UNIX and Bash Fundamentals

Rustie Lin

October 31, 2017

Introduction: Why UNIX?

It is important to understand file structure and how to use UNIX, especially when dealing with large R projects. It is imperative that every statistics student, or anyone interested in data anlysis, be competent in navigating and using the UNIX bash shell because they will likely be spending quite a significant amount of time in the terminal.

Chances are, individuals in the data analysis field will most likely be working in UNIX most of the time, because of all the data manipulation tools that are available in the UNIX operating system. We will cover some of the most basic UNIX commands, and how to use them, in this blog post. UNIX commands exist for various use cases, and pretty much anything you would want a computer to do can be calculated by stringing together a series of UNIX commands, as we will see in the examples below. In the examples, most commands presented are mainly useful for data preparation, filtering, and formatting.

UNIX is a computer operating system which is capable of handling operations from multiple users at the the same time. Development for UNIX started 48 years ago, in 1969, and UNIX to this day still remains relevant because it is the basis for the majority of today's systems. Android, Mac, and Linux machines are all UNIX-based systems, for example.

Some Useful Commands

The following are some useful commands to learn in order to manupulate data sets effectively.

sed, the stream editor

One useful command that is especially relevant to data analysis is sed, the stream editor. sed is useful in the data preparation phase of data analysis. sed allows one to perform basic text transformations on an input stream.

sed is a very powerful utility that offers many different configurations and options, to maximize the number of use cases it can handle. One of the most useful configurations to use with sed is the 'substitute' option. As one can imagine, this could allow someone to manipulate data sets pretty efficiently.Because it is a UNIX command, it can be chained together with other commands to create some impressive workflows. For example, one could use sed to change the delimiter for a huge data set, a task that would be a lot more difficult with just your average text editor.

Consider the following data set, which is delimited by the colon character ':' I have stored this data into a file called `data/colon.data' for reference.

```
# a sample data set in 'data/colon.data'
sed:10:A
awk:20:B
ls:30:C
grep:40:D
cd:50:E
```

We could easily convert this data into a commonly used CSV (Comma Separated Values) format. I will present the command first, and then walk through the syntax and what each option/configuration is doing.

```
# sample sed command, substituting all colons with commas
sed 's/:/,/g' data/colon.data
```

```
## sed,10,B
## awk,20,A
## ls,30,C
## grep,40,E
## cd,50,D
```

The word 'sed' invokes the sed command, and lets UNIX (specifically the bash shell) know that we want to call sed, and that the following strings will be arguments and options for the sed command. The following 's' indicates that we want to perform a sed substitute. The next two symbols separated by the forward slash represent what we want to substitute. The format so far looks like this: 's/[from]/[to]'. sed will search an input (standard input or a file) for any string that matches [from], and then replace it with [to]. The 'g' indicates that we want to substitute all instances. Without the 'g' parameter, sed would only substitute the first match.

After running the above command, the result is the following. I have placed the output in a file called 'data/comma.data' for reference.

awk, text processor

It's actually hard to classify awk, since it can do so much. Probably instead of text processor, it should be called a programming language. In fact, its syntax is similar to what one would imagine a full fledged programming language to have.

The easiest way to understand what awk can do is to see it in action. Given the previous data set 'data/colon.data', we can perform some column operations as follows:

```
# sample awk command, printing first and third columns
awk -F':' '{print $1 "\t" $3}' data/colon.data
```

```
## sed  B
## awk  A
## ls   C
## grep E
## cd   D
```

The example above, the word 'awk' calls the awk UNIX program, as in the previous section. The flag 'F' indicates that the following string will be the delimiter by which we parse our input. In this case, we specified -F':'which means that delimiter is a colon':'. It might be useful to note that the single quotes in the bash shell indicates that the enclosed text is strongly typed, and will not be evaluated. The following argument is the awk program script itself, passed in as a string. The script'{print $1 "" $3}' is read as such: print the first and third columns of the input, with the prespecified delimiter. Print a tab character between the first and third columns. Indeed, after awk executes, the output is the following:

Tab delimited data is generally easier to read for the average person, and is also a commonly used data format. Such data is stored in TSV file formats (Tab-separated values), and are commonly used in database tables or spreadsheet data, and are also used to exchange information between databases. TSV is an alternative to the common CSV file format because commas sometimes need to be escaped, while tabs do not.

Next, we can leverage the power of UNIX command chaining. Say we want to get the previous table, but sorted alphabetically. We would perform the following chain of commands:

```
# sample pipe from awk to sort
awk -F':' '{print $1 "\t" $3}' data/colon.data | sort
```

```
## awk  A
## cd   D
## grep E
## ls   C
## sed  B
```

The '|' character is called a pipe. What the pipe operator does is take the output of the awk command, and 'pipe' it to the input of the sort command. Using this method of awk'ing and then sorting is useful in this case because we do not want to deal with the entire data set. awk reduces the table size because we are only interested in two of the three columns. sort then sorts each line of the output alphabetically. What if we want more flexibility with our sorting? What if we want to sort by a column, and display the entire table?

sort, sort command

Luckily, the sort command in standard UNIX is very powerful, and comes prebuilt with column filtering functionality. For example, if we wanted to sort alphabetically for the entire data set, we would execute the following command:

```
# sample sort
sort -t':' data/colon.data
```

```
## awk:20:A
## cd:50:D
## grep:40:E
## ls:30:C
## sed:10:B
```

In the above code sample, we are calling the sort command on our data set. The 't' flag indicates that we want a custom delimiter, which we set to the colon character ':'. By default, sort will try to sort alphabetically for each line (from the first column), and that works for us in this case because we are sorting by the first column.

