Slide 1

Administrivia

- Reminder: Homework 1 due today (preferably 5pm, hardcopy). (Or should we extend the deadline — Friday or next week?)

Slide 2

Minute Essay From Last Lecture

- See slides for one answer. Various ways to improve so it doesn't include lines that don't match "alias" as a whole word. A perhaps-better solution is to use the `alias` built-in to generate the list!
- Can use `-c` option rather than piping to `wc`.

Slide 3

The Big Picture, Again

- Material in this course can come across as a bunch of parlor tricks — fun in their way, but "so what?"
- The "big picture" view — introduce you to a range of tools that can help you "work smart, not hard". ("Laziness in programmers is a virtue"?)

  The idea — if it's tedious and repetitive and can be done by the computer rather than by the human, make the computer do it! even if that requires the human to think a bit more.

  Once you start thinking along these lines, you may work differently with other tools too (using keyboard shortcuts rather than menus, cutting and pasting rather than retyping, etc.).

Slide 4

Review/Recap

- "UNIX philosophy" emphasizes small programs operating on text and ways to connect them.
- As part of that — I/O redirection, pipes, "filter" programs.

Slide 5

More Useful Commands

- `find`. Very powerful/flexible, though if you don't use it often you probably will have to read the man page to remember syntax.
- Simple examples:
  - Find all files in the current directory modified in the last week.
    ```
    find . -mtime -7
    ```
  - Find all files in your home directory whose name contains `hello`.
    ```
    find $HOME -name "*hello*"
    ```
  - Find all files in the current directory and subdirectories that end in `.bak` and remove them.
    ```
    find . -name "*.bak" -exec rm {} \;
    ```
    (The `-i` flag doesn't work in this context, but if you want to be prompted, replace `-exec` with `-ok`.)

Slide 6

More Useful Commands, Continued

- `diff` — compare files or directories. (A good use — "regression testing" of programs.)
- `pushd`, `popd` (actually shell built-ins) — manipulate shell's stack of directories.
- `cat` (concatenate — one or more inputs to output). Sometimes used when it doesn't need to be, as a substitute for redirecting input ("Useless Use Of Cat (UUOC)").

Slide 7

More Useful Commands, Continued

- `xargs` — "build and execute command lines from standard input".
  - Find all processes for program `java` and kill them:

```
ps aux | grep java | awk '{print $2}' | xargs kill
```

Slide 8

Command Substitution

- Can "inline" output of one command as parameters of another using backquotes. Example:

```
vim `find . -name "*.c"`
```

  or use newer `bash` syntax

```
vim $(find . -name "*.c")
```

- The "inlined" command can even be a pipeline. Example:

```
ls -ld `echo $PATH | sed 's/:/ /g'`
```

- (Notice that these are *backquotes*, not single quotes!)

Slide 9

Shell Input as a Programming Language — the Good

- What `bash` understands is in a sense a programming language, with the shell as its interpreter:
  - Variables (usually untyped).
  - Expressions (arithmetic and logical).
  - Conditionals (if/then/else) and loops.
  - Functions.
- Can be used interactively, or collected into "scripts".
- I will talk about `bash`, but most shells provide similar functionality, just sometimes with different syntax.

Slide 10

Shell Input as a Programming Language — the Bad

- Writing portable scripts is tough. Sticking to the `sh` subset of `bash` helps, as does avoiding GNU-only commands and extensions, but how to do that . . .
- Dealing with spaces (in filenames, e.g.) is a huge pain. Rules for quoting are tricky, and sometimes it seems the only way to get it right is to just try things until something works. (Yuck!)
- Advice: For long and complex scripts, a scripting language such as Perl or Python may be a better choice than a shell script.

Slide 11

Shell Scripts

- A “shell script” is just a sequence of things you could type at the shell prompt, collected in a (text) file.
- Normally, first line of script is `#!` followed by path for shell (`/bin/bash`, e.g.), and the file is marked “executable” (with `chmod`). But you can also execute commands in file `anyfile` via `bash anyfile`.
- With the exception of the first line, lines starting with # are comments.

Slide 12

Shell Variables

- Define/assign variables with, e.g., `myvar="hello"`. (Notice absence of spaces.)
- Reference with, e.g., `$myvar`.
- What’s the difference between these and “environment variables” already mentioned? Shell variables are local to the shell, not passed on to child processes. Distinction is somewhat blurred in Bourne shells. Convention is that environment variable names are all caps.

Slide 13

Shell Functions and Parameters

- Define functions as described previously — `function` followed by name, parentheses, then function definition in curly brackets. Separate/end commands with `;` or newlines.
- Parameters for functions and shell scripts are positional — `$0` for function name, then `$1`, etc. `$*` is a list of all parameters; `$#` is the count of parameters, not including `$0`.
- Call functions or shell scripts by giving name and then parameters, separated by whitespace. (If a parameter should include whitespace, use quoting or escape characters.)

Slide 14

Conditionals and Loops

- Basic syntax for if/then/else:

  `if` command
  `then` list-of-commands
  `else` list-of-commands
  `fi`

  Which branch is taken depends on return code from command after `if` — 0 considered “true”, other values “false”.
- Basic syntax for while loops:

  `while` command
  `do` list-of-commands
  `done`

  Continues until return code from command after `while` is non-zero.

Slide 15

Conditionals and Loops, Continued

- Basic syntax for `for` loops:

  `for` var `in` list-of-values
  `do` list-of-commands
  `done`

- Other constructs include `case` (like C `switch`), `until`.

Slide 16

Useful Commands for Conditions, Loops, Etc.

- Probably the most common for conditions is `test` (commonly abbreviated as square brackets). Many options. Example:

```
if [ -z "$1" ]
then echo Usage:  ‘basename $0‘ someparameter; exit
fi
```

- For lists/loops, `seq`, wildcards, and command substitution are good. Examples:

```
for n in ‘seq -w 0 21‘
do echo Xena$n
done

for f in ‘ls -A $HOME‘
do du -sh $HOME/$f
done
```

Slide 17

Other Features

- Evaluating (numeric) expressions — next time.
- Reading from standard input — next time.

Slide 18

Minute Essay

- What has been interesting, difficult, or otherwise noteworthy about the homework?