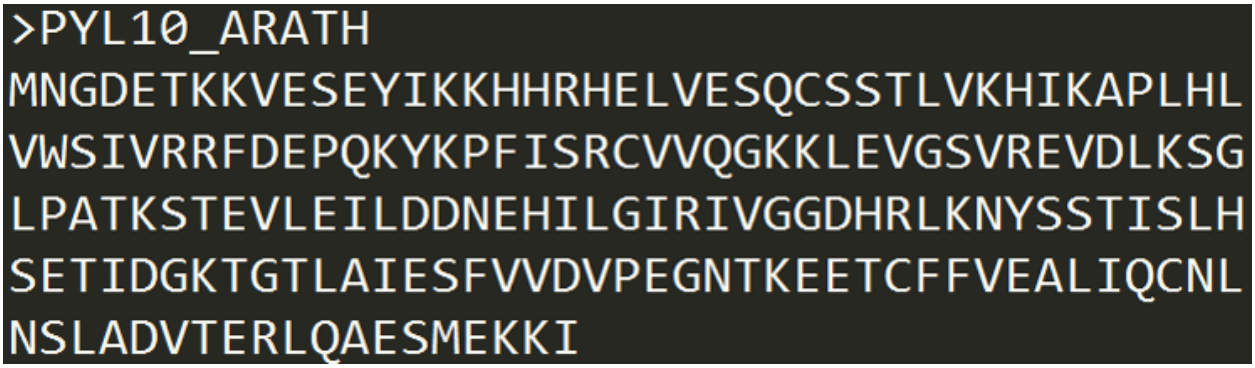


Figure 11.1: Protein sequence of PYL10 in Arabidopsis.

```
# Print header
print "Chrom\tLength\n";
while(my $chunk = <FASTA>){
    # Remove > in the first chunk
    $chunk =~ s/^>//;
    # Remove \n> from the end of the chunk
    chomp $chunk;
    # Assign ID and seq to two variables
    my ($chrom, $seq) = split(/\n/, $chunk, 2);
    # Remove \n in the string
    $seq =~ s/\n//g;
    # Calculate the length of sequence.
    my $seq_length = length $seq;
    # Print ID and length of the sequence.
    print "$chrom\t$seq_length\n";
}
close FASTA;
```

You can consider the content in the fasta file as a string (Figure 11.2) sperarated by `\n>`.

We set the input record separator `$/` as `\n>`.

In the first cycle of the `while` loop, the first chunk is `>chr1\nACGCTAGCTAGTCAGTCGATCGT\nCGTAGCTAGCTAG\n>`.

You should notice that there is a leading `>` in the first chunk. So you need to use `s/^>//` to remove the leading `>`.

To remove the trailing `\n>`, you can use `chomp $chunk;`. Then the string is:

`chr1\nACGCTAGCTAGTCAGTCGATCGT\nCGTAGCTAGCTAG`

The string before first `\n` is the sequence ID. The string after the first `\n` is the sequence. To get the sequnce ID and sequnce and assign to two variables, you can use `split(/\n/, $chunk, 2)`. Here LIMIT is set to 2, it represents the maximum number (here 2) of fields into which the EXPR may be split.

The goal of this code is to output the sequence IDs and their lengths. Currently the string of the variable (`$seq`) is `ACGCTAGCTAGTCAGTCGATCGT\nCGTAGCTAGCTAG`. To do this, you first need to remove `\n` in the sequence by using `$seq =~ s/\n//g;`. Now the string of the variable (`$seq`) is `ACGCTAGCTAGTCAGTCGATCGTCGTAGCTAGCTAG`. Then built-in function `length` can be used to calculate the length of the sequence. The function `print` is used to output the sequnce ID and the length of the sequence.

In the second cycles of the `while` loop, the chunk is `chr2\nCTGCGGGCTAAATCGATCGATCG\nGTACGTACGAGCTAGCTA\n>`.

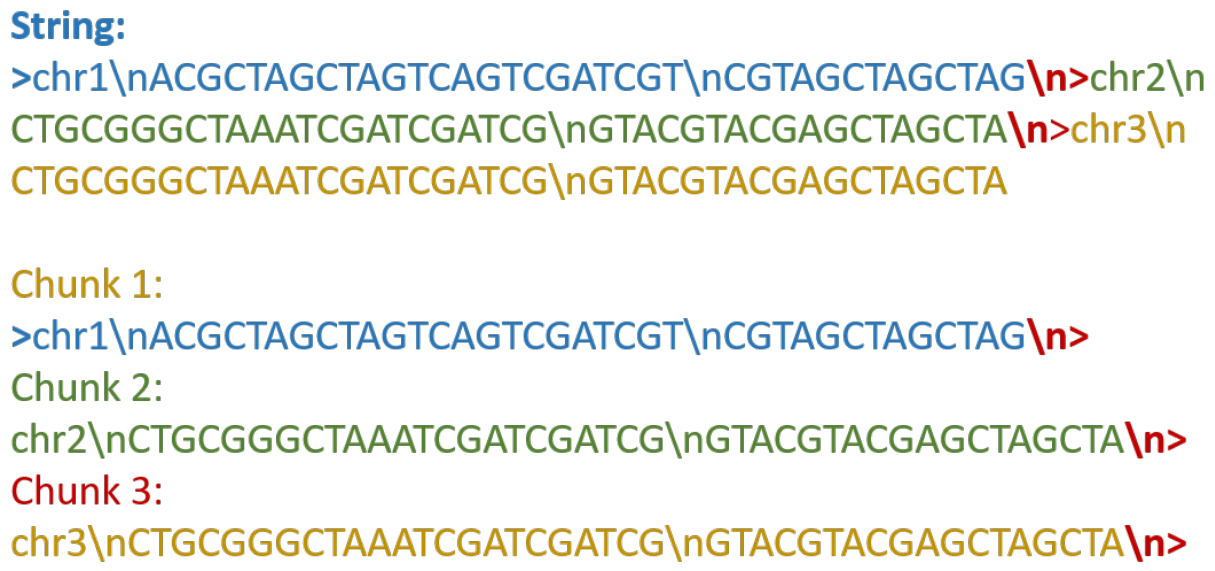


Figure 11.2: Fasta string.

Unlike the first chunk, there is no leading `>` in the second chunk. So the code `s/^>//` will do nothing for the second chunk.

