Unlocking the power of the Command Line:

I.e. – Why I should continue to learn about and use the Unix shell

Basic familiarity with the Bash shell is assumed and required for this class

Based on the book *Data Science at the Command Line* by Jeroen H.M. Janssens (O'Reilly). Copyright 2015 Jeroen H.M. Janssens, 978-1-491-94785-2.

- Ebook available to Curtin Students through library
- Ebook also available online: http://datascienceatthecommandline.com/
- Github: https://github.com/jeroenjanssens/data-science-at-the-command-line
- Accompanying docker image: datascienceworkshops/data-science-at-the-command-line

Extremely handy tools you should use:

- Great summary of commands: https://github.com/Idnan/bash-guide
- Shell syntax checker: https://www.shellcheck.net/
- Simpler alternative to git: https://gitless.com/

Full software carpentry BASH courses:

- https://swcarpentry.github.io/shell-novice/
- https://swcarpentry.github.io/shell-extras/

If you want to dive deeper into BASH, check these out:

- http://tldp.org/LDP/abs/html/index.html
- https://github.com/jlevy/the-art-of-command-line
- https://github.com/alebcay/awesome-shell

1. Preparation

- Access to the Bash shell is required through any of the following:
  - Mac OSX terminal
  - Linux terminal
  - Git Bash
  - MobaXterm
  - Windows Subsystem for Linux (Windows 10 only)
  - The Data Science Toolbox virtual machine
- If you don't already have one of the above, MobaXterm is the easiest to install
- If you're Docker savvy and interested in following along with *Data Science at the Command Line* later, a docker image has been setup with every tool used in the book

2. Introduction

- Data science is OSEMN (pronounced awesome) – 1) Obtaining data, (2) Scrubbing data, (3) Exploring data, (4) Modelling data, and (5) iNterpreting data. Sounds a lot like research!

- Becoming proficient with the command line will make you more efficient and productive at data science (and using computers more generally)
- Command line has several advantages. It is valuable to be able to identify when it is going to be the best tool for the job, and when it isn't.
- Command line is agile
  - read-eval-print-loop often more convenient than the edit-compile-run-debug cycle associated with scripts
  - very close to the filesystem
  - generally high action-to-keystroke ratio
- Command line is augmenting – integrates well with other technologies
  - can use command line tools within Python and R scripts
  - can turn Python or R scripts into command line tools
  - there are command line tools which easily work with various databases and file types
- Command line is scalable
  - everything on the command line can be automated with scripts and tools
  - therefore everything is easily repeated, expanded or scheduled
  - command line tools can be parallelised (gnu parallel)
- Command line is extensible
  - command line tools can work together for additional flexibility
  - new tools are being developed all the time
  - the tools can be written in any language
  - you can create your own tools
- Command line is ubiquitous
  - comes with all Unix-like operating systems including Linux and Mac OS X
  - the vast majority of supercomputers, cloud computing services, and servers are based on the command line
  - it has been around for four decades and it isn't going away

3. Comparison of scripting languages

- Repetitive tasks should be made into executable scripts to simplify your life
- top-words example
  - scripts read input and list the n most repeated words and their number of occurrences
  - have a look at each script
  - each script starts with a shebang which identifies how it should be executed
  - compare the length of each script:
*wc top-words.**
- shell scripts allow you to do simple things with minimal effort

- complex things are easier to do in more sophisticated languages with extensive libraries of existing functions
- you can mix and match languages by executing other scripts from within a shell script

4. Customisation of .bashrc and creating your own command line tools

- Meet your new best friend: .bashrc, he lives in your home directory
- Any command you add to your .bashrc file will be automatically run every time you open a new terminal
    - If you do not have an existing .bashrc file in your home directory, copy ours there
    - If you do have an existing .bashrc file, you can add the commands from our .bashrc file to it.
- Aliases allow you to redefine commands and create your own custom short-cuts for frequently used commands
- The PATH environmental variable specifies which directories are searched to find executables
    - create a new directory for your executables:
      mkdir ~/bin
- You can turn any script into a command which is executable from anywhere
    - Make the script executable:
      chmod u+x top-words.sh
    - Move it into a PATH directory:
      mv top-words.sh ~/bin
    - Run script from anywhere:
      top-words.sh random.txt

5. Some essential tools for preliminary data analysis and file manipulation

- wildcards (and regular expressions if you're hard-core)
    - List all files with 4 letter filenames beginning with t and any extension:
      *ls t???.**
- sort
    - Merge (m) the unique (u) lines in numeric order (n) of multiple files:
      sort -n -m -u file1 file2 -o file3
- pipes and redirects
    - Count the number of lines in each txt file and sort output numerically
      *wc -l *.txt | sort -n*
    - Execute script and append output to logfile
      ./file.sh >> file.log
- grep
    - Search file for the case insensitive (-i) word and echo the 3 following lines (-A)
      grep -A 3 -i "word" file

- find
    - Find all mp3 files in any directory

      ```
      find / -name ‘*.mp3’
      ```

    - Find all errors in log files below a specific directory

      ```
      find /path/to/directory -name ‘*.log’ -exec grep -i error {} \;
      ```

- sed
    - Substitute a specified phrase with another

      ```
      sed s/old/new/ file1 > file2
      ```

    - Starting with 21st line, extract every hundredth line from a line (-i overwrite original)

      ```
      sed -n '21~100p' file1 > file2
      sed -n -i '21~100p' file
      ```

- awk
    - Print out the first and third columns of a file

      ```
      awk ‘{print $1,$3}’ file
      ```

    - Print the first column squared

      ```
      awk ‘{print $1*$1}’ file
      ```

- loops
    - Run analysis script on every .in input file

      ```
      for file in *.in
      do
              ./analyse.py $file
      done
      ```

- conditionals and tests
    - If input file is newer than output, rerun analysis

      ```
      if [[ input -nt output ]]
      then
              ./analyse.py input
      fi
      ```