MONASH University

FIT1043 Introduction to data science - S2 2022

Week 10 tutorial solution

In this tutorial we investigate tools for managing and processing large quantities of data.

Manipulating large files with shell commands

In this part of the tutorial we investigate simple command line tools available in the Unix shell that allow us to manipulate large data files **without needing to ever load them fully into memory**.

Starting a shell

In the lab machines: Option 1: Within Windows open the *Linux Student Image - Player*, which will start a Linux virtual machine. In the virtual machine you will need to log-in using your Authcate again. Then you can open a terminal. Option 2: Restart the machine, press and hold F12, when a screen appears asking you to choose operating system, choose *Monash GNU/Linux*. After it loads login, then run a terminal.

Your own Windows machine: If you would like to complete the lab on your own machine and it happens to be running Windows, you can install Cygwin, which allows you to run a Unix-like shell inside Windows. Note Cygwin is not ideal!

Your own Linux machine: Click on the black square at the top left of the screen to open up a terminal.

Your own MacOSX machine: Go to Applications -> Utilities -> Terminal.

With the shell running, you will now need to download the directory of files and scripts. The complication here is if you are running the Linux Student Image on the labs. You will need to run a browser under the Linux image in order to access the Moodle page.

Running the shell

Once you have Unix shell type the following commands in order to familiarise yourself with basic shell:

ls
pwd
cd ..
pwd
cd

Practice 1 : What do all of the above commands do?

ls - lists the files contained in the current directory

pwd -path to the working directory

cd .. - One directory up or parent directory

cd - Go to the home directory

Download the bundle tutorial_data.zip from Moodle and then copy it from the Downloads directory to the current directory:

```
cp Downloads/tutorial_data_shell.zip .
```

The dot "." notation is shorthand for the current directory, and since we didn't specify a different name for the file, it will be copied with the same name.

Uncompress the zip file to have a look at the contents. You can do that from the command line using:

Uncompress the zip file to have a look at the contents. You can do that from the command line using:

```
unzip tutorial_data_shell.zip
```

And then change directory to the books directory:

```
cd tutorial_data/books
```

We'll now have a look at "less", which is a very simple program for viewing (but not editing) text files.

```
less book1.txt
```

What book is it? Use the following commands to do basic navigation in file:

```
[up/down] - move one line the file
[space]   - move down a whole page
q         - to quit
```

If you ever get into trouble while using *less*, just hit **[Control]+c** to kill the program.

Here are some other basic less commands that allow you to skip to the end of a file or search for a string in the file.

```
[shift]+g - skip to end of file
/keyword  - search for the first occurrence of "keyword" within the file
/         - find the next occurrence of the keyword
```

```
Practice 2 : In which chapter did Alice go to a "Mad Tea-Party"?
CHAPTER VII
```

We can find out quickly what their titles are by running a "grep" command which checks each line of a file to see if it matches a particular pattern and returns the line if it does. Try the following two commands:

grep "Title" book2.txt
grep "Title" book*.txt

Practice 3: Have a look at the other books in the folder. Who are their Authors?

grep "Author" book*.txt

book1.txt:Author: Lewis Carroll
book2.txt:Author: James Joyce
book3.txt:Author: Leo Tolstoy
book4.txt:Author: Arthur Conan Doyle
book5.txt:Author: Oscar Wilde
book6.txt:Author: Franz Kafka
book7.txt:Author: Bram Stoker
book8.txt:Author: Charles Dickens
book9.txt:Author: Jane Austen
book10.txt:Author: Rudyard Kipling

In the second command the asterisk "*"is a wild-card and matches against all the files book1.txt through to book10.txt, and the results is the same as if we called grep on each of those files in turn.

Practice 4: Now try to display the Author names for books 1 to 5 only? (Hint: Use bracketed wildcard characters)

grep "Author" book[1-5].txt

The third book is really long. It's Tolstoy's "War and Peace". Use the word count command "wc" to find out how long it is:

```
wc book3.txt
```

The command should return three numbers. Can you work out what the numbers mean? Google "wc unix" to check.