The following steps is the same as you can do for the first cycle.

In the third cycle of the `while` loop, you can do the same thing on the third sequence.

```
perl code_perl/fa_seq_len.pl
```

```
## Chrom    Length
## chr1 36
## chr2 41
## chr3 41
```

11.2 Output to a file on the disk

```
#!/usr/bin/perl
use warnings;
use strict;

my $fasta   = "./data/test_ref.fa";
my $out_len = "./data/test_ref_len.txt";

open FASTA, $fasta or die "Can't open file for reading: $! $fasta";
open OUT, "+>$out_len" or die "Can't open file for writing: $! $out_len";
# Set the input record separator
$/ = "\n>";
# Print header
print OUT "Chrom\tLength\n";
```

```
while(my $chunk = <FASTA>){
    # Remove > in the first chunk
    $chunk =~ s/^>//;
    # Remove \n> from the end of the chunk
    chomp $chunk;
    # Assign ID and seq to two variables
    my ($chrom, $seq) = split(/\n/, $chunk, 2);
    # Remove \n in the string
    $seq =~ s/\n//g;
    # Calculate the length of sequence.
    my $seq_length = length $seq;
    # Print ID and length of the sequence.
    print OUT "$chrom\t$seq_length\n";
}
close FASTA;
close OUT;
```

```
perl code_perl/fa_seq_len_out.pl
cat ./data/test_ref_len.txt
```

```
## Chrom    Length
## chr1 36
## chr2 41
## chr3 41
```

11.3 Arguments from command line

Imagine you have 5 fasta files and you want to calculate sequence lengths for all the 5 files, if you modify the file each time you run the code, it will be very tedious.

Perls provides an array called `@ARGV`. `@ARGV` holds all the arguments from the command line. The first one will be `$ARGV[0]`.

`@ARGV` will automatically hold all the arguments. If no arguments are provided, `@ARGV` will be empty.

The following example shows you how it looks like in real code.

```
#!/usr/bin/perl
use warnings;
use strict;

my ($fasta, $out_len) = @ARGV;
print "First arguments \$ARGV[0] : $ARGV[0]\n";
print "Second arguments \$ARGV[0]: $ARGV[1]\n";

print "The variable \$fasta:   $fasta\n";
print "The variable \$out_len: $out_len\n";
```

```
perl code_perl/test_argv.pl test_ref.fa test_ref_len.txt
```

```
## First arguments $ARGV[0] : test_ref.fa
## Second arguments $ARGV[0]: test_ref_len.txt
## The variable $fasta:   test_ref.fa
## The variable $out_len: test_ref_len.txt
```

In the above example, the first argument is `$ARGV[0]` which stores the file name: test_ref.fa. The second one stores the file name: test_ref_len.txt. It's highly recommended to assign the values in `@ARGV` to some variables that the readers (including yourself) can understand the variables based on the names.

Here I'll show you how to use ARGV using an example. The code below will show you what to do if you want to filter a fasta file based on the sequence length.

```
#!/usr/bin/perl
use warnings;
use strict;

my ($fasta, $out_len) = @ARGV;

open FASTA, $fasta or die "Can't open file for reading: $! $fasta";
open OUT, "+>$out_len" or die "Can't open file for writing: $! $out_len";
# Set the input record separator
$/ = "\n>";
# Print header
print OUT "The input file name is: $fasta\n";
print OUT "Chrom\tLength\n";
while(my $chunk = <FASTA>){
    # Remove > in the first chunk
    $chunk =~ s/^>//;
    # Remove \n> from the end of the chunk
    chomp $chunk;
    # Assign ID and seq to two variables
    my ($chrom, $seq) = split(/\n/, $chunk, 2);
    # Remove \n in the string
    $seq =~ s/\n//g;
    # Calculate the length of sequence.
    my $seq_length = length $seq;
    # Print ID and length of the sequence.
    print OUT "$chrom\t$seq_length\n";
}
close FASTA;
close OUT;
```

For example if you want to remove the sequences that is shorter than 30 bps, you can use the following example.

```
printf "Before filtering:\n"
cat ./data/test_ref2.fa
perl code_perl/fa_seq_len_out_argv_fil.pl ./data/test_ref2.fa 30 ./data/test_ref2_30.fa

printf "\nAfter filtering: \n"
cat ./data/test_ref2_30.fa
```

```
## Before filtering:
## >chr1
```

```
## ACGCTAGCTAGTCAGTCGATCGT
## CGTAGCTAGCTAG
## >chr2
## CTGCGGGCTAAATCGATCGATCG
## GTACGTACGAGCTAGCTAA
## >chr3
## CTGCGGGCTAAATCGATCGATCG
## GTACGTACGAG
## >chr4
## CTGCGGGCTAAATCAGCTAA
## >chr5
## CTGCTCGATCGATCGACGAGCTA
## GCTA
## After filtering:
## The input file name is: ./data/test_ref2.fa
## Chrom    Length
## >chr1 36
## ACGCTAGCTAGTCAGTCGATCGTCGTAGCTAGCTAG
## >chr2 42
## CTGCGGGCTAAATCGATCGATCGGTACGTACGAGCTAGCTAA
## >chr3 34
## CTGCGGGCTAAATCGATCGATCGGTACGTACGAG
```

If you have multiple files, you can just change the file names in the command line. It's very handy. You can also change the length cutoff in the command line.

Chapter 12

Perl modules

12.1 What is a Perl module

Perl modules are a set of related functions in a library file. They are specifically designed to be reusable by other modules or programs. There are more than 100,000 modules ready for you to use on the Comprehensive Perl Archive Network.

Most Perl modules are written in Perl, some use XS (they are written in C) so require a C compiler. Modules may have dependencies on other modules (almost always on CPAN) and cannot be installed without them (or without a specific version of them). Many modules on CPAN now require a recent version of Perl (version 5.8 or above).

The reason that we need to use Perl module is Perl module can largely reduce our coding work.

12.2 How to install a Perl module

Here we'll only discuss how to install Perl module in Linux system. The easies way I think is to use `cpanm`.

Here is how you can do this. First go to webpage and download the `cpanm` source code.

```
## Web link:https://raw.githubusercontent.com/miyagawa/cpanminus/master/cpanm

chmod 755 cpanm

./cpanm Bio::Seq
```

NOTE

Possible problem when using `cpanm`

You may encouter the following error message. This is a bug from Perl. You cna run `yum install perl-CPAN` resolved the issue.

