Introduction to Unix shell

- 2015/16 Part III Systems Biology SEB module
- 10 Feb 2016, 10:00-13:00
- Bioinformatics Training Room, Craik-Marshall Building, Downing Site
- Avazeh Ghanbarian, Alexey Morgunov

Introduction

Unix shell is a command line interpreter that provides a user interface for directing the operation of the computer by entering commands as text for a command line interpreter to execute, or by creating text scripts of one or more such commands. In plain English, it is a powerful way of telling your computer what to do. You can read more about the history of Unix shell here http://www.softpanorama.org/People/Shell_giants/introduction.shtml.

Developing skills for coding in any language consists of the following components:

- *Logic* - understanding the syntax, how commands and scripts are structured and how components fit together. This is something one has to learn.
- *Awareness* - knowing what commands, methods and tricks exist and what they can be used for. This is like checking your inventory of LEGO bricks - you need to know what you have in order to start thinking how to put them together to build what you want.
- *Practice* - and a lot of practice. Learning how to combine the bricks together to solve increasingly more complex problems is best achieved through continuous practice.
- *Google and Stack Overflow http://stackoverflow.com/* - what coding really is about. It is likely that unless you are doing something very very novel, someone else has run into the same problem and has a solution. Find it and use it, don't reinvent the wheel. This is an important part of the learning and practice process.

In this tutorial we focus on explaining the *Logic* component and on building some *Awareness* about existing commands and methods in Unix shell. Finally, we give some exercises for *Practice* and leave it up to you to familiarise yourself with how to search for answers if you get stuck.

If you have previous experience with Unix shell. Skip to the *Exercises* Exercises.md section below and try your skills at it.

Basics

The syntax of commands:

```
[command] -[options] [file or folder]
```

N.B. The angular brackets [] do not need to be typed. They are used here as a placeholder of specific type (e.g. a filename).

Very useful starting points:

```
man [command] #manual entry for the command ('q' to exit)
which [command] #locate the program aliased to the command
whatis [command] #one-line description
apropos [keyword] #match commands with keyword in their man pages
ls #list files in the directory
ls -l #long information
ls -lh #human readable format
ls -lht #sort by time
ls -A #include hidden files
pwd #print working directory
cd [folder] #change directory into folder
cd ~ #change to home folder
cd .. #move up a directory
```

Working with files and directories:

```
mkdir [name] #create a new directory
cp [file1] [file2] #copy file1 to location file2
cp [file] . #copy file from its location to working directory
mv [file1] [file2] #move file1 to location file2, e.g. rename
rm [file] #delete file
rmdir [directory] #delete directory
```

Careful when using `rm` recursively (`rm -r`). It is better and safer to use `find` instead, e.g. to remove all files with `.pdf` as their extension in the current working directory: `find . -name '*.pdf' -delete` . (The star in `*.pdf` here means all files that end in `.pdf` .)

Working with text files

Viewing file contents:

```
clear #clear the terminal screen
cat [file] #outputs file contents in terminal window
less [file] #one page at a time, space for next, (q)uit
gedit [file] #open text editor, also 'emacs', 'vi'
head -N [file] #display top N lines of file
tail -N [file] #display bottom N lines of file
wc [file] #word, line, character and byte count
```

Sorting:

```
sort [file] #sort alphabetically/numerically each line from file
sort -u [file] #sort unique entries
sort -r [file] #print reverse order
uniq [file] #print only unique lines
uniq -c [file] #show number of times the line occurs before each line
```

Operations on lines:

```
rev [file] #reverse the characters in each line of file
cut -c2 [file] #cut 2nd character from each line
cut -c3-5 [file] #cut 3rd, 4th and 5th characters
cut -c3- [file] #cut from 3rd character until end of line
cut -d':' -f2,5 [file] #cut the 2nd and 5th fields delimited by semicolons
join [file1] [file2] #join corresponding columns into one
paste [file] #same as `cat` without any options
paste -d',' -s [file] #join all lines in file into one using the delimiter
paste - - < [file] #paste the data in file into two columns
paste -d',:' - - - < [file] #three columns, two different delimiters
paste -d',' [file1] [file2] #join two files by columns, c.f. `join`
paste -d'\n' [file1] [file2] #read lines in both files alternatively
```

`join` is a very useful tool - more tricks here http://www.albany.edu/~ig4895/join.htm.

Comparisons:

```
comm [file1] [file2] #outputs 3 columns: lines unique to file1, to file2, common
comm -12 [file1] [file2] #suppress output columns 1 and 2, i.e. show only common
diff [file1] [file2] #shows per line changes needed to make file1 into file2
sdiff [file1] [file2] #compare two files side-by-side
```

`diff` is another overloaded tool, check it out here http://www.computerhope.com/unix/udiff.htm.

`grep`:

```
grep [string] [file] #print lines in file containing string
grep 'multiple words' [file] #use quotes for phrases
grep -i [string] [file] #case-insensitive
grep -v [string] [file] #lines that DON'T match string
grep -n [string] [file] #show line numbers
grep -c [string] [file] #only total count of matching lines
```

Redirection & Pipes

To take keyboard input and put it into a file, we can use `cat > file1.txt`. Type as many lines as you like to put into the text file (press `<Enter>` to start a new line) and when done finish with `<CTRL+D>`.

To load the contents of the file, use `cat file1.txt`. To append the contents, e.g. taking contents of a different file `file2.txt` and adding them to the end of `file1.txt`, use `cat file2.txt >> file1.txt`.

To combine (i.e. to concatenate) two files, use `cat file1.txt file2.txt > long_file.txt`.

N.B. `>` overwrites existing files, `>>` only appends to the end.

To take input from `file1.txt`, sort it and output it as `file2.txt`:

```
sort < file1.txt > file2.txt #using redirection into command, then into file
cat file1.txt | sort > file2.txt #using | to pipe output as next input
```

Using piping, many commands can be joined together, e.g.:

```
cat file1.txt | cut -d',' -f2 | sort -u | wc -l #number of unique entries in second column (as delimited by commas) of file1.txt
```

Wildcards and Regular Expressions

To match none or more characters in a file name, a wildcard `*.pdf` can be used, as seen above. Some more examples of wildcards:

```
*ouse #any number or none: matches GRouse, House, Mouse and ouse
?ouse #only one character: matches House and Mouse
^mouse #only at the beginning of line
mouse$ #only at the end of line
```

Regular Expressions are sets of characters and/or metacharacters that match (or specify) patterns. It is a world of both wonder and pain. For a brief introduction, if you dare, see here http://www.tldp.org/LDP/abs/html/x17129.html.

License