## Administrivia

- Note about reading assignments: Yes, they're long! Meant to be skimmed — read carefully only parts that we talked about in class, or that interest you.

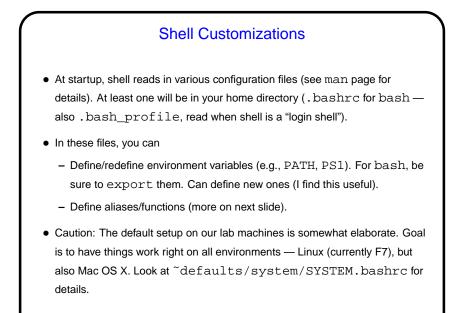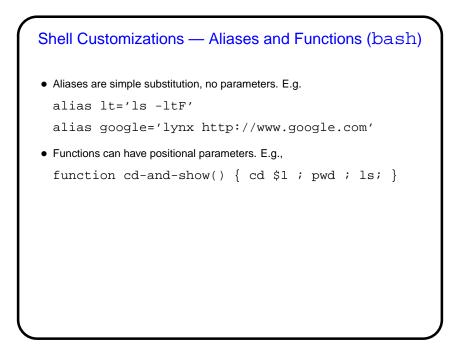- Homework 2 on Web, due in a week.

**Slide 1**

## Shell Built-Ins Versus Commands, Recap
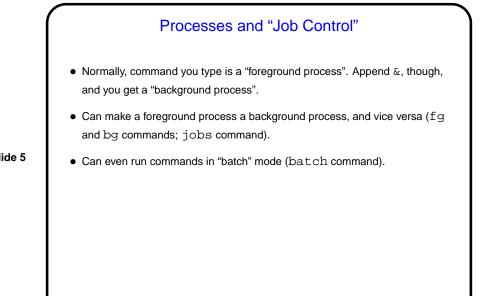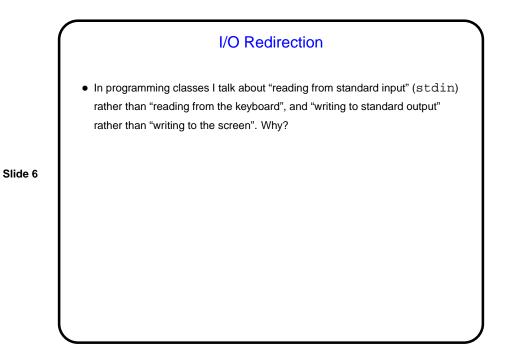
- Last time: First token on each input line is "command", which can be an external command or a shell built-in.

- One key difference — external command executes as a separate ("child") process, so cannot change shell's "execution environment", including environment variables and current directory. (So, $cd$ is a built-in, as is any command that sets environment variables.)

**Slide 2**

## Shell Customizations

**Slide 3**

- 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 — also .bash_profile, read when shell is a "login shell").

- 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 F7), but also Mac OS X. Look at ~defaults/system/SYSTEM.bashrc for details.

## Shell Customizations — Aliases and Functions (bash)

**Slide 4**

- 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; }
```

## 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).

**Slide 5**

- Can even run commands in "batch" mode (batch command).

## I/O Redirection

- In programming classes I talk about "reading from standard input" (stdin) rather than "reading from the keyboard", and "writing to standard output" rather than "writing to the screen". Why?

**Slide 6**

## I/O Redirection, Continued

**Slide 7**

- `stdin` (standard input) can come from keyboard, file, or inline in shell script.

- `stdout` and `stderr` (standard input, output) 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 first class?

  "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."

## Pipes

**Slide 8**

- "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, and there are lots of them (some in next slides and next time). `less` and `more` also useful.

**Slide 9**

## Filters

- `head`, `tail`.

- `sort`, `uniq`.

- `grep` — search for text (or regular expression — more later).

- `wc` — count characters, words, lines.

- `tr` — "translate". Good for converting, e.g., upper-case to lower-case.

- `tee` — duplicates input. Good for capturing output to a file while also displaying it onscreen.

**Slide 10**

## Filters, Continued

- `sed` — "stream editor". Example — convert DOS/Windows-style text file (each line ends with $\verb|\r\n|$) to UNIX-style (each line ends with $\verb|\n|$).

- `awk` — "pattern scanning and processing language" — many interesting possibilities; simplest is just to break up input into whitespace-delimited fields.
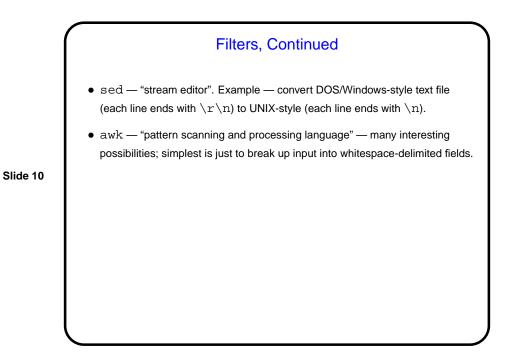
## Examples

- Find all processes that belong to your username:

  ```
  ps aux | grep $USER
  ```

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

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

**Slide 11**

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

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

  (Unfortunately this omits directories starting with a dot.)

## Minute Essay

- What command could you use to count the number of aliases in your `.bashrc` file?

**Slide 12**

# Minute Essay Answer

- One possible answer:

```
grep alias .bashrc | wc -l
```

**Slide 13**