Shell Scripting

How, why, and what not to do

Cameron Patrick

PLUG Seminar, September 2004

Why write shell scripts?

Number one reason:

Laziness

- Let computers do repetitive tasks rather than humans.

Things the shell is good at

- Automating routine tasks
- Calling other programmes
- Manipulating files and directories
- Simple text manipulation
- But **not** :
  - numerical computation
  - writing network servers
  - anything graphical

How it works

- You put your shell commands in a file.
- Make sure the operating system knows how to execute it.
  - Starts with a “#!” line
  - Chmod it to be executable
- Away you go!

Example

simple.sh:

```
#! /bin/sh
# this is a comment
ls -l
echo “Hello world”
```

```
$ chmod a+x simple.sh
$ ls -l simple.sh
-rwxr-xr-x  1 cameron cameron 56 2004-09-14 13:28 simple.sh*
$ ./simple.sh
```

Shell script concepts

- A line starting with a "#" is a comment, ignored by the shell
- Pretty much anything else is treated as the name of a programme to run, followed by arguments to give it
  - e