Running Programs in UNIX

Outline

- Cmdline
- Running Programs in UNIX
- Capturing Output
- Using Pipes in UNIX to pass Input/Output

cmdline options in BASH

- `^` means "Control key"
- cancel a running application: `^C`
- end a session: `^D` (End of File message)
- Tab to try to autocomplete (applications, filenames, directories)
- while typing on cmdline - jump to end of line: `^E`
- get back to the beginning of line: `^A`
- Up and down keys cycle through history of commands
- Type `!!` to execute the last command
- Type `history` to see list of previous commands
- Type `!NUMBER` to excute cmd from that list

Foreground, Background

Jobs are run in the foreground by default

- `^Z` - suspend job
- `bg` - puts process in background
- type `fg` - puts process back in foreground
- `cmd &` launches puts process in background

```
$ emacs
^Z
[1]+  Stopped                 emacs
$ bg # job in background
[1]+ emacs &
$ fg
emacs
```

Copy files

cp to make a copy of a file or directory

```
$ cp one.txt two.txt# copy one file to another

$ mkdir books # make a directory
$ cp one.txt books        # copy into a directory
$ ls books                # list the contents
one.txt

$ cp books more_books     # copy the folder, will fail
cp: books is a directory (not copied).

$ cp -r books more_books  # recursive copy succeeds

$ cp one.txt two.txt books # can copy more than one at a time
                           # will also OVERWRITE the previous
                           # one.txt that was in the folder
$ ls books

$ ls more_books
```

Move files

`mv` to move a file

This is essentially the same thing as renaming. The absolute path that designates a file is simply being changed when something is moved.

```
$ mv one.txt three.txt     # rename one.txt to three.txt
$ mv three.txt books       # relocate three.txt to books folder

$ cd books
$ mv one.txt two.txt three.txt .. # move these files back UP one
directory
$ ls                       # nothing in the 'books' directory
$ cd ..                    # go back
$ ls                       # these files are in the current folder
one.txt two.txt three.txt books more_books
$ ls books                 # is now empty, we moved everything
                           # out of there
```

Let's Try it!

Open up your terminal and let's do a few examples

The PATH variable

How does UNIX determine what program to run?

- Try **`echo $PATH`** to see your search directory
- Also do **`env`** to see all environment variables
- Open a new terminal window; type

```
$ export PATH=""
# try running any program, eg ls or hostname, etc
# will fail because PATH is now empty
```

- You can use the command **`which`** to determine the path to an executable

```
$ which nano                # will tell you where the nano program
                            # is located
```

Some useful commands

- **`wc`** - count the number of lines,words,characters
- **`tail`** - see the last N lines of a file
- **`head`** - see the first N lines of a file
- **`cat`** - print out entire file to screen
- **`sed`** - 'stream editor' - edit data stream on the fly
- **`curl`** - downloading tool for web/ftp data streams
- **`which`** - list path to a program based on the $PATH
- **`pwd`** - print the current working directory
- **`ps`** - processes running on the system
- **`man`** - view manual pages about a command or program
- **`date`** - date and time
- **`time`** - prefix a command/program, report how long it took to run
- **`find`** - find files/folders by name or other property
- **`du`** - reports disk usage (e.g. how big a file or folder is)
- **`awk`** - a simple language for processing files/great for column delimited data

Counting lines with **wc**

Word counting with **wc** is a simple task.

```
$ echo "hello world" | wc
      1       2      12
# process a file
$ wc -l long-file.txt
    9734 long-file.txt
```

Other important commands when running programs

Sometimes something will go wrote, you need to kill/quit the program or you need to suspend the program and come back to it (This is one the powerful things about UNIX).

- Control-C - cancel out of the program (exiting it). Try this:

```
$ cat          # whatever you type will be re-echoed.
# To exit type Control-C
```

- Control-D - sends and End of File (EOF) message

```
$ cat          # whatever you type will be re-echoed.
# To exit can also type Control-D, this also tells cat the
# "file" is over
```

Process control

Control-Z - This send a suspend message to a program that is running. To bring it to

```
$ ping www.ucr.edu  # will keep running till you hit Control-C
64 bytes from 138.23.226.208: icmp_seq=3 ttl=241 time=33.058 ms
# but instead, type Control-Z
[2]+  Stopped                 ping www.ucr.edu
# now put it into the background
$ bg
64 bytes from 138.23.226.208: icmp_seq=22 ttl=241 time=44.175 ms
# it will continue to run now in the background
# bring it back to the foreground with 'fg'
$ fg # still running
64 bytes from 138.23.226.208: icmp_seq=5 ttl=241 time=32.900 ms
# now cancel it with Control-C
```

Process control (cont.)