Say that we want to sort by the third column instead. This is easy in sort, because it provides the 'k' flag, which defines the field (key) that we are sorting by. Since we are sorting by the third column, we would invoke the options '-k3'. The full command is as follows:

```
# sample sort by third field
sort -t':' -k3 data/colon.data
```

```
## awk:20:A
## sed:10:B
## ls:30:C
## cd:50:D
## grep:40:E
```

As you can see, our lines are now sorted by the third column, which contains all the capital letters.

cut, extracting columns

Previously, we showed that we could extract columns with the 'awk' command. We can also do the same thing in less code using the 'cut' command. If we want to cut the second column from our input data, we would perform the following command:

```
# sample cut third field
cut -d':' -f 3 data/colon.data
```

```
## B
## A
## C
## E
## D
```

Again, let us step through the command to understand each of the parameters. The ‘d’ flag lets us specify a custom delimiter (the default is the tab character). The ‘f’ flag allows us to specify the fields, or columns, we are interested in cutting from the source data. In the example above, ‘-f 3’ tells cut that we want to cut out the third column from ‘data/colon.data’.

The ‘f’ flag also supports specifying lists of columns. Imagine we want to cut out columns 2 and 3 from the input data. We could rewrite our command very easily:

```
# sample cut fields 2 to 3 inclusive
cut -d':' -f 2-3 data/colon.data
```

```
## 10:B
## 20:A
## 30:C
## 40:E
## 50:D
```

Some specifics about the cut command, here are some options we can leverage to specify exacly which colums we want to cut.

N cuts the *N*th byte, character, or field, and is 1 indexed

N- cuts from the *N*th byte, character, or field, to the end of the line

N-*M* cuts from the *N*th to the *M*th byte, character, or field and is inclusive

-*M* cuts from the first to the *M*th byte, character, or field

Bringing Everything Together

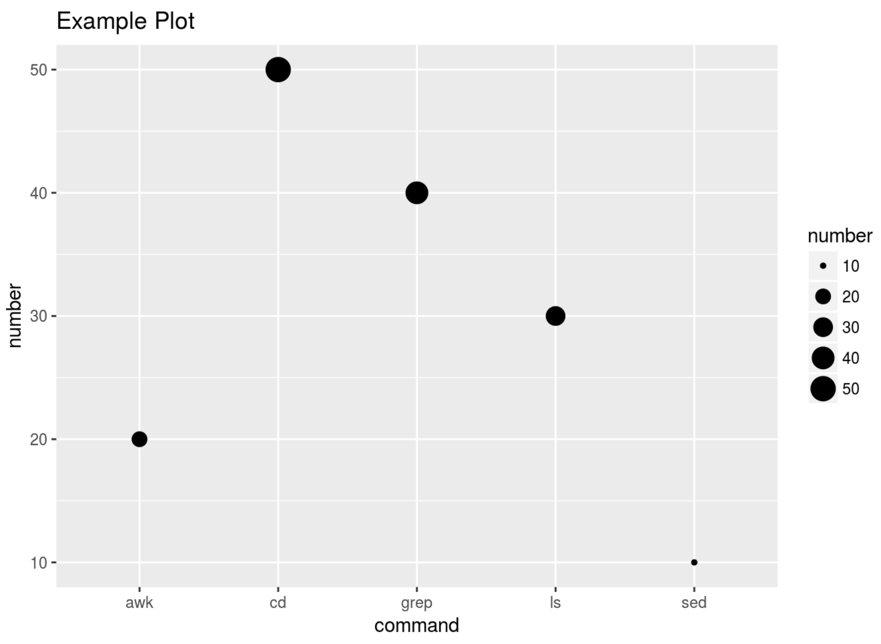
The real power of UNIX is the ability to chain various commands together, and getting everything to work together to calculate a desired result. Consider the first code sample, in which we converted a table from colon seperated to be comma separated, CSV, which is a much more standard data format. Just to exemplify the power of UNIX, let us chain together this sed command with some R commands, and eventually display a graph by ggplot.

First, some UNIX commands to save a CSV that we’ll be working with.

```
# save into a separate CSV
sed 's/:/,/g' data/colon.data > data/comma.data

# give our CSV some column labels
echo 'command,number,letter' | cat - data/comma.data > data/labeled.csv
```

Next, we’ll read in the CSV with R, and perform some data analysis. In this case, we’ll be drawing a sample plot with ggplot2.