Here is the ERROR MESSAGE:

! ! Can't write to /usr/local/share/perl5 and /usr/local/bin: Installing modules to /home/xie186/perl5 ! To turn off this warning, you have to do one of the following: ! - run me as a root or with –sudo option (to install to /usr/local/share/perl5 and /usr/local/bin) ! - Configure local::lib in your existing shell to set PERL_MM_OPT etc. ! - Install local::lib by running the following commands ! ! cpanm –local-lib=~/perl5 local::lib && eval $(perl -I ~/perl5/lib/perl5/ -Mlocal::lib) ! –> Working on Bio::Seq Fetching http://www.cpan.org/authors/id/C/CJ/CJFIELDS/BioPerl-1.007001.tar.gz ... OK ==>

Found dependencies: Module::Build, ExtUtils::Install –> Working on Module::Build Fetching http://www.cpan.org/authors/id/L/LE/LEONT/Module-Build-0.4222.tar.gz … OK ==> Found dependencies: Module::Metadata, version, CPAN::Meta, Perl::OSType –> Working on Module::Metadata Fetching http://www.cpan.org/authors/id/E/ET/ETHER/Module-Metadata-1.000033.tar.gz … OK ==> Found dependencies: ExtUtils::MakeMaker –> Working on ExtUtils::MakeMaker Fetching http://www.cpan.org/authors/id/B/BI/BINGOS/ExtUtils-MakeMaker-7.26.tar.gz … OK Configuring ExtUtils-MakeMaker-7.26 … OK Can't locate ExtUtils/Manifest.pm in **?** (**?** contains: FatPacked::26160008=HASH(0x18f2b88) /usr/local/lib64/perl5 /usr/local/share/perl5 /usr/lib64/perl5/vendor_perl /usr/share/perl5/vendor_perl /usr/lib64/perl5 /usr/share/perl5 .) at ./cpanm line 132.

12.3 How to use a Perl module

12.4 How to use BioPerl module

12.5 How to write a module

The name `R` is partly based on the (first) names of the first two R authors (Robert Gentleman and Ross Ihaka).

Convert string to a variable name https://stackoverflow.com/questions/6034655/convert-string-to-a-variable-name

```
assign("x", 5)

x
[1] 5
```

https://cran.r-project.org/doc/FAQ/R-FAQ.html#Why-is-R-named-R_003f

Chapter 13

Producing simple graphs

13.1 Producing simple graphs using R

13.1.1 Line Charts

13.1.2 Bar Charts

13.1.3 Histograms

13.1.4 Pie Charts

13.1.5 Dotcharts

13.1.6 Misc

Chapter 14

Preparation of figures for manuscript

Inkscape is an open source software which can be used to arrange, crop, and annotate your images; bring in graphs and charts; draw diagrams; and export the final figure in whatever format the journal wants.

14.1

Chapter 15

Start a project

15.1 Experimetal design

Before

15.1.1 How many biological replicates

15.1.2 How big

15.1.3 How much data to generate

15.1.4

Where to start?

There are many ways to start to talk about Bioinformatics analysis. Here we'll go from whether there is a reference genome or not. Let's take RNA-seq as an example.

Chapter 16

Introduction of NGS

16.1 What is NGS

Genetic information is stored within the chemical structure of the molecule DNA. DNA can be transcribed into RNA. Then RNA can be translated into protein.

NGS (Next geneneration sequencing) is not an accurate term. An more accurate term maybe high throughput sequencing. NGS is specifically used to refer to sequencing technologies after Sanger sequencing (1st generation) and single molecular sequencing (3rd sequencing, e.g. PacBio) and nanopore based sequencing (4th generation, e.g. Oxford Nanopore).

16.2 Application of NGS

16.3

16.4 Usefull links:

http://www.ecseq.com/support/ngs/trimming-adapter-sequences-is-it-necessary

http://www.gendx.com/illumina-adapter-ligation-librx

http://veleta.rosety.com/plasmid.html

Chapter 17

Practical Perl program

17.1 Extract sequences from genomic DNA for some specific regions

Figure 8.1 shows how to extract sequences from genomic DNA for some specific regions. For example, when we perform ChIP-seq for a transcription factor. We will get the binding sites (some regions) of transcription factor after analyzing ChIP-seq. You want to know whether there is motifs that can be recognized by the transcription factor.

The Perl program in Figure 8.1 has two arguments (input files): genomic DNA sequences and ChIP-seq peak regions. It will print the genomic DNA sequences for each region in fasta format. We can use > to generate a new file to record the result (Figure 8.1).

The genomic DNA sequence data is saved in fasta format (See 11.1). ChIP-seq peak regions are saved in BED format (See 12.3).

Let's go through each line in detail.

A normal paragraph.

The above code provides an interactive HTML page (figure 17.1)

```
plot(1)  # a scatterplot
```

