Bash III

- Bash-Bulak, a village in the Osh Province of Kyrgyzstan
- Bash-Kaindy, a village in the Naryn Province of Kyrgyzstan
- Bash-Karakain, a village in the Naryn Province of Kyrgyzstan
- Bash-Khynysly, a village in the Shamakhi Rayon of Azerbaijan

Dictionary.com, "shell," in *Dictionary.com Unabridged*. Source location: Random House, Inc. http://dictionary.reference.com/browse/shell. Available: http://dictionary.reference.com. Accessed: October 04, 2009.

David Notkin ● Autumn 2009 ● CSE303 Lecture 4

Today

- Processes and basic process management
- More commands: alias, using commands as data, …
- Basic scripting
- Friday:
  - More scripting
  - Social impacts – see copyright FAQ on next slide

From US Copyright office FAQ

- What is copyright?
- What does copyright protect?
- How is a copyright different from a patent or a trademark?
- Can I copyright my website? my domain name?
- How do I protect my idea?
- Does my work have to be published to be protected?
- How do I protect my sighting of Elvis?
- Do I have to send in my work? Do I get it back?
- Does my work have to be published to be protected?
- How much do I have to change in my own work to make a new claim of copyright?
- Which form do I use to register a computer software application I am creating?
- How much of someone else's work can I use without getting permission?
- How much do I have to change in order to claim copyright in someone else's work?
- Could I be sued for using somebody else's work? How about quotes or samples?
- Is it legal to download works from peer-to-peer networks and if not, what is the penalty…
- Can I backup my computer software?
- Can I buy or sell backup copies of computer software?...
- Is it legal to download works from peer-to-peer networks and if not, what is the penalty...

Processes

- A set of Unix commands deal with processes – examples include **`ps`**, **`fg`**, **`bg`**, **`kill`**, …
- What is a process?
- Is it the same as a program? Actually, what is a program?
  - **`hello.c`**, **`hello.s`**, **`a.out`**, …

Rough idea: process

- A process is a running execution of a program
  - Lots of details about processes vary across operating systems – beyond the scope of 303
- When you execute a command, a process is created, the program is instantiated and executed – when the program completes, the process is killed
- If you execute one command twice simultaneously – how would you do this? – then each execution takes place in its own process
  - Each has its own variables, own **`stdin/stdout`**, can take different branches, doesn't know about the other, etc.

Processes: a bit more

- The operating system has its own processes, too
  - Some manage disks, other manage processes, …
  - In Unix, OS processes are owned by **`root`** and each process has a unique ID (PID)
- And other users sharing the same operating system have their own processes
- The OS makes sure that each process gets its chance to execute on the CPU(s) – this is called scheduling

Process commands

command	description
`ps`	list processes being run by a user; each process has a unique integer id (PID)
`top`	show which processes are using CPU/memory; also shows stats about the computer *Keeps executing until killed!*
`kill`	terminate a process by PID
`killall`	terminate processes by name

- use `kill` or `killall` to stop a runaway process (infinite loop)
- similar to `^C` hotkey

Background processes

command	description
`&`	(special character) when placed at the end of a command, runs that command in the background
`^z`	(hotkey) suspends the currently running process
`fg` `bg`	resumes the currently suspended process in either the foreground or background

- You would like some processes to continue while you are doing other things – maybe your editor, maybe a browser, etc.
- You can do this by running some processes "in the background", so the shell doesn't have to wait until those processes finish; ex:
  `$ emacs &`
- If you forget to use `&`, suspend your process with `^z`, then run `bg`

Searching and sorting: redux

command	description
`grep`	search a file for a given string
`sort`	convert an input into a sorted output by lines
`uniq`	strip duplicate lines
`find`	search for files within a given directory
`locate`	search for files on the entire system
`which`	shows the complete path of a command

- `grep` is a very powerful search tool; more over time

Keyboard shortcuts: redux

^***KEY*** means hold `Ctrl` and press ***KEY***

key	description
`Up arrow`	repeat previous commands
`Home/End or ^A/^E`	move to start/end of current line
"	quotes surround multi-word arguments and arguments containing special characters
*	"wildcard" , matches any files; can be used as a prefix, suffix, or partial name
`Tab`	auto-completes a partially typed file/command name
`^C or ^\`	terminates the currently running process
`^D`	end of input; used when a program is reading input from your keyboard and you are finished typing
`^Z`	suspends (pauses) the currently running process
`^S`	don't use this; hides all output until ^G is pressed

