

Slide 1

### Administrivia

- The calendar date brings back memories. History to y'all; not just that for me!
- Reminder: Homework 1 due next Wednesday. Submit by e-mail. PDF or plain-text please.

Slide 2

### Processes, Continued from Last Time

- (Review last few slides briefly.)
- I think a lot of this functionality goes back to the days when for many people using UNIX meant logging into a shared mainframe or "minicomputer" from a text terminal. In that environment, you don't just open a second terminal window, so ways to do multitasking from a single terminal were attractive. Still (I think!) have their uses.

### Starting a Shell

Slide 3

- 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`.

### A Little About Shells

Slide 4

- 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 — 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 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. (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 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 be done in a new (“child”) process!)  
(This ignores aliases and shell functions. More soon!)
- 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 in 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 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 child processes, `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 11

### Another `bash` Feature — Directory Stack

- `bash` maintains a stack of directories. Use commands `pushd`, `popd`, `dirs` to manipulate it.
- Very useful (I think!) if you want to navigate from one deeply-nested subdirectory to another without losing your place.

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), ... ?

Slide 13