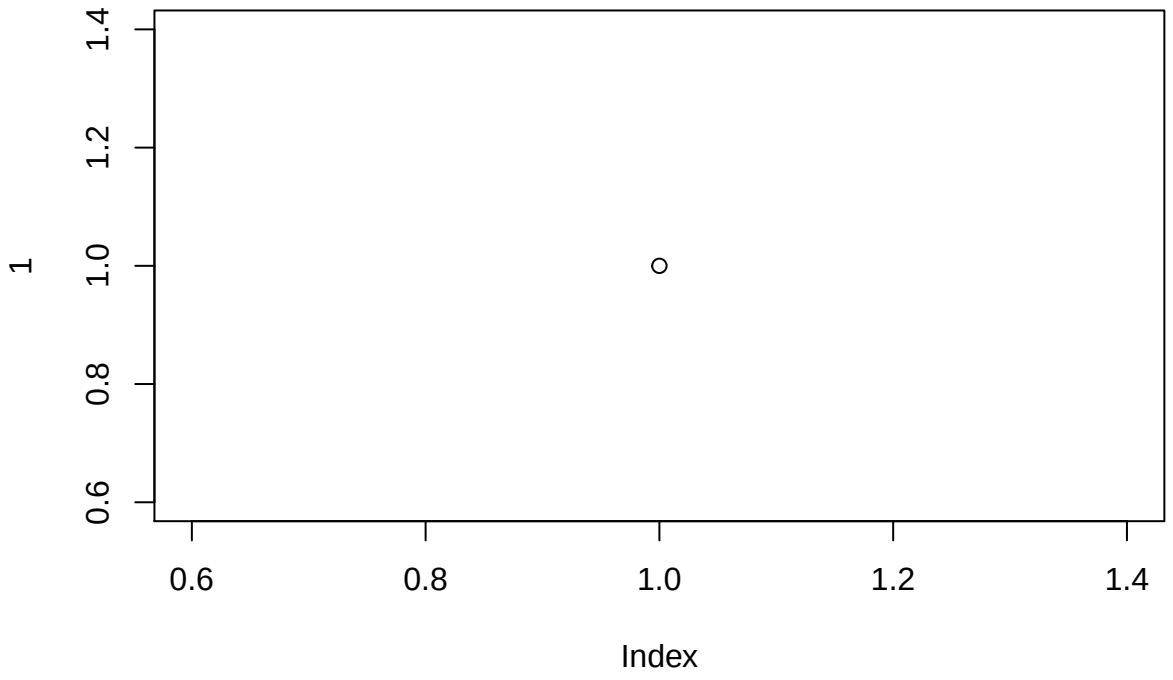


Figure 17.1: A scatterplot of the data `cars` using **base** R graphics.

Chapter 18

String manipulation

18.1 String concatenation

Dot (`.`) can be used to concatenate two strings together.

```
# concatenate two strings together and assing to $z
$z = $x . $y;
# Append $y to $x
$x = $x . $y;
# Append $y to $x
$x .= $y;
```

A more convenient way is to use operator `.=` to append one variable to another.

As is any other assignments in Perl, if you see an assignment written this way $x = $x op expr, where op stands for any operator and expr stands for the rest of the statement, you can make a shorter version by moving the op to the front of the assignment, e.g., $x op= expr. The string concatenation operator `.` is just one possible op among many others such as `+`, `-`, `*` and `/`.

```
$x = 5;
$y = 6;
# Append $y to $x
$x = $x + $y;
# Append $y to $x
$x += $y;
```

