Intermediate Unix

Notes by Kasper D. Hansen

Last Modified 4/25/2013

The Shell

Most people use Bash (Bourne Again Shell). This is a program running inside a terminal program.

There are many different shells, but most modern systems ship with bash per default.

One other shell is worth knowing about: 'sh' the most basic shell. Many configure scripts are written in sh.

Startup files / configurations

On Unix: `.bashrc` and `.bash_profile`. Hard to remember the difference, but roughly `.bash_profile` is for code that only should be run once.

On Mac it is called `.profile`. Different name, same beast.

I use a couple of highly convenient settings in my `.bashrc`

```
export PS1="\h:\w/> "
export LSCOLORS="gxxxxxxxxxxxxxxxxxxxx"
export HISTCONTROL=erasedups
export HISTSIZE=10000
shopt -s histappend
shopt -s cmdhist
bind '"\e[B": history-search-forward'
bind '"\e[A": history-search-backward'
```

The last two lines are awesome. The `\e[B` is what is being sent to the shell when I press up-arrow (or is it down?). It is bound to a command which searches the history based on what is already entered.

```
ls ~/
ls ~/Work
ls ~/bin
pwd
ls
```

We also have `.ssh/config` which is a very convenient file for `ssh`/`scp` shortcuts. Syntax is easy

```
Host enigma2.jhsph.edu e
  HostName enigma2.jhsph.edu
  User khansen
Host lore.ebalto.jhmi.edu lore
  HostName lore.ebalto.jhmi.edu
  User khansen
Host bitbucket.org bb
  Compression yes
  HostName bitbucket.org
  User hg
  ForwardX11 no
  ForwardX11Trusted no
Host *
  ForwardX11 yes
  ForwardX11Trusted yes
```

Differences between Linux and OS X

OS X is based in BSD, which has slight differences to Linux. For example, `sed -i` is different between the two OS. This can be confusing and irritating. This could in principle be 'fixed' by installing the GNU Linux tools on OS X.

Commands

Bash has a number of built in commands. Most of what we use (like `ls`) is other programs residing in `/bin` or perhaps `/usr/bin`.

Example: contrast the `time` built in bash command with `/usr/time` which had me confused for a while.

The way `ls` gets colors is by adding control sequences around the output.

```
ls -G
ls -G > tmp
less tmp
```

Piping

You should know about piping.

```
cat count.txt
cat -t count.txt
cut -f1 count.txt
cut -f1 -d' ' count.txt
cut -f1 count.txt | uniq -c
cut -f1 count.txt | sort | unic -c
```

Here, the really big computation is `sort` which is a specialized program.

Viewing files

`less` has a number of build in commands. You can search and jump to a specific percentage of the file, which can be really useful.

You probably know `tail`, I use `tail -f` a lot.

Redirection

What is the deal with the input and output streams? You have

```
stdin    Standard input
stdout   Standard outout
stderr   Standard error
```

There are also named pipes. I won't discuss those, but they can be very convenient for the right task.

```
bowtie.sh
bowtie.sh &> tmp.out
bowtie.sh 1> tmp.out
bowtie.sh 2> tmp.out
```

Background processes

Every code/command you run is a process on the machine. Most of the processes have a parent process, which is the shell you are running in.

First we look at processes and jobs inside a single shell.

A job can be in the foreground, in the background and suspended. Think of a switching between full-screen applications. Only one can be visible at any time, but the other apps are still running. You suspend a running app by ctrl-z.

You may be familiar with `command &` which runs command as a background process. Each jobs has a single number, which you can refer to like `%1` or `%emacs`.

Commands `jobs`, `fg`, `bg`.

```
ssh e
jobs
R
ctrl-z
jobs
fg
ctrl-z
emacs
ctrl-z
jobs
fg %2
... stop everything ...
./until.sh
ctrl-c
./until.sh >> until.out
ctrl-z
tail -f until.out
ctrl-c
bg
tail -f until.out
ctrl-c
jobs
```

This is extremely handy for doing multiple things at the same time.

Now, `jobs` only knows about jobs inside the specific shell process.

Show this. Discuss `ps` and `top`.

Bash programming

Can be frustrating, but worth learning. This is entirely a macro language.

Use `#!/bin/bash -e` which stops on error.

```
export VARIABLE
```

Tips

Go to the last directory `cd -`.

Learn how to do for loops on the command line

```
for f in $(/bin/ls *.fastq); do echo $f; head -1 $f; done
```

Learn `screen`.