Practice 5: Write the shell command to list the word count of all the books at once?

wc book*.txt

Another command line tool that is very useful is "cat", which does nothing more than load a file and output it to the terminal:

```
cat book1.txt
```

The cat command becomes particularly useful when combined with a Unix pipe operator "|". A pipe is used to connect programs using the syntax:

```
program1 | program2
```

All the pipe operator does is **redirect the output of one program to be the input of another program**. So in the case above, it grabs the the text that program1 tries to print out to the screen and instead passes it as input to program2 (as though the user had typed it all in on the keyboard). Most shell commands are designed to be able to take input from a pipe and output to a pipe. We call them pipes, because they allow us to control the flow of data, just like a real pipe controls the flow of water.

The reason pipes are interesting to us when dealing with big files in data science applications is they allow us to **work on data as a stream, rather than needing to load the data into memory in order to view or use it**. To understand what this means, let's have a look at the rather large file from last week's tutorial. First change to the "data" directory and show its contents:

```
cd ../data
ls
```

It should contain a 12MB file called "hourly_44201_2014-06.csv.gz". Don't decompress the file. We'll do that on the pipe using the "gunzip" application:

```
cat hourly_44201_2014-06.csv.gz | gunzip | less
```

Notice how fast it loads! The reason is that only the start of the file is being accessed. The display only shows the start of the file and doesn't need to wait until the whole CSV file is loaded in order to display content. -- In fact the whole CSV file isn't ever loaded if quit *less* without ever scrolling to the end of the file!

The pipe is buffered (with a buffer size of usually around 64KB), which means that each program on the pipe will only produce new output (the next 64KB of data) when the previous output (on the buffer) has been emptied by the subsequent program. So unless any one of the programs needs to load all the data into memory for some reason, the overall command line will execute **without ever loading the data all into memory at once**. Each subsequent program on the pipe simply accesses the data it needs as and when it is ready to process it.

We don't need to finish the pipe with the less command. If we just wanted to know how many lines there were in a file, we could instead pipe the results to the word count "wc" command:

```
cat hourly_44201_2014-06.csv.gz | gunzip | wc
```

Practice 6 : It takes much longer to execute this command line than the previous one (that ended with less). Explain why is that?

To get the word count we need to load whole dataset to the memory. Less function will print only 1st few lines of the files. Therefore to get the output of the less function it is not necessary to load whole dataset to the memory.

If we would like to see just the start or end of the file (without loading it interactively in *less*) we could use the "head" and "tail" commands, similar to R:

```
cat hourly_44201_2014-06.csv.gz | gunzip | head
cat hourly_44201_2014-06.csv.gz | gunzip | tail
```

Notice how much longer the command line with *tail* takes to execute than does the one with *head*!

If all we are wanting to do with the file is extract the first say 1000 lines of the file, then we could use the head command to do this by appending the flag -n1000. We could then save these lines to a file rather than printing them out on the screen by doing a redirect with the ">" symbol:

```
cat hourly_44201_2014-06.csv.gz | gunzip | head -n1000 > first_1000_lines.txt
```

So a pipe "|" passes data to another program, while a redirect ">" saves data to a file.

Practice 7 : Since you are a bit familiar with pipes, why do you think it is useful for big data processing?

Pipe operators make a direct connection between programs. This direct connection between commands/ programs/ processes allows them to operate simultaneously and permits data to be transferred between them continuously rather than having to pass it through temporary text files or through the display screen.

A very handy tool is "awk" which is a program for processing a text file one line at a time. When using awk all we need to do is (i) specify the delimiter it should use to break up each line (into columns) and (ii) tell it what to do with each line. For example, we can use awk to select a subset of the columns as follows:

```
cat hourly_44201_2014-06.csv.gz | gunzip | awk -F',' '{print $6,$7,$14}' | less
```

Here we have used the flag -F',' to tell awk that the columns are separated by commas, not whitespace (which it assumes by default). And we have passed it the instruction {print $6,$7,$14}, which tells it to print the value in columns 6, 7 and 14 for each line of input it sees.