18.2 Substring extraction

18.3 Substring search

18.3.1 index

18.3.2 rindex

18.4 Split String

18.4.1 split

18.4.2 join

18.4.3 tr

18.4.4 reverse

18.4.5 markdown

18.5 Regular expression

18.5.1 Pattern matching

18.5.2 Pattern substitution

18.5.3 Modifiers to pattern matching and substitution

18.5.4 Greedy or non-greedy

18.5.5

The split function

Chapter 19

Heatmap Tutorial

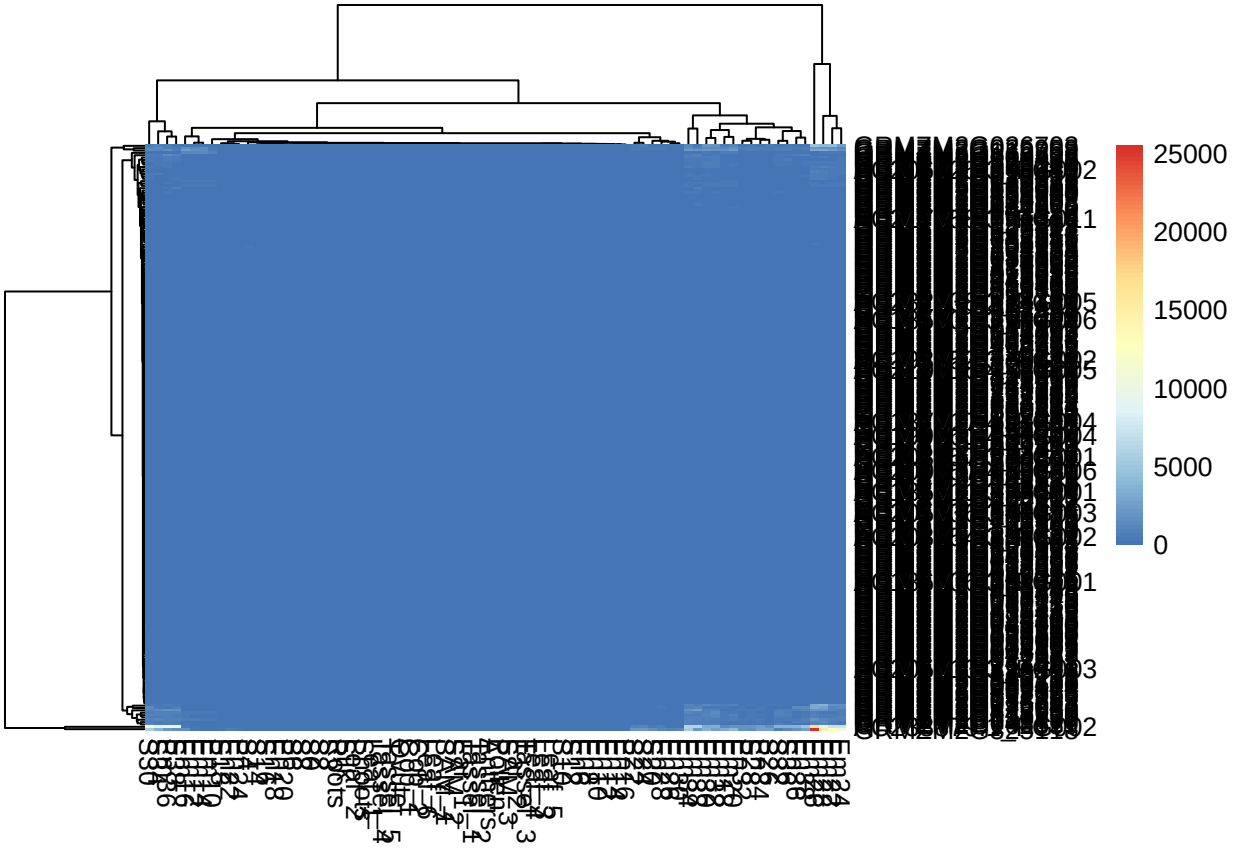
Data link

19.1 Install pheatmap package

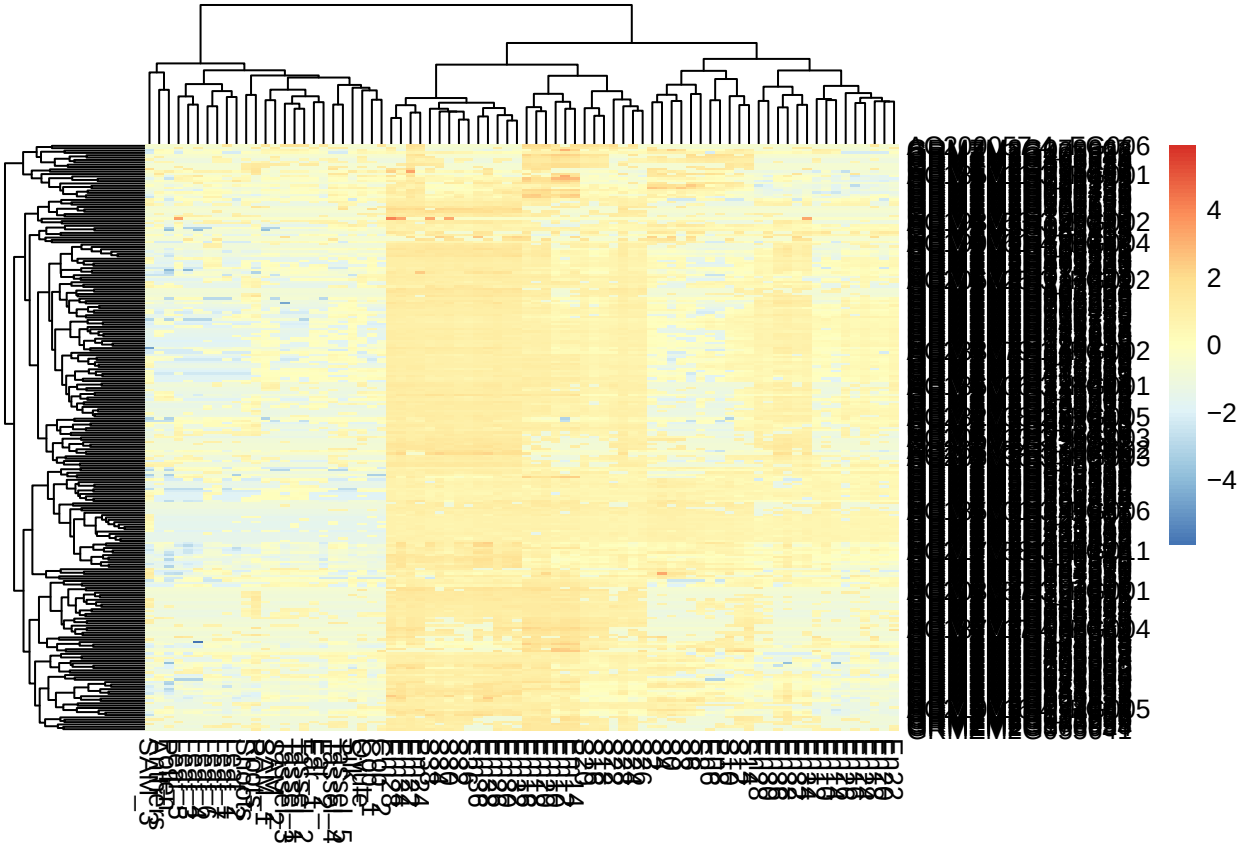
```
install.packages("pheatmap")
```

19.2 Draw a heatmap for gene expression of RNA-seq data

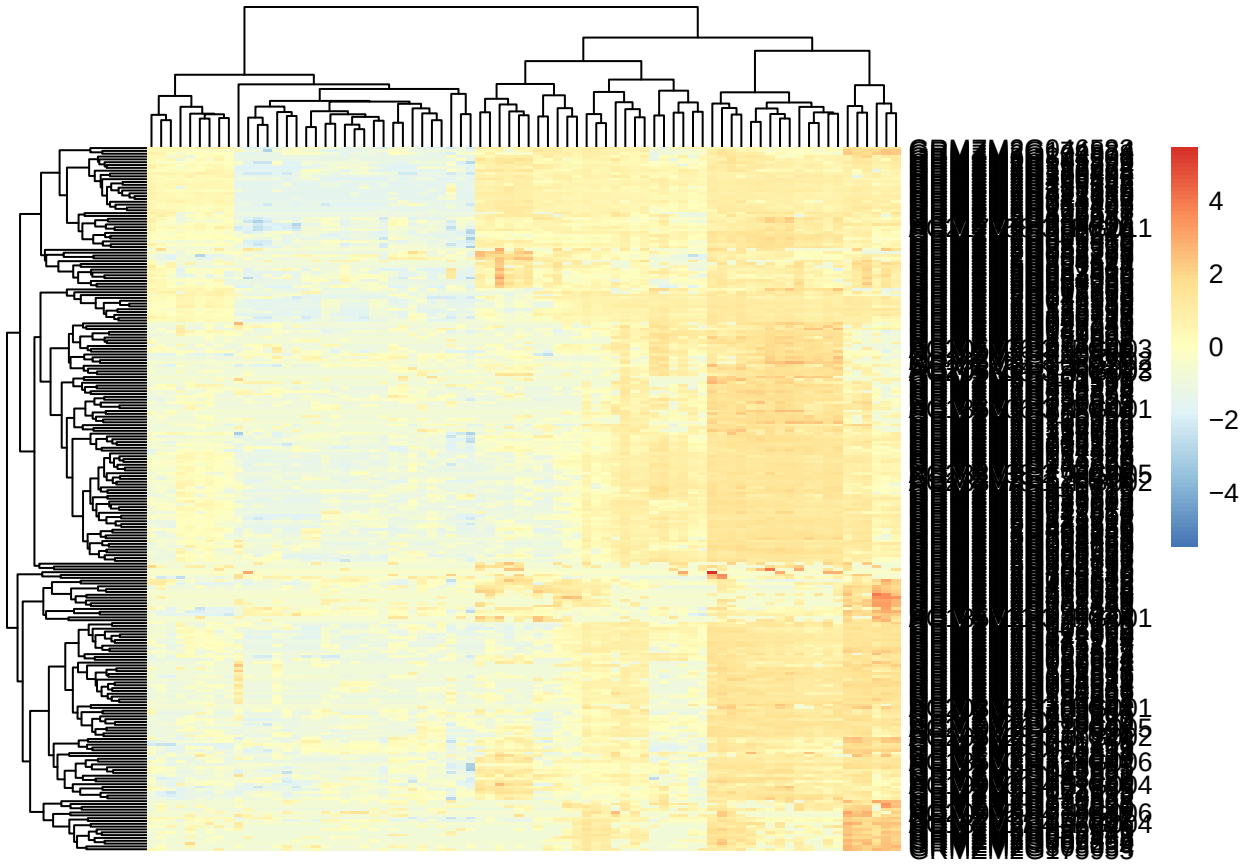
```
library(pheatmap)
gene_exp <- read.table("data/maize_embryo_specific_gene_Sheet1.tsv", header=T, row.names=1)
pheatmap(gene_exp)
```



