Another adv. command: find

- `find dirName... options`..
  - do recursive searching or processing of given directories and all the files & subdirectories they contain, based on options
  - options can be tests that decide whether to consider the file, or commands to perform on that file

Some find tests

- `-name filenamePattern`
  - only match files whose names match `filenamePattern`
- `-type t` (t is f or d or ...)
  - only match files that are plain files (f) or directories (d) or ...
- `-not, -or, \(` ... \)`
  - allow boolean combinations to be specified (and is implicit connector)

Some find actions

- `-print`
  - print out the path name of the current file
  - the default action
- `-exec command arg... \;`
  - run command
    - () in args replaced with matching path name
- `-prune`
  - don't recursively search this directory

Another adv. command: diff

- `diff oldFile newFile`
  - Compare the argument files line-by-line
  - Print out where they differ
    - Some lines may only be in first file (deleted)
    - Some lines may only be in second file (added)
    - Some lines may be different (changed)
  - Clever algorithm & heuristics to find correspondence between parts that are the same in both files

Redirecting output

- So far, commands have appeared to always print their results out to the screen
- Really, output goes to standard output (stdout), which defaults to the screen
  - There's also standard error (stderr), for any error messages, which also defaults to the screen
- It's easy to redirect stdout, e.g. to a file
  - Good if need to to save output for later
  - Good if want to use output as input file for another command (but more on this later)

Redirecting output to a file

- `command arg... > fileName`
  - Redirects command's stdout to `fileName`
- `Overwrites fileName if it exists`
- `Use >> instead to append to file`
- `Leaves stderr alone`
  - `Use > or >> instead to redirect both stdout & stderr to the same file`
    - in csh; bash somewhat different
Programs as stream processors
- Since output redirection is easy, many Unix programs defined to produce their output on stdout, and then let users decide what to do with it
- Likewise, many programs defined to take their input from standard input (stdin), if no explicit file arguments are given
  - stdin defaults to the keyboard
  - can be redirected to a file using <
- Model: stdin → program → stdout

Pipelines
- To exploit this uniform input/output processing, can arrange sequences of programs in pipelines
  - stdin → cmd1 | cmd2 | ... | cmdN → stdout

Pipeline utilities
- Pipelining leads to lots of simple utilities that do one thing well that can be combined to create interesting effects
- Some sources:
  - cat, echo, ls, find, diff, yes,
    - input file redirection
- Some filters & processors:
  - grep, sed, sort, uniq, tee, wc, head, tail
- Some sinks:
  - more, output file redirection, > /dev/null

Defining your own commands
- 3 ways to define your own commands:
  - Write a new program, compile it, and put the executable somewhere in your path
    - Heavyweight
  - Write a script, put it somewhere in your path
    - Lightweight
  - Define an alias, e.g., in your .cshrc
    - Flyweight

Aliases
- alias aliasName command arg...
  - Defines aliasName to be an abbreviation for command arg...
  - Whenever type aliasName aliasArg... at the shell prompt, replaced with command arg... aliasArg...
    - Doesn't work in other contexts, e.g., -exec args

Shell scripts
- Aliases work for one-liners
- For more complex tasks, can write shell scripts
  - A script is a file containing a sequence of regular Unix shell commands
    - can include control structure commands like if, while, foreach, switch
    - can include argument processing operations
  - (.cshrc is just a script run at log-in)
Making a script into a program

- Must start with `#!/bin/csh`
- This says that `/bin/csh` should be used to interpret the rest of the lines
- Can use other interpreter programs, e.g. `/bin/perl`, `/bin/sh`, ...
- Must be marked as executable
- `chmod +x scriptName`
- Must be in a directory in the path

Shell script arguments

- **The `argv` shell variable** is set to the list of arguments to the shell
  - `$argv` expands to the list of arguments
  - `$*` is a synonym for `$argv`
- `$var(n)` refers to the `n`th element of the `var` list
  - `$argv[n]` is the `n`th shell argument
  - `$n` is a synonym for `$argv[n]`
- `$#` refers to the length of the `var` list
- ` $#` is the number of shell arguments
  - `0` is the name of the script being run

Foreach command

- foreach **varName** ( **arg...** )
  - ... body command lines ...
  - end
  - sets `varName` to each `arg` in turn
  - `arg...` is often a pattern
  - evaluates `body command lines` for each setting

Examples

- foreach **f** (*.htm *, .html)
  - echo "moving $f to www/$f"
  - mv $f www
  - end

- foreach **arg** ($*)
  - ... do something to `$arg` ...
  - end

Advanced variable substitution

- Often want to process shell variable bindings (e.g. foreach loop variables)
- Can add qualifiers to extract pieces e.g. of pathnames
- if `$var == a/b/c.d.e` then
  - head: `$var:h == a/b`
  - tail: `$var:t == c.d.e`
  - root: `$var:r == a/b/c.d`
  - extension: `$var:e == e`
- Can repeat modifiers, e.g. `$var:h:h == a`

Example

- foreach **f** (*.htm)
  - set **g** = `$(f:r).html`
  - echo "fixing ext'n of $f to $g"
  - mv $f $g
  - end

  - Note that can use braces after `$` to clearly identify the variable subst. expr.
If command

- if (expr) then
  _ commands _
else if (expr2) then
  _ commands _
else
  _ commands _
endif
  zero or more else-if cases
  optional else case

Test expressions

- String comparisons: ==, !=
- String pattern-matching: =~, !~
- Numeric comparisons & operators, e.g. +, -, <
- Boolean expressions, e.g. &&, ||, !
- Parenthesized subexprs

- if ("$f" == README || "$f" == *.c) ...
- if ($#argv < 2) ...

File test expressions

- Also can test properties of files
  - -e fileName: fileName exists?
  - -f fileName: fileName is a plain file?
  - -d fileName: fileName is a directory?
  - -x fileName: fileName is executable?

- if (-e $f && ! -d $f) ...

See also

- while
- break, continue
- switch, case, default, breaksw
- shift
- exit
- pushd, popd
- time

Shell as a programming language

- How is shell script programming different from regular programming?
  - Types
  - Declarations
  - Procedures
  - Data structures
  - Primitive/built-in operations
  - Libraries
  - Compilation/execution model