File system: redux

directory	description
`/`	root directory that contains all others (drives do not have letters in Unix)
`/bin`	programs
`/dev`	hardware devices
`/etc`	system configuration files ▪ `/etc/passwd` stores user info ▪ `/etc/shadow` stores passwords
`/home`	users' home directories
`/media,/mnt,...`	drives and removable disks that have been "mounted" for use on this computer
`/proc`	currently running processes (programs)
`/tmp, /var`	temporary files
`/usr`	user-installed programs

Aliases

command	description
`alias`	assigns a pseudonym to a command

- Ex: Type `q`, log out of the shell
- Ex: Type `ll`, list all files in long format.
  - `alias q=exit`
  - `alias ll="ls -la"`
- Must enclose the command in quotes if it contains spaces
  - Note: quotes in the shell are *very very very tricky*!
  - Different kinds (`" ' `) with different meanings in different contexts

Another way to combine commands

- **`command1 `command2``**
  - run **`command2`** and pass its output to **`command1`** as a parameter
  - ` is a back-tick, on the ~ key; not an apostrophe
  - best used when **`command2`**'s output is short
- Ex: Create directory "notkin" (when logged in as notkin)
  - **`mkdir `whoami``**
  - What about **`whoami | mkdir`** ?
- Ex: Display all files modified during this calendar year
  - **`ls -l | grep `date +%G``**

xargs: run data as commands

command	description
`xargs`	runs each line of its input as a command

- **`xargs`** allows you to repeatedly run a command over a set of lines
  - often used in conjunction with **`find`** to process each of a set of files
- Ex: Remove all files in my directory tree beginning with **`#`** (probably not a great idea)
  - **`find ~ -name "#*" -print | xargs rm`**

Users

- Unix/Linux is a multi-user operating system
- Every program/process is run by a user
- Every file is owned by a user
- Every user has a unique integer ID number (UID)
- Each user has access permissions for each file, allowing the file to be
  - read or written
  - browsed (if it's a directory)
  - executed (if it's a program)
  - ...

Groups

command	description
`groups`	list the groups to which a user belongs
`chgrp`	change the group associated with a file

- *group*: a collection of users – a user can belong to many groups
- Every file has an associated group
  - a group can be given access to a file or resource
  - the owner of a file can grant permissions to the group
- Every group has a unique integer ID number (GID)

File permissions

dir?	owner	group	others	
-	r w x	r - x	r - x	a.out
d	r w x	r - x	r - x	assign1-scratch
-	r w -	r - -	r - -	a.txt

command	description
`chmod`	change permissions for a file
`umask`	set default permissions for new files

Changing permissions

- letter codes: **`chmod who(+-)what filename`**
  - **`chmod u+rw myfile.txt`** (allow owner to rw)
  - **`chmod +x banner`** (allow everyone to execute)
  - **`chmod ug+rw,o-rwx grades.xls`** (owner/group can read and write; others nothing)
- octal (base-8) codes: **`chmod NNN filename`**
  - three numbers between 0-7, for u, g, o
  - +4 to read, +2 to write, and +1 to execute
  - **`chmod 600 myfile.txt`** (owner can rw)
  - **`chmod 664 grades.dat`** (owner rw; group rw; other r)
  - **`chmod 751 banner`** (owner rwx; group rx; other x)

Basic script syntax

- **`#!interpreter`**
  - The first line of an executable script, causing the file to be run by the given interpreter
  - We will use **`/bin/bash`** as our interpreter
- Ex: A script placed in **`myscript.sh`** that removes some files and then lists all files:

```
#!/bin/bash
rm output*.txt
ls -l
```

Running a shell script

- by making it executable (most common)

  `chmod u+x myscript.sh`

  `./myscript.sh`
- by launching a new shell
  - **`bash myscript.sh`**
- by running it within the current shell
  - **`source myscript.sh`**
  - advantage: any variables defined by the script remain in this shell (more later)

`.bash_profile`

- every time you log in to bash, it runs the file **`~/.bash_profile`**
  - you can put any common startup commands you want into this file
  - useful for setting up aliases and other settings
- Exercise : Make it so that whenever you try to delete or overwrite a file during a move/copy, you will be prompted for confirmation first

Questions?