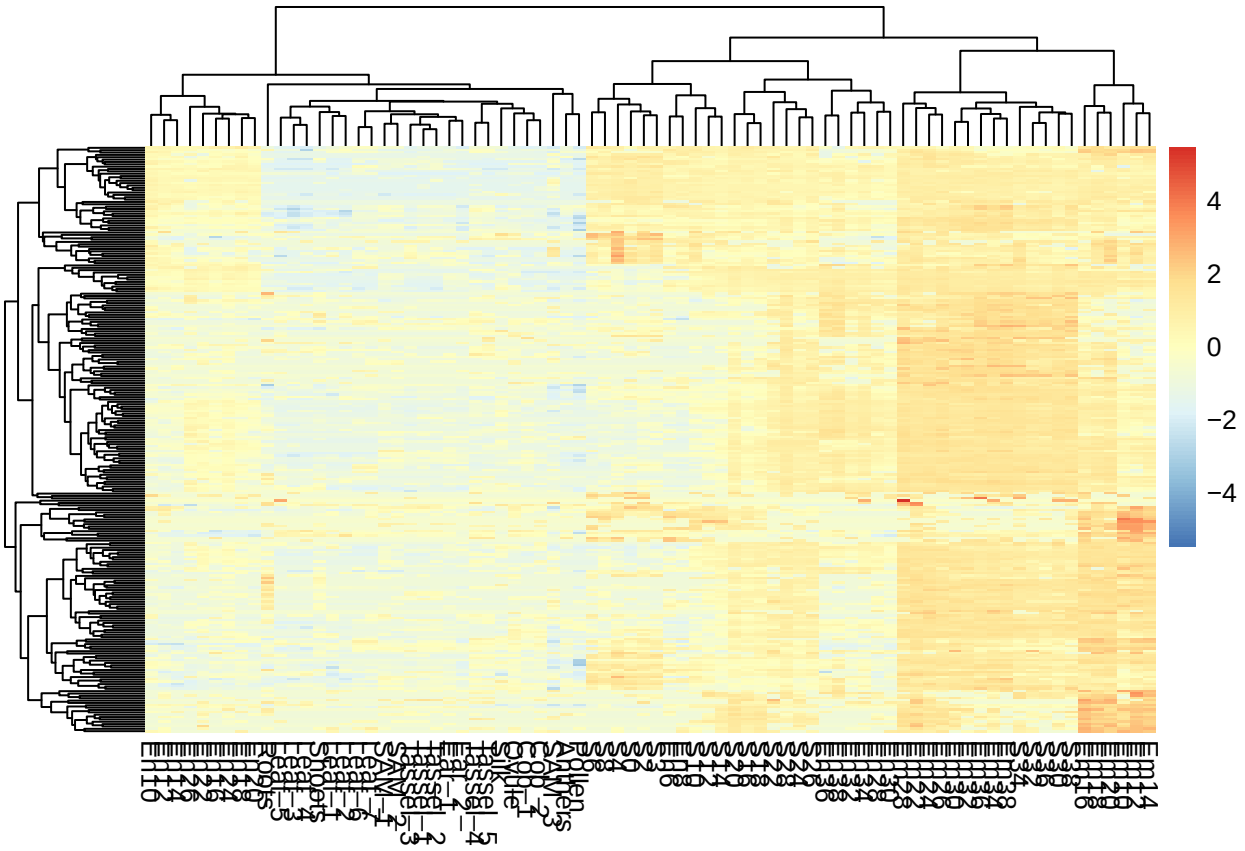

```
pheatmap(log2(gene_exp + 0.000001), scale="row")
```




```
pheatmap(log2(gene_exp + 0.01), scale="row", show_rownames = T, show_colnames = F)
```




```
pheatmap(log2(gene_exp + 0.01), scale="row", show_rownames = F, show_colnames = T)
```