If you have more than one process in the background, **`fg`** will bring up the most recent one. You can switch to a particular one using the name

```
$ fg ping
# or use the job number
$ fg 2 # since it said [2]+  Stopped ...
# can report the currently background jobs with
$ jobs
[2]+  Running                 ping www.ucr.edu
```

ps for process information

you can see all the processes you are running with **ps**

```
$ ps
  PID TTY           TIME CMD
35995 ttys000    0:00.13 -bash
36064 ttys001    0:00.23 -bash
36525 ttys001    0:00.79 ssh pigeon
36150 ttys002    0:00.17 -bash
36597 ttys003    0:00.08 -bash

$ ps -a # show a longer version
35991 ttys000    0:03.17 login -fp stajich
35995 ttys000    0:00.13 -bash
36063 ttys001    0:00.02 login -fp stajich
36064 ttys001    0:00.23 -bash
41211 ttys005    0:00.00 cat
```

Killing jobs

If you have a runaway process you need to stop you can also use the **`kill`** command

```
$ kill 41211
Terminated: 15
```

Sometimes jobs are hard to kill - suspended or locked up use special option -9 which sends a "Hard Quit" to the program

```
$ kill -9 41211
Killed: 9
```

For those in need of a humourous song about the importance of this command (Note: explicit lyrics!) you can hear CS student/Rapper's take on the command Monzy

Data download with `curl`

`curl` to download data from any URL (http://, ftp://)

```
$ curl http://www.uniprot.org/uniprot/E3Q6S8.fasta
>tr|E3Q6S8|E3Q6S8_COLGM RNAse P Rpr2/Rpp21/SNM1 subunit domain-containing p
OS=Colletotrichum graminicola (strain M1.001 / M2 / FGSC 10212) GN=GLRG_023
MAKPKSESLPNRHAYTRVSYLHQAAAYLATVQSPTSDSTTNSSQPGHAPHAVDHERCLET
NETVARRFVSDIRAVSLKAQIRPSPSLKQMMCKYCDSLLVEGKTCSTTVENASKGGKKPW
ADVMVTKCKTCGNVKRFPVSAPRQKRRPFREQKAVEGQDTTPAVSEMSTGAD

$ curl -OL http://www.uniprot.org/uniprot/E3Q6S8.fasta
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Curre
                                 Dload  Upload   Total   Spent    Left  Speed
100   345  100   345    0     0    724      0 --:--:-- --:--:-- --:--:--

$ curl -o myseqfile.fa  http://www.uniprot.org/uniprot/E3Q6S8.fasta
```

There are some other similar commands that may be installed on other systems

- often use the '-L' in curl to allow URL redirects
- wget - also gets web/FTP on commandline
- ncftpget - for ftp
- lftp - a command line FTP client - also works for http/web

Redirect output

- `>` - write out the output to file (create it if empty, and overwrite if exists)

```
$ curl http://www.uniprot.org/uniprot/E3Q6S8.fasta > E3Q6S8.fa
```

- `>>` - write out output to a file (create it if empty) but append to the end of the file

```
$ echo "my name is " >> what_is_my name
$ echo "Joe" >> what_is_my_name
$ cat what_is_my_name
my name is
Joe
```

- `<` - This is for redirecting INPUT from a file. We'll talk about this more but is how we might pull a set of commands into a program expecting input

```
R --no-save < My_R_commands.R
```

Compression

File compression can save disk space, reduce file transfer time when copying between computers

1. **`gzip`** for GNUzip compression. Single file at a time. (biocluster) pigz is parallelized and can use multiple processors

```
$ du -h data/Nc20H.expr.tab         # report how big the file is
656K    data/Nc20H.expr.tab
$ gzip data/Nc20H.expr.tab          # to compress
$ du -h data/Nc20H.expr.tab.gz      # report size of compressed file
236K    data/Nc20H.expr.tab.gz
$ gunzip data/Nc20H.expr.tab.gz     # to uncompress
```

2. **`bzip2`** for Bzip compression. Better compression than gzip but slower. (biocluster) pbzip2 is parallelized and can use multiple processors

```
$ bzip2 data/Nc20H.expr.tab         # compress with bzip2
$ du -h data/Nc20H.expr.tab.bz2     # report size of bzipped file
204K    data/Nc20H.expr.tab.bz2
$ bunzip2 data/Nc20H.expr.tab.bz2
```

3. **`zcat`**, **`zmore`** and **`bzcat`**, **`bzmore`** to read compressed files on the fly

File / disk usage with **du**

- -h -- human readable (eg bytes, K, G, and T)

```
$ du -h data/Nc20H.expr.tab        # report how big the file is
```

Manual pages with **man**

man prints out a manual page for a program or command

```
$ man ls
NAME
 ls - list directory contents

SYNOPSIS
 ls [OPTION]... [FILE]...

DESCRIPTION
 List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

 -a, --all
        do not ignore entries starting with .

 --author
        with -l, print the author of each file

 -b, --escape
        print C-style escapes for nongraphic characters
```

Setting on the system preferences with .bashrc, .bash_profile, .profile

How the shell determines what variables to set and load is based on a series of system and local configuration files. These are the initialization options that are setup every time a shell is run - when you login or if you type **`bash`** you start a new shell session.

