Slide 1

Administrivia

- Reading assignments and Homework 1 on Web; homework due next Monday at 5pm.

Slide 2

Shells — Recap/Clarifications/Corrections

- As noted earlier — when you're typing in a text window, you're likely talking to a "shell".
- Several choices; most commonly used are probably `bash` and `tcsh`. By default, you get the one in your entry in the password file. (Change with `chsh` command on some systems.) Can start a different one by typing its name, like any other command.
- Following discussion is about `bash`, but other shells provide similar functionality.

Slide 3

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

- Simple editing — left and right arrows; ctrl-a, ctrl-e, etc.
- 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.

Slide 4

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 5

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

- Shell locates command in "search path" (`PATH` environment variable) and forks off a new process.
- Command's return code then available via shell variable. (Why would anyone care? Useful in writing scripts.)

Slide 6

What `bash` Does With What You Type — Miscellaneous

- Notice that some keys have meanings other than what Windows users are used to — ctrl-C, ctrl-D, ctrl-Z, possibly also ctrl-S, ctrl-Q (depending on environment — e.g., which terminal emulator).

Slide 7

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 8

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 9

Shell Customizations

- At startup, shell reads in various configuration files (see `man` page for details). At least one will be in your home directory (`.bashrc` for `bash`).
- In these files, you can
  - Define/redefine environment variables (e.g., `PATH`, `PS1`). For `bash`, be sure to `export` them. Can define new ones (I find this useful).
  - Define aliases/functions (more on next slide).
- Caution: The default setup on our lab machines is somewhat elaborate. Goal is to have things work right on all environments — Linux (currently FC4), but also Mac OS X. Look at `~defaults/system/SYSTEM.bashrc` for details.

Slide 10

Shell Customizations — Aliases and Functions (`bash`)

- Aliases are simple substitution, no parameters. E.g.

```
alias lt='ls -ltF'
alias google='lynx http://www.google.com'
```

- Functions can have positional parameters. E.g.,

```
function cd-and-show() { cd $1 ; pwd ; ls; }
```

Slide 11

Processes and "Job Control"

- Normally, command you type is a "foreground process". Append `&`, though, and you get a "background process".
- Can make a foreground process a background process, and vice versa (`fg` and `bg` commands; `jobs` command).
- Can even run commands in "batch" mode (`batch` command).

Slide 12

I/O Redirection

- In programming classes I talk about "reading from standard input" (`stdin`) rather than "reading from the keyboard". Why?

  How about `stdout`, `stderr`?
- `stdin` can come from keyboard, file, or inline in shell script. `stdout` and `stderr` can go to terminal or file (overwrite or append), separately or together. (Syntax depends in part on which shell you're using.)
- How is this useful? (e.g., in program development? testing?)
- *OR* — remember quotation from last time?

  "Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface."

Slide 13

Pipes

- "Pipes" provide one-way communication between programs — output of program A becomes input of program B.
- Key component of "the Unix philosophy" — emphasis on providing a toolkit of small programs, mechanisms for combining them.
- "Filters" are programs designed to work this way: `sort`, `head`, `wc`, `sed`, `awk`, and too many others to name.

  Other programs that fit in well — `more`, `less`, `grep`.

Slide 14

Filters

- Some commonly-used filters:

`head`	`tail`
`sort`	`uniq`
`grep`	`wc`
`cut`	`paste`
`tr`	`expand`
`awk`	`sed`

- Use these in combination with, e.g., `ps`, `ls`.
- More next time, and examples.

Slide 15

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), . . . ?