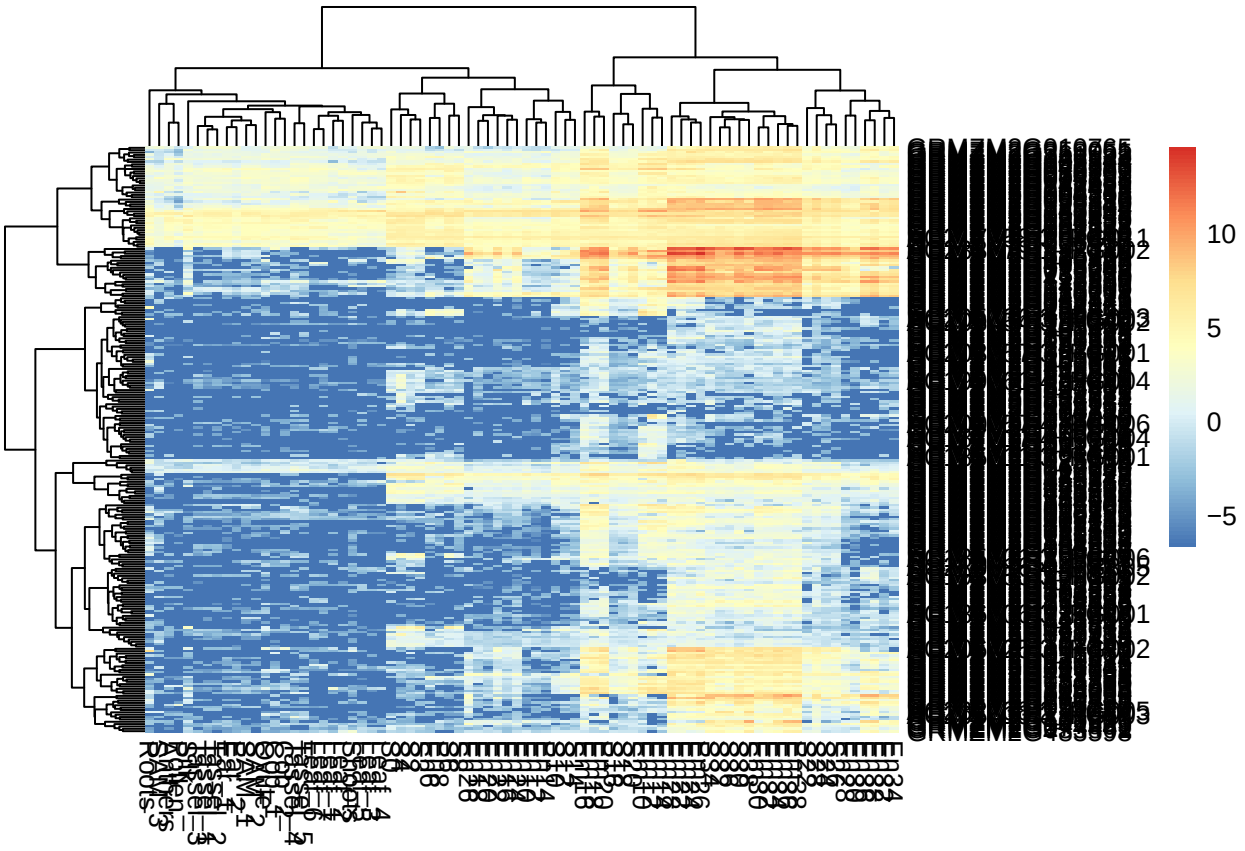


19.3 Add the annotation

19.4

19.5 Transfrom the data

```
pheatmap(log2(gene_exp + 0.01))
```



19.6 How to add annotations

19.7 How to cut the trees

19.8 How to get the cluster information from the heatmap

19.9

Chapter 20

Data analysis of RNA-seq

Chapter 21

BS-seq

21.1

https://www.slideshare.net/secret/Da9IOe8wLsaF8V

21.2 What is BS-seq

21.3

21.4 How to analyze BS-seq data

21.4.1

MINI REVIEW: Statistical methods for detecting differentially methylated loci and regions
Strategies for analyzing bisulfite sequencing data

Chapter 22

Shell

What is Linux Shell ?
http://www.freeos.com/guides/lsst/

Chapter 23

Capstone project:

23.1 Introduction

23.2 Method

23.3 Pipelines

23.4

Chapter 24

Good resouces to learn Bioinformatics

24.1 References:

https://www.bioinformatics.babraham.ac.uk/training.html

Chapter 25

Python

25.1 os

25.2

25.2.1 How to write python script to a command line program

http://omgenomics.com/python-command-line-program/

25.2.2 Python module: pysam

25.2.3 Python module: pybedtools

Chapter 26

R introduction

26.1 Basic R function

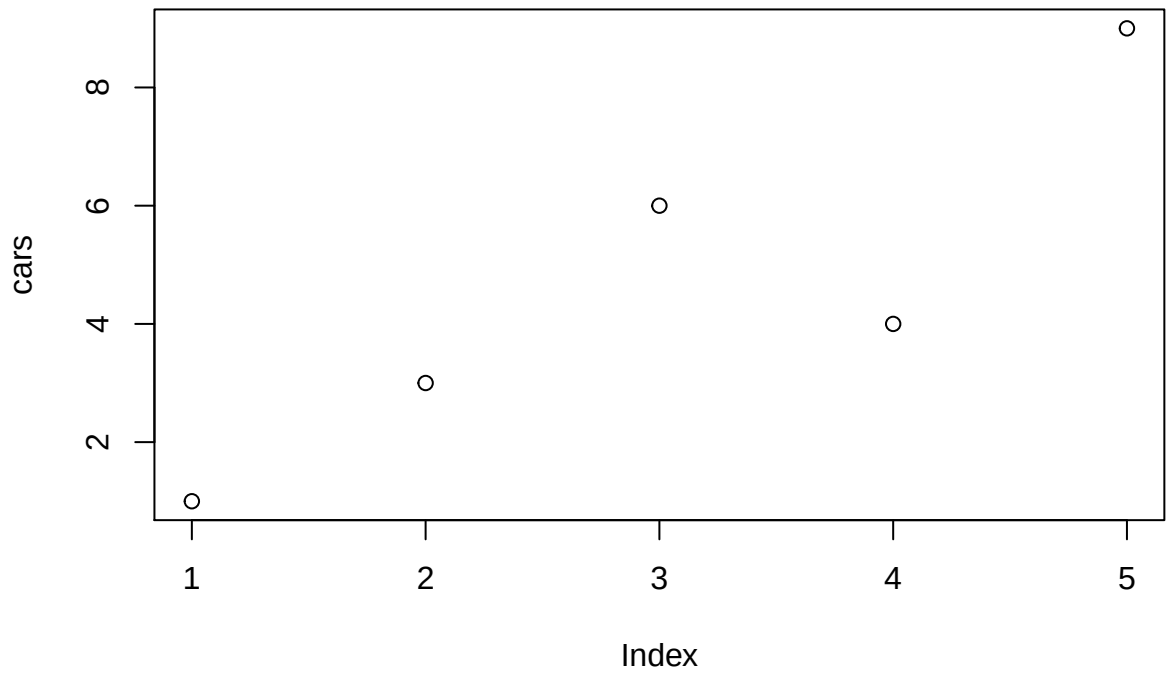
26.2 Producing Simple Graphs with R

The credit of this section goes to Dr. Frank McCown (Frank McCown (2006)).

