Slide 1

Administrivia

- As noted in e-mail, I put a link in TLEARN to the course Web page, so you can find it that way if that's easier to remember.
- For minute essays, put "minute essay" *and the course name or number* in the subject line. (Most class days I teach multiple courses, so this helps me quickly and reliably pick out the minute essays for each one.)

  You can ask me anything course-related, but if your question needs a quick reply, *please* put "urgent" in the subject line.
- Homework 1 on the Web; due next week.

  Note request to tell me about collaboration. I will fill in details probably tomorrow and send mail.
- Slides from Monday expanded to include some of what was mentioned in class.

Slide 2

Minute Essay From Last Lecture

- What are the implications of "everything's a file"?

  C low-level library functions for working with many sources of input and output are "the same" whether applied to regular file or device file or whatever — e.g, `read` to read, `write` to write.
- If everything is in a single hierarchy, how does the O/S know that different parts should be operated on differently (e.g., different physical devices)?

  Single hierarchy can (and usually does) encompass multiple filesystems — e.g., disk filesystem such as EXT4 and `/proc` pseudo-filesystem. Library functions such as `read` make system calls, which in turn call appropriate device driver.
- Not in minute essays but: Mystery of `man -a`? Behavior has changed with this year's build: In previous versions, `q` takes you out of one man page and into the next if more than one. Now brings up a prompt for what to do next. (!)

Slide 3

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 the graphical console is at ctrl-alt-F1 and the virtual consoles are at ctrl-alt-F2 through ctrl-alt-F6.
- From a graphical environment, start a “terminal emulator” (`xterm`, `gterm`, etc.).
- From a Windows system, run `putty`.
- Other ways (log in remotely with `ssh`, …)

Slide 4

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 5

What Your Shell Does With What You Type

- 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 6

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.
- 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 7

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 8

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 alias 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 9

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. (We talked about this a little in class; I’ll review later.)

Slide 10

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 other commands, `export FOO`.

Slide 11

Filename Expansion

- You probably already know about using `*` 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 12

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), . . . ?