Operating System Interaction via *bash*

- *bash*, or the "**B**ourne-**A**gain **Sh**ell," is a popular operating system shell that is used by many platforms
- *bash* uses the command line interaction style — generally accepted as appropriate for advanced operating system use (i.e., efficiency over learnability)
- While *bash* is primarily associated with Unix-derived operating systems, there are *bash* implementations for other platforms, such as Cygwin for Windows

Accessing *bash* (or Other Command-Style Shells)

One can access *bash*, or any other operating system shell that uses the command line interaction style, in many ways:

- Operating systems that start up with a GUI shell typically have a *terminal* or *console* application that automatically runs *bash* inside a window
- Text-only operating systems usually run the command-style shell upon user login
- Text-based network connections like *ssh* also run the command-style shell upon [remote] user login

Anatomy of a Command Line Invocation

1. The computer indicates that it is ready for the next command (via a *prompt*)
2. The user (you) type in a command
   - The arrow, backspace, and other keys (you'd be surprised how many) can be used to edit what has been typed so far
   - To completely start over, hold down the *control* or *ctrl* key then hit *c* (i.e., "*control-C*"); this results in a fresh prompt
   - When the command is fully specified, hit the *Enter* or *Return* key
3. The computer performs the command, showing you the result of that command
4. "Rinse and repeat"

For Your Convenience: History and "Autocomplete"

- The *command history* keeps track of the commands that you have typed:
  - Vertical arrow keys navigate through the history
  - Backspace or the horizontal arrow keys stop at the current command and permit editing
  - *control-R* triggers a history search; the *history* command displays the history itself, with each command assigned to some sequential number
- The *tab* key triggers *"autocomplete:"*
  - When there is only one valid possibility, *bash* spells everything out for you
  - With multiple choices, the first *tab* will do nothing (though on some systems you may hear a beep); hitting *tab* a second time will display these possibilities
  - You can repeatedly type a few letters, then hit *tab*, until there is only one choice and that choice appears fully spelled-out on the command line

Command Discovery via "Autocomplete"

- The *exit* command ends your *bash* session
- You can use "autocomplete" to explore other available commands that start with *ex* — first, type just *ex*, then, as stated previously, press *tab* twice; you should now see the other commands that start with *ex*
- Next, type *i* so that your partial command now reads *exi*, then press *tab* twice again; note how the list has either narrowed down, or resulted in spelling *exit* outright, since no other command starts with *exi*

Command Discovery via *man*

Most Unix-derived operating systems come with detailed, text-based documentation, accessible via the *man* command (short for ***man****ual*)

- Invoking *man –k <keyword>* will search for manual "pages" that match the given keyword
- Each manual page has a title (typically the name of the command); invoke *man <command>* to display its "page"
- Use *space* and *b* move forward and backward; *q* quits
- Yes, *man man* works; so does *man bash*…and many more!

Navigating the File System

- A major component of our interaction with an operating system has to do with navigating its *file system*(s) — typically, this is a hierarchical organization of *files* (i.e., individual sequences of bytes) and *folders* or *directories* (i.e., files that "contain" additional files)
- GUI shells usually display the file system in a window, with some indicator for the current folder and its position within the hierarchy
- On a command line, this concept is represented with a string called the *working directory*

- *pwd*, or ***p****rint* ***w****orking* ***d****irectory*, displays the working directory: folders/directories are separated or *delimited* with an OS-defined *separator* symbol
  - **/** ("forward slash") on most Unix derivatives; \ ("backslash") on Windows
  - The original Mac OS used the colon (**:**), and Mac OS X remains "aware" of this (try it)
- *cd* <*directory*> will ***c****hange* ***d****irectory* to *directory*, which can be specified in different ways
  - *Absolute paths*, specified by using the file separator as their first character, start from the very top (or *root*) of the file system
  - *Relative paths* start from the working directory
  - Special shortcuts include **.** for the current directory, **..** for the previous directory, **~** for your home directory, and **~**<*username*> for *username*'s home directory (see below)
  - Yes, "autocomplete" via *tab* works with *cd*
- When *bash* first starts, the working directory is initialized to your user's *home directory* — a folder designated specifically for your files and documents

Listing Files in a Directory

- The *ls* (***ls**t files*) command displays the files in the working directory
- There are many ways to display a file list; current GUI shells typically offer "view options" or something similar
- On the command line, variations such as these are specified using *command line parameters* or *switches*
- By convention, these switches start with one or two dashes (**-**, **--**) followed by a keyword; non-boolean parameters also expect some value (e.g., **--***width=40*)

- Recommendation: Use *ls -F* for your typical *ls* invocation; this appends a file type indicator (***** = executable, **/** = directory, **@** = link, etc.) to the filenames in the list
- The last argument to *ls*, if not a documented switch, is essentially "what to list"
  - Like *cd*, you can specify a directory to list, using the same absolute, relative, or "shortcut" notation described previously
  - In what you should now be noting as a recurring theme, *tab* autocompletion will work for this last "what to list" parameter
  - Beyond *cd*, you can also specify filenames or filename filters, using * as wildcards; other filter variations are also available
- You'd be surprised at what else you can get *ls* to do or display — practice your *man* reading skills by invoking *man ls* and looking at what's available

Distinguishing the Shell from the Programs It Runs

- As seen, command line interaction is conceptually simple *on the surface*: type a command; look at the response; "rinse and repeat"
- Delving deeper, however, note that some commands are *intrinsic* to the shell, while others are simply invocations of programs that are on the system
- Fortunately, there's an easy way to tell: you can use *which <command>* or *locate <command>* — if these command produce the path to a file, then that "command" is actually an executable program

- *An exercise:* Can you tell which of the commands described so far are actually executable programs in the operating system?
- *A follow-up question, once you've gotten a good idea of which commands are implemented internally by bash vs. which ones are actually external, executable programs:* What do you think determines whether or not a command should be implemented *within* a shell as opposed to an external program?
- One of these internal commands is particularly tricky — on some systems, there is, in fact, an executable version of this command; rest assured, though, that *bash* does not run that program, instead implementing its functionality internally…and with good reason