Slide 1

Administrivia

- Reminder: Homework 2 due today by 5pm.

Slide 2

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 3

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 4

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 5

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 6

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 7

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 8

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 9

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 10

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 11

Other Features

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

Slide 12

Minute Essay

- 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.

Slide 13

Minute Essay Answer

- One possible answer:

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

- Another answer (contributed by a student one year):

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