Slide 1

Administrivia

- Reading assignments coming soon.
- Homework 1 will be on the Web soon. Due in about a week.

Slide 2

Reading The Fine Manuals

- One of the most useful things you can learn is how to learn more. Documentation on UNIX systems is not always perfect, and it's not particularly novice-friendly, but usually it's thorough.
- Places to look:
  - `man` pages.
  - `info` pages.
  - Elsewhere on the system. `locate` on Linux may help.
  - The Web, via your favorite search engine.

Slide 3

RTFM — `man` pages

- Reference documentation (as opposed to tutorials).
- Organized into “sections” (user commands, sysadmin commands, library functions, etc.). Can have entry with same name in multiple sections. `-a` option or section number gives access to non-default.
- Of particular interest is the section `SEE ALSO`.
- `man -k` (or `apropos`) to search for command names.
- Try `man man` . . .
- Now you might want to know about `more`, or `less`.

Slide 4

RTFM — `info` pages

- Also reference documentation, sometimes more current / complete than `man` pages. (Why are there are two systems? Probably historical reasons!)
- Organized in a way somewhat similar to hypertext.
- Try `info info` . . .

Slide 5

Other Useful Info-Gathering Commands

- `whereis`.
- `type`.
- `file`.
- `which`.

Slide 6

A Little About Files

- A key underlying concept — "everything's a file" (sequence of bytes). Directories are files. Devices are represented as "special files". Many files are text.
- Things to note:
  - Windows/DOS "extensions" idea doesn't really apply.
  - Also no notion of "drive letters" — all paths form a single hierarchy. Removable media can be "mounted".
  - Security model is simple but fairly flexible — rights (read, write, execute) for owner, group, others.
  - "Links" (hard or soft) allow non-tree directory structure.
- Be familiar with basic commands to manipulate/navigate filesystem.

Slide 7

A Little About Processes

- Another key concept — process as one of a set of “concurrently executing” entities (users, applications, etc.)
- Things to note:
  - Processes can spawn “child” processes. (This happens, e.g., every time the shell runs a command.)
  - Processes can have “environment variables”, inherited by child processes. Examples — `USER`, `PATH`.

Slide 8

A Little About Shells

- Several choices; most commonly used are probably `bash` and `tcsh`. 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

- 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 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.
- 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 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!)
- 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 — 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 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 other commands, `export FOO`.

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 15

Minute Essay

- None really — sign in, unless you have questions?

Slide 16