Slide 1

Administrivia

- Reminder: Homework 2 due today by 5pm.
- Homework 1 solution on Web, linked from “lecture topics and assignments” page.

Slide 2

Shell Scripts — Review

- 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 program to use to execute it (e.g., `/bin/bash`), 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 3

Shell Variables

- Define/assign variables with, e.g., `myvar="hello"`. (Notice absence of spaces.)
- Reference with, e.g., `$myvar`.
- (Same idea as environment variables — in fact there seems to be no clear distinction, except the latter are usually "exported" so they're available to child processes.)

Slide 4

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'`
  ```

Slide 5

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 6

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.

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`.

Useful Commands for Conditions, Loops, Etc.

- Probably the most common for conditions is `test`. 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 $HOME`
do du -sh $HOME/$f
done
```

Arithmetic

- Most basic/portable way probably `expr`. Example: ``n=`expr $n + 1` ``.
- In `bash`, can also use double parentheses. Example: `n=$((n + 1))`.

Slide 9

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" ] ....
```

Slide 10

Slide 11

“Here” Documents

- We talked about redirecting input and output. One more option for input, useful in scripts, is to get it from the script itself — “here” document. Example:

```
#!/bin/sh
mail -s "a subject" bmassing << EOF
hello
I am here
who are you?
is this fun?
EOF
```

Slide 12

A Few More Useful Things

- `getopt` — process command-line options (to help you write scripts that accept options in any order, in the same way most Unix commands do).

Minute Essay

Slide 13

- The command `ping -c 1 Janus00` will test to see if `Janus00` is network-reachable. Write a few lines of `bash` input that would let you “ping” all the `Janus` machines.

Minute Essay Answer

Slide 14

- One possible answer:

```
for n in `seq -w 0 21`
do
    ping -c 1 Janus$n
done
```

- Another answer (contributed by one of you):

```
for n in `ruptime | grep Janus | awk '{print $1}'
do
    ping -c 1 Janus$n
done
```