If we are happy with the new table we have produced, we could save it to a file, but imagine we would only like to save a small sample of the file, say five hundred lines starting from line 1001. We can tell awk to apply the print command only to those lines as follows:

```
cat hourly_44201_2014-06.csv.gz | gunzip | awk -F',' 'NR>1000 && NR<=1500 {print $6,$7,$14}' > 3columns_500rows.txt
```

Often we'd like a random sample of the data. Again this is very simple awk. If we want to sample 1% of the rows, we can use the random number function "rand()" to produce a value in the range 0 to 1 and only output a line if the value is less than 1/100:

```
cat hourly_44201_2014-06.csv.gz | gunzip | awk -F',' 'rand()<1/100 {print $6,$7,$14}' | less
```

We can also select only rows that have a particular value in one of the columns. For instance, we could select measurements around the Los Angeles area (34.0522, -118.2437) by restricting the latitude and longitude coordinates, as follows:

```
cat hourly_44201_2014-06.csv.gz | gunzip | awk -F',' '$6>=34 && $6<=34.5 && $7>=-118.5 && $7<=-118 {print $6,$7,$14}' | less
```

There are many other programs available in the Unix shell. Particularly useful is the "sort" command, which can sort a column of values either lexicographically (i.e. alphabetically) or numerically (by using the -n flag):

```
cat hourly_44201_2014-06.csv.gz | gunzip | awk -F',' '{print $14}' | sort -n | less
```

Be patient! Notice how long it takes for the values to appear? That's because the **whole file has to be loaded** into memory before they can be sorted. The sort operation itself can also take some time if the list of numbers is very large.

If you have multiple columns in your output, you can tell the *sort* command to sort on a particular column by using the -k flag, (*sort -k2,2*will sort based on the second column, while *sort -k2,4* will sort based on column 2 then 3 then 4). You can find out more about any unix command by either googling it or opening its man page:

```
man sort
```

The man page has the same controls as *less*.

Now let's have a look at a much bigger file: hourly_WIND_2015.csv.gz, which contains wind data readings across the US for all of 2015. It is 87MB compressed and 2.5GB uncompressed.

This file is compressed using the gzip format, so again we'll use the gunzip command to decompress it, but instead of generating a 2.5GB file, we'll output the data to a pipe. We tell the gunzip program to do this by using the "-c" flag. We'll then take the output and view it in less.

```
gunzip -c hourly_WIND_2015.csv.gz | less
```

How many lines does this file contain? Use wc to find out.

Let's extract the wind measurements ("Sample Measurement" column) for the state of California (i.e. where the column "State Name" has the value "California") and save them to a text file.

Note that the strings in the State Name column are wrapped in double quotes characters ("), so to match the value "California" (quotes included) we need to escape the double quotes using the backslash character as follows: "\"California\"". We also want to print out the column names to the file, so we'll pass through the first line of the file by checking if NR==1.

```
gunzip -c hourly_WIND_2015.csv.gz | awk -F',' '$22=="\"California\"" ||
NR==1 {print $6,$7,$14}' > california_wind.txt
```

This could take a while! The file you are creating is quite big...

Now, we want to move the file back to where our R can read and process it. Let's load and visualise it with R.And then run some commands:

```
df <- read.table('california_wind.txt', header = TRUE)
head(df)
tail(df)
summary(df$Sample.Measurement)
```

Challenge [Practice 5]: From the file hourly_WIND_2015.csv.gz, extract the latitude and longitude values for locations with extremely high wind measurements, say greater than 30 Knots (about 55 km/h), into a new file. Return to R, load in the data and plot the latitude and longitude values against each other as a scatter plot to see the locations where the extreme measurements were recorded

Command in bash:

```
gunzip -c hourly_WIND_2015.csv.gz | awk -F',' '$14>30 || NR==1 {print $6,$7,$14}'>lati.txt
```

Locate the exported ' lati.txt' file in your folder, then set you R Studio directory to the same location using R code

```
setwd('put_your_directory_path_here')
```

Then, open a new R script (hint: new file) from R studio and run the following code:

```
df<- read.table('lati.txt',header=TRUE)
install.packages("ggplot2")
library(ggplot2)
ggplot(data = df, mapping = aes(x = Latitude, y =Longitude )) + geom_point() + geom_smooth(formula = y ~ x, method = "lm") )
```

Last modified: Wednesday, 5 October 2022, 10:24 PM

◄ tutorial_data.tar.gz

Jump to...

Tutorial recording ►