Slide 1

Administrivia

- Reminder: Homework 1 due next Wednesday at 5pm. Hardcopy please.

Slide 2

Minute Essay From Last Lecture

- One person mentioned lots of commands not previously known about. That's a goal of this course: Likely you won't remember details of many of the topics we discuss, but what I hope you do remember is what's available.
- Others mentioned questions I think are of general interest . . .

Slide 3

Uses for Links

- Symbolic links are similar to Windows "shortcuts", so can give you a quick or uniform way to access files. `/usr/local/bin` on our systems contains many such links.
- Symbolic links can also be useful if you install multiple versions of a package; if each is installed in a directory whose name includes the version and you set up a "latest" link that points to the most recent one, users can access a specific version or the most recent one (whatever it is).

Slide 4

Uses for Links

- Uses for hard links are less obvious, but a traditional use is to give multiple names to the same file. On our systems, `c++` and `gcc++` are names for the same file. (Curiously enough, `cc` is a symbolic link to `gcc`, however.)
- Digression/detail: UNIX/Linux filesystems usually keep track of what blocks on disk make up a file by using an "i-node" (index node), which also stores timestamps, permissions, etc. A file's directory entry points to an i-node; for hard links, each link points to the same i-node. `ls -i` lists i-nodes so you can know . . .

Slide 5

Uses for Background Jobs

- One use is to launch a GUI-based program from a terminal window; by running it in the background you avoid tying up the window, (Why would you want to do that? maybe it's easier than finding the program in the menus, or maybe you want to look at any error messages it prints. Or this may provide an easy way to make the program killable — to kill, bring it to foreground with `fg` and interrupt with ctrl-c.)
- Another is to start a long-running program; this is most useful if you redirect output (more about that soon).
- Yet another use, sort of, is if you're logged in remotely and editing something with `vim`; if you need to, say, consult a `man` page, rather than exiting `vim` you can suspend it with ctrl-Z, consult the page, and then get `vim` back with `fg`.

Slide 6

`man` pages, Revisited(?)

- `SYNOPSIS` section can be quite dense, but tries to specify in a standard way all the options/arguments:
- Arguments in brackets are optional.
- Arguments separated by | are "either/or".

Slide 7

Starting a Shell

- From the console, type ctrl-alt-Fn, where n is ...

  Well, it used to be 1 through 6, with the graphical console accessible via ctrl-alt-F7. Now graphical consoles start at ctrl-alt-F1 (can be more than one if more than one user logged in), and the virtual consoles start at ctrl-alt-F2 or later, up through ctrl-alt-F6.
- From a graphical environment, start a "terminal" (a.k.a. terminal window, terminal session, etc.).
- From a Windows system, run `putty`.
- Log in remotely with `ssh`.

Slide 8

A Little About Shells

- Several choices; most commonly used are probably `bash` and `tcsh`. (There are others! This is UNIX. `zsh` and `ksh` are two I've heard of.)

  By default, you get the one in your entry in the password file.
- How to find out what that is? `echo $SHELL`. (This displays the environment variable `SHELL`. More about those later.)
- How to change? `chsh` command on some systems; on others, can only be changed by administrator.

  Or start a different one by typing its name, like any other command.
- Following discussion is about `bash`, but many other shells offer similar functionality.

Slide 9

What Your Shell Does With What You Type — Overview

- Shell provides in-place editing (arrow and other keys), command history, tab completion of filenames, etc. — until you press "return".
- Shell then processes command line — expands wildcards and references to variables, "tokenizes" command into commandname and parameters.
- Shell then either processes command (if a builtin), or locates executable in "search path" (`PATH` environment variable) and forks off a new process.
- Command's return code then available via shell variable.
- (Aside: Wonder what a simple shell program looks like? Look at first programming homework for CSCI 3323 . . .)

Slide 10

What `bash` Does With What You Type — In-Place Editing

- Simple editing — left and right arrows; ctrl-a, ctrl-e, etc. Also ctrl-u for "line kill" and ctrl-k for "delete to end of line".
- Command history — move forward/back with up and down arrows, search with ctrl-r.
- Tab completion — for filenames, command names, etc. (Press tab key twice to show choices, if more than one.)
- Read about `bash` and/or `readline` — `man` and `info` pages for more info. (If you ever write a program that needs command-line functionality, `readline` library is useful.)

Slide 11

What `bash` Does With What You Type — Processing Command Line

- Shell takes completed line and expands filename wildcards, references to variables (more about both in next slides), "tokenizes" command into commandname and parameters, splitting (by default) at whitespace.
- If that's not what you want — e.g., to include a space in a filename, inhibit expansion of filename wildcards, etc. — use escape character (backslash) or quotes. Single quotes inhibit all of this, double quotes all but variable substitution.

Slide 12

What `bash` Does With What You Type — Processing Command Line

- Shell locates command. Two cases:
  - Builtin command — shell executes directly.
  - External command — shell finds an executable by looking in "search path" (`PATH` environment variable) and forks off a new process.

  (Why the distinction? Some things can't reasonably by done in a new ("child") process!)

  (This ignores aliases and shell functions. Next time!)
- Command's return code then available via shell variable `$?`.

  (Why would anyone care? Useful in writing scripts.)

  (Where does the return code come from? whatever is returned by program — e.g., from C program's `main`.)

Slide 13

What `bash` Does With What You Type — Special Keys

- Notice that some keys have meanings other than what Windows users are used to:
- ctrl-c interrupts current process (technically, sends it a particular signal).
- ctrl-d signals "end of file" for input from keyboard. Can use this is programs that read from stdin. In a shell, means "exit", though you can override this.
- ctrl-s may "lock" input and output until ctrl-q is entered. Depends on terminal emulator. Useful to know if it ever happens!
- ctrl-z suspends current process.

Slide 14

Environment Variables

- Associated with a process (e.g., a shell) there can be "environment variables". Useful as another way (in addition to command-line arguments, input from file/keyboard, etc.) of giving process information.
- Some variables of interest — `PATH`, `SHELL`, `HOME`, `USER`.
- To display current value, `printenv FOO` or `echo $FOO`.
- To set value, `FOO=value` (no spaces) in `bash`.
- To make value available to child processes, `export FOO`.

Slide 15

Filename Expansion

- You probably already know about using $\star$ as a wildcard for specifying one or more files. Other options too — "filename expansion" section in full `bash` manual or `info` pages.
- `echo` can be used to check what a particular expression expands to.

Slide 16

Minute Essay

- How is the pace of the class so far? too fast (too much new-to-you info), too slow (too little new-to-you info), . . . ?