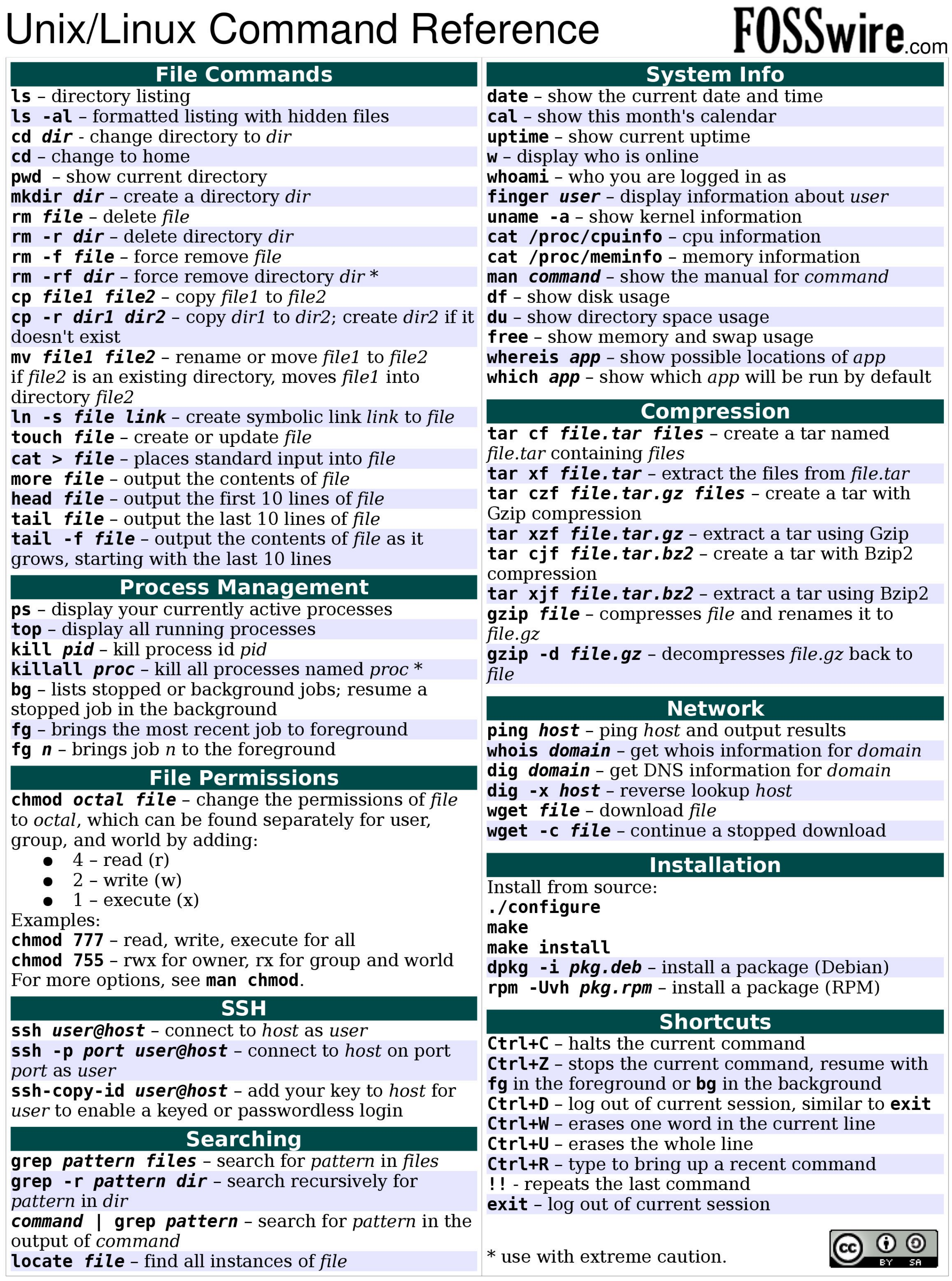
```
# import the data into R
dat <- read.csv('./data/labeled.csv')

# plot the data
library(ggplot2)
p <- ggplot(dat, aes(command, number))
p + ggtitle("Example Plot") + geom_point(aes(size = number))
```



Conclusions

Of course we only mentioned a couple commands here. There are thousands of awesome UNIX commands and programs that are especially useful for those who are interested in data analysis, and also those who are power users of computers. It is important for all statistics majors to understand how to interact with UNIX and the bash shell, because it streamlines workflow. It is also nice to understand the fundamental technology that many programs and operating systems today are based off of. Despite being nearly 50 years old, UNIX is only increasingly relevant in the modern day.

Here is a UNIX command reference list:



Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.

source: http://www.gauravkgupta.com/6-best-linuxunix-command-cheat-sheet/

References

UNIX reference sheet

Tab-separated values

UNIX logo

Sed - An Introduction and Tutorial

Linux cut command

How to Write AWK Commands and Scripts

Awk - An Introduction and Tutorial

Grep