26.2.1 Line Charts

```
# Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

# Graph the cars vector with all defaults
plot(cars)
```



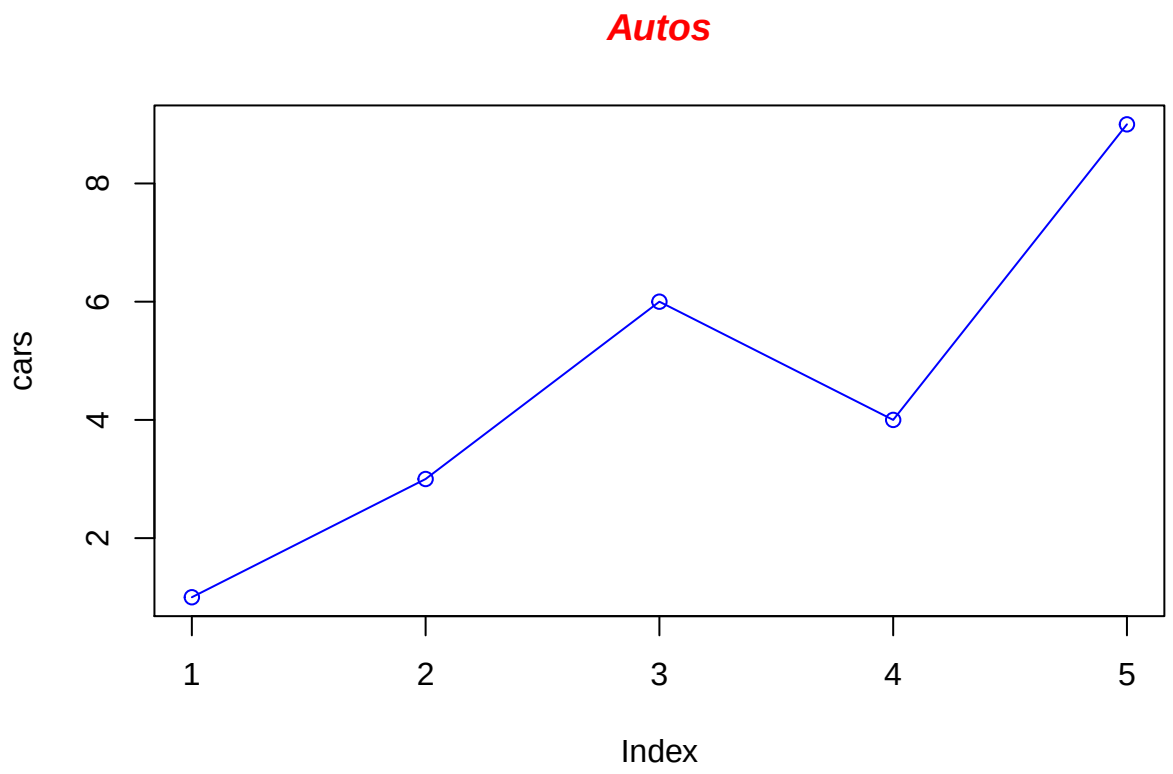

```
?plot
```

Let's add a title, a line to connect the points, and some color:

```
# Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

# Graph cars using blue points overlayed by a line
plot(cars, type="o", col="blue")

# Create a title with a red, bold/italic font
title(main="Autos", col.main="red", font.main=4)
```



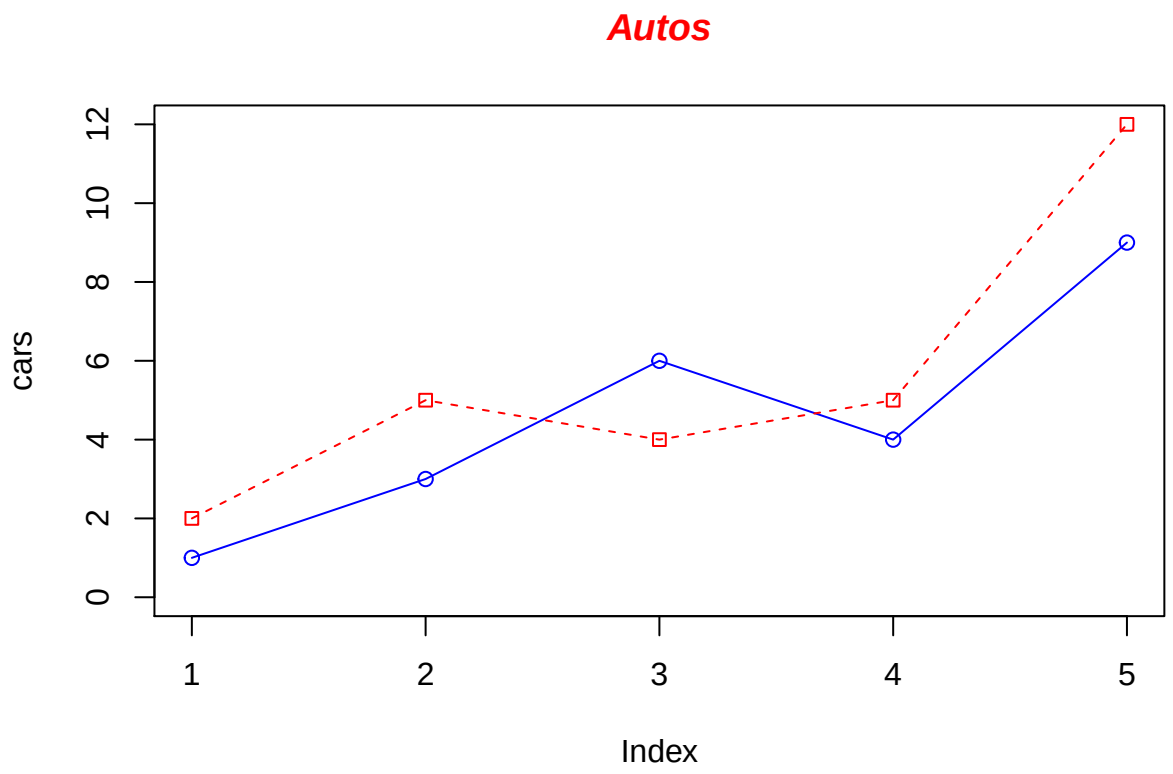
Now let's add a red line for trucks and specify the y-axis range directly so it will be large enough to fit the truck data:

```
# Define 2 vectors
cars <- c(1, 3, 6, 4, 9)
trucks <- c(2, 5, 4, 5, 12)

# Graph cars using a y axis that ranges from 0 to 12
plot(cars, type="o", col="blue", ylim=c(0,12))

# Graph trucks with red dashed line and square points
lines(trucks, type="o", pch=22, lty=2, col="red")

# Create a title with a red, bold/italic font
title(main="Autos", col.main="red", font.main=4)
```


Autos
cars
0
2
4
6
8
10
12
1
2
3
4
5
Index

Bibliography

Deitel, H. M., Deitel, P. J., and McPhie, D. C. (2001). *Perl How to Program*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Frank McCown (2006). *Producing Simple Graphs with R*. Harding University, Searcy, AR.

Torvalds, L. (2015). Linux kernel release 4.x.

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.3.