shell pipes & loops

previous (notes0908.html) & next (notes0922.html)

Go to the software carpentry introduction (http://swcarpentry.github.io/shell-novice/) and do

- pipes & filters (http://swcarpentry.github.io/shell-novice/04-pipefilter/index.html) and
- loops (http://swcarpentry.github.io/shell-novice/05-loop/index.html)
  - do ask all your questions as you go through this carpentry: ask others on Piazza, or in office hours.
  - ask yourself: why does `uniq` only remove adjacent duplicates? what is the advantage?
  - ask yourself: what happens when we do `wc -l`? why? how do we get out of this?

summary:

- wild cards:
  - `*` matches zero or more characters (anything).
  - `?` matches exactly 1 characters

  the shell expands the wild cards *before* running the command.

- pipes and redirection:
  `>` to redirect the output of one command to a file
  `|` pipes the output of one command to the input of another command: pipeline! very fast: uses streams only.
  `>>` redirects output and appends to a file
  `2>` redirects standard error
  `&>` redirects both output and error (bash shell)

shell loops & scripts

- a variable named `xxx` is later used with `$xxx`
- use all commands seen before, including wild cards
- `echo` to print info during execution of the script
- `;` to separate the pieces
- save the script in a file, say `myscript.sh`, then execute it with `bash myscript.sh`.

```
for xxx in *
do
  echo will analyze this thing next: $xxx
  ls $xxx
done
```

or on one line:
`for xxx in *; do echo will analyze this thing next: $xxx; ls $xxx; done`

examples of ways to loop:
`for i in {1..9}; ...` or `for extension in pdf log png; ...`

how to assign a variable:
`file=out/timetest$i` or `file=out/timetest${i}` or `file="out/timetest${i}"`.

But `xxx` is a very bad variable name: use some other name instead to write your script for a human (yourself in 2 months)

We will skip the section on “Shell Scripts” for now, because we will cover this topic differently later (and we will write safe scripts).

more on redirection

```
ls -d * unknownfile
```

What is this command doing? (hint: do `man ls` to learn about the `-d` option, and ask on Piazza if you don’t figure it out trying out the code below.)
It gives both: some output and some error. Let’s try to capture the output and the error separately.
Navigate to a directory in which you:

- have at least something (at least one file or one subdirectory)
- do *not* have a file named `unknownfile`
- do not have files named `outfile` or `errfile` that cannot be overwritten.

Then try this below:

```
ls -d * unknownfile > outfile
cat outfile
rm outfile
ls -d * unknownfile 2> errfile
cat errfile
rm errfile
ls -d * unknownfile > outfile 2> errfile
cat outfile
cat errfile
rm outfile errfile
ls -d * unknownfile &> outerrfile
cat outerrfile
rm outerrfile
```

What would `2>>` do?

each open file has a “file descriptor”

- standard input: 0, standard output: 1, standard error: 2
- `>` does the same as `1>`

How could `tail -f` (f=follow) be useful to check status of a program that takes very long to finish? example (see here (https://github.com/UWMadison-computingtools-master/lecture-examples/tree/master/mrbayes-example) to reproduce it):

```
cd ~/Documents/private/st679/classroom-repos/lecture-examples/mrbayes-exampl
mb mrBayes-run.nex
```

What if a program generates a whole lot of “standard output” to the screen, which we are not interested in? (interesting output might go to a file)? We can redirect the screen output (STDOUT) to a “fake” disk `/dev/null` (black hole):

```
myprogram > /dev/null
```

processes

let’s repeat this “long” analysis:

```
mb mrBayes-run.nex
```

how to pause/restart/monitor/kill processes:

- **Control-Z** to pause a job (zzz… sleep, or suspend)
- `fg` to resume in the foreground; `bg` to resume, but in the background

- **Control-C** to cancel the job

- `jobs` to see the list of jobs
- `&` added at the end of a command to run it in the *background*, and get the shell back to do other things
- `ps` to see the list of current processes PID = process ID
- `kill` to send a signal to a process: like to kill it (signal 9). `man kill` to see other signals. `kill -9 12167` to kill process # 12167.
- `top` to see all processes, refreshed, shows CPU and memory consumption.

warning: closing the terminal kills the processes started from that terminal: sends a *hangup* signal to its child processes before closing. We will see `tmux` later to avoid this.

- unrelated: **Control-D** to say "done": end of standard input. Explain what happens when you type this:

```
grep "on"
oh my, what is going on?
how to stop this?
^D
```

- unrelated again: **Control-A** to go to beginning of the line, and **Control-E** to the end.

less and man

- `man ls` to get help on `ls`
- other very standard option: `--help`
- the result of `man` is actually passed on to the "viewer" `less`
- try `more` on a long file: shows more and more, one page at a time
- `less` is similar, but much better. Name from "less is more". Power of text streams: can read very long files without having to load the whole thing in memory.

some commands for `less` (there are many more!):

q	quit
enter	show next line
space	show next "page"
d	down next half-page
b	back one page
y	back (yp = up?) one line
g or <	go to first line. 4g or 4G: go to 4th line
G or >	Go to last line
/pattern	search forward
?pattern	search backward
n	next: repeat previous search

- use these commands for `less` to search a manual page and navigate fast between the top, bottom, marked positions, and searched keywords: `man less`
- how to search for anything that does *not* match a pattern?

previous (notes0908.html) & next (notes0922.html)