Slide 1

Administrivia

- None.

Slide 2

Minute Essay From Last Lecture

- What are your goals?

  Many answers, most pretty much in line with my ideas about the course.

  Interesting answer:

  "I don't even know what I don't know."

  One of *my* goals is to give you an idea of what things are possible . . .

Slide 3

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 4

A Little (More) About Shells

- As noted earlier — when you're typing in a text window, you're likely talking to a "shell".
- "Which shell am I using?" can usually find out with `echo $SHELL`.
  How to change? On many Unix systems, `chsh` command. (On some, must be done by sysadmin.)
- In general — to display an environment variable, `echo $ITSNAME`. To set — depends on shell; for `bash`, `ITSNAME=newvalue`. `export` makes available to other programs.

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".

  For `bash`, you probably know about up and down arrows and tab completion for filenames. Tab completion works on commands too, and you can search the command history with ctl-R.

- Shell then processes command line — expands wildcards and references to variables, "tokenizes" command into commandname and parameters.

  Notice — if a parameter needs to include a space, must either "escape" (precede with a space) or enclose in single/double quotes.

- Shell locates command in "search path" (`PATH` environment variable) and forks off a new process.

- Command's return code then available via shell variable.

Slide 6

What Your Shell Does With What You Type, Continued

- 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.

Slide 7

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 FC2), but also Mac OS X. Look at `~defaults/system/SYSTEM.bashrc` for details.

Slide 8

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 9

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 10

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 11

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 12

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`.

Slide 13

Examples

- Find all processes that belong to your username.

  (`ps aux | grep $USER`)

- Find all users who are running processes on the system.

  (`ps aux | awk '{ print $1 }' | sort | uniq`)

- Generate a list of machines that are "up".

  (`ruptime | grep up | awk '{ print $1 }'`)

- Show how much space each subdirectory of your home directory is using, sorted by size.

  (`du -sk $HOME/* | sort -n`)

Slide 14

Minute Essay

- What command could you use to find all aliases defined in your `.bashrc` file and print them out in sorted order?

Slide 15

Minute Essay Answer

- One possible answer:

```
grep alias .bashrc | sort
```