PATH and all other variables set in these files. They can be dependancies on other files, so multiple places can be sourced for these parameters.

Other environment variables are important when we start to install Python (PYTHONPATH) or Perl (PERL5LIB) or other libraries.

If you make changes and want to immediately see the difference without logging out/logging back in. Use

```
$ source ~/.bash_profile
```

Connecting programs with pipes

There are three types of data streams in UNIX

- STDOUT - standard output stream (stream 1)
- STDERR - standard error stream (stream 2)
- STDIN - standard input stream (stream 0)

When run a program and it prints to screen, that is usually STDOUT. This can be captured with > to redirect to a file.

```
$ ls > listing.txt
listing.txt
one.txt
```

STDERR error message capture

If there are error messages they go to the STDERR stream this can be captured

```
$ ls JUNK >> listing.txt
$ cat listing.txt
listing.txt
one.txt
$ ls JUNK 2>> listing.txt
listing.txt
one.txt
ls: JUNK: No such file or directory # this is an error message
$ program 1> log 2> err # STDOUT to one file, STDERR to another
$ program >& all_output # combine STDOUT and STDERR
```

Piping program output

The pipe operator | allows you to instead of redirecting output to a file, redirect it to another program. Specifically the STDIN of the other program. This is very powerful and allows you to chain together different processes

```
$ zcat data/Nc20H.expr.tab.gz | wc -l
# output from zcat is printed to STDOUT and that is redirected
# to the command wc with the -l option which in turn
# expects input on STDIN.

# Output from a program can be fed to a pager like less
$ blastn -help | less
$ fasta36 query db | more
$ fasta36 query db | tee report.out | more
```

`tee` is a program which reads from STDIN and writes this BOTH to a file and to STDOUT. A way to monitor a program but to also detach from reading the messages and still capture it all to a file.

Building pieces into useful, reusable components

Multiple pipes can be used, and building together we can start to construct a series of queries. Will go into this more in detail in the next lecture but here we process a file and capture

```
# shows top ten results from a blast report
$ zcat data/blast.out.gz | head -n 10
# returns the total number of unique items found in column 1
$ zcat data/blast.out.gz | awk '{print $1}' | sort | uniq | wc -l
# take output from blast program, compress it on the fly to a
new file
$ blastn -query query.fa -db db.fa -outfmt 6 | gzip -c > blastresult.gz
```

Wildcards on the commandline

- use the '*' to match anything
- use '?' to match any character 1 time
- indicate specific characters in '[]'

```
$ rm *.jpg   # match all files ending in .jpg
$ ls [A-Za-z]*.jpg # match JPG files with names starting with a
letter
$ ls ???.jpg # match all filenames with three characters
```

Some commandline/filename gotchas

```
$ echo "jeepers" > --oops
$ ls
-rw-r--r--  1 stajich  staff      8 Oct  1 23:06 --oops
$ rm --oops
rm: illegal option -- -
usage: rm [-f | -i] [-dPRrvW] file ...
 unlink file
$ rm ./--oops   # here's how to remove it, give a path

$ touch star
$ echo "oh boy" > st*r
-bash: st*r: ambiguous redirect
$ echo "oh boy" > "st*r"
$ ls st*r
st*r star
$ rm st*r
$ ls st*r # oops this deleted both 'star' and 'st*r' file
$ echo "oh boy" > "st*r"; touch star
$ rm "st*r" # or rm st\*r
```

Practice practice practice

Practice these commands. Many concepts but these are the heart of UNIX access which will give you confidence to start to play with the data files.

- Try out a curl command to download data files (or just a web page directly)
- read the man page for several programs that are new to you - can you find out about other command line options to gzip?

Connecting to different computers with SSH

SSH - Secure Socket Shell, encrypted connection to another computer

```
$ ssh username@biocluster.ucr.edu
The authenticity of host 'biocluster.ucr.edu (138.23.51.30)' can't be estab
ECDSA key fingerprint is SHA256:AAAAC3NzaC1lZDI1NTE5AAAAIAap2P4y+xQArucZdVe
Are you sure you want to continue connecting (yes/no)? yes
password:
```

you will not see your password when typing!

SSH keys

- Keys are long numbers which are used as part of encryption protocol
- They are generated in pairs as 'public' and 'private'
- Can use these keys as ways to login INSTEAD of your password
- Also use these for logging into github from cmdline

See SSH Key Info