Slide 1

Administrivia

- (None?)

Slide 2

Shell Scripts — Review/Recap

- What you type in a shell is a programming language, with the shell as a REPL and "shell scripts" as saved programs.
- First line of a shell script specifies what interpreter to use.
- Can declare variables, functions.
- "Command substitution" allows inlining one command in another.

Slide 3

Conditionals

- 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". (Aha! At last, why C programs return a value from `main()`!)

Slide 4

Conditionals, Continued

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

```
if [ -z "$1" ]
then echo usage $(basename $0) someparameter; exit 1
fi
```

- `case` (like C `switch`) also available.
- `lcname`, `upmachines` examples.

Slide 5

Loops

- Basic syntax for while loops:

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

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

- Basic syntax for `for` loops:

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

- There's also `until`, which executes until the command returns a non-zero (false).

Slide 6

Loops — Examples

- A silly example (runs until interrupted):

```
while true
do
    date ; sleep 1
done
```

- Another somewhat silly example:

```
for n in $(seq 1 5)
do
    ssh dias0$n hostname
done
```

  (Note that this only works well if you have your account set up to allow passwordless login. You can find instructions for setting that up on my home page.)

Slide 7

More Examples

- Rename all `.htm` files in the current directory to `.html` (`-v` isn't really necessary but does show you what's being done):

```
for f in $(ls *.htm)
do
  mv -v $f $(basename $f .htm).html
done
```

(But this fails if names contain spaces. See `rename-files` example.)

Slide 8

More Examples

- Descend into each of several subdirectories and launch a subshell (`exit` to move on):

```
for d in d1 d2
do
  pushd $d ; pwd ; ls ; bash ; popd
done
```

- (`find-broken-links-1` example. But this also does not cope well with names with spaces.)

Slide 9

Arithmetic

- Shell supports simple *integer* arithmetic.

  Most basic/portable way probably `expr`. Example:

  `n=$(expr $n + 1)`.

  In `bash`, can also use double parentheses. Example:

  `n=$((n + 1))`.

  `factorial-1`, `factorial-2` examples.

- But if you're doing significant calculations, you should probably be using some other tool — `awk`, `bc`, `dc`, or a program in a "real" programming language.

Slide 10

`dc` and `bc`

- Both are simple text-mode calculator programs. `dc` uses reverse Polish notation, `bc` the more familiar algebraic notion.
- Both are "arbitrary-precision", which can be useful. Both support non-integer values, but how to set "precision" can be tricky. Details in their `man` pages.
- Used interactively, `bc` may be more useful, since you can use variables within it.
- Both are useful in shell scripting, e.g.,

```
echo "2 + 3" | bc
echo "2^10" | bc
```

- (`powers-of-two` example.)

Slide 11

Reading from Standard Input

- To read from shell's / script's standard input: `read`.
- Example:

```
echo "Do you really want to do this?  (y/n)"
read ans
if [ ".$ans" = ".y" ] ....
```

  (Why the dots? if nothing is read, `$ans` may be empty, with possibly awkward results. May be okay to omit, but a lot of shell scripts use them.)
- Also useful as a way of coping with names with spaces. (`rename-files`, `find-broken-links-2` examples.)

Slide 12

Minute Essay

- Questions?