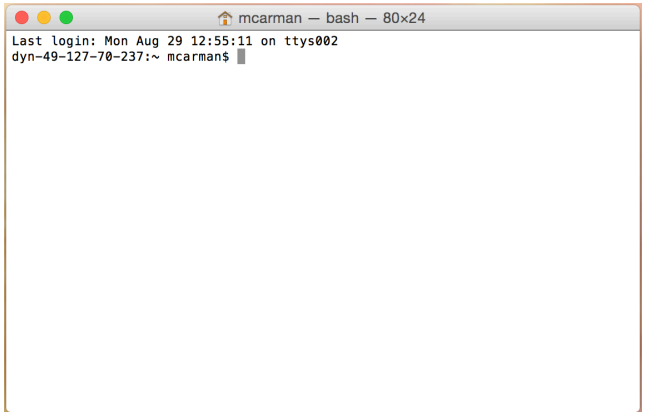
A very brief Introduction to Unix Shell Commands for Data Science

Slides by: Mark Carman

What is a Unix Shell?

- Command line interface to a Unix computer
  - different shells have been around since the 70s
- Why are shells interesting for Data Scientists?
  - Provide powerful & easy way to manipulate large data files
  - And move data around a network
- Available on most Unix based operating systems
  - Linux
  - Mac OSX (BSD based)



Getting started

Installing a Shell on Windows:

https://www.cygwin.com/

Running a Shell:

- In Linux, click on the black square at the top left of the screen.
- In MacOSX, go to Applications -> Utilities -> Terminal.

Navigating the Filesystem

- Change directory:
  cd [destination]
  cd /my/favourite/place

- Special cases:
  cd .. ← takes you up one directory
  cd ← without argument, takes you to home directory

- List files in the current directory:
  ls

- Copy files from one location to another:
  cp [source] [destination]
  cp /Downloads/myfile .

Reading a Text File

- Open a text file for reading using *less*:

  less myfile.txt

- Navigate within the text file using

  [up/down] ← move one line the file

  [space] ← move down a whole page

  q ← quit

  [shift]+g ← skip to end of file

  /keyword ← search for the first occurrence of “keyword”

  / ← find the next occurrence of keyword

Some useful commands

- Count the number of words/lines in a file

  wc myfile.txt

- Find lines in a file containing a keyword

  grep "elephant" myfile.txt

- Print the first/last few lines of a file

  head myfile.txt

  tail myfile.txt

- Print the contents of a file to the screen

  cat myfile.txt

Flags and Arguments

Many programs take flags and command line arguments that modify their behaviour, for example:

- Sort the contents of a file lexicographically (alphabetically)

  sort myfile.txt

- Sort the lines of a file by numeric value

  sort –n myfile.txt

- Sort the data by column one, then column two and finally column three:

  sort –k1,3 myfile.txt

Pipes

Sometimes we'd like the output of one program to be used as the input to another.

- Doing this is super easy in the shell. We just use the pipe operator "|"

  program1 | program2
- We can chain as many programs together as we want, for example:

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

The pipe is buffered

- each program in the list only generates data **as it is needed** by the next stage in the pipeline.
- thus memory requirement for processing the data is limited
- crucial for **scaling up processing to enormous data files**.

Redirects

If we want to save the results in a file rather than pipe them to a new program:

- just change the pipe operator “|” to be a greater than symbol “>” and provide a filename:

```
cat hourly_44201_2014-06.csv.gz | gunzip > newFile.txt
```

Wildcards

- Some unix commands can take multiple files as input, for example:

  cat myfile1.txt myfile2.txt

- In order to avoid listing large number of files, we can use the wildcard syntax to specify all files in a directory with a certain pattern, e.g.:

  cat myfile*.txt

awk

In the tutorial, we'll have a look at a powerful command for processing text files one line at a time called awk

- awk syntax:

  awk '[select line?] {do something}'

- example

  awk 'rand()<1/100 {print $6,$7,$14}'

- Since awk processes data one line a time, **it can scale up to massive datasets!**

scripts & parallel execution

- In the tutorial, we also see how scripts written in Python can be used as programs in the shell.

- And find out how to run programs in parallel using the ampersand notation:

  myprogram &

End Of Introduction

- We'll be experimenting with the Unix shell in next week's tutorial
- There are MANY excellent shell tutorials online if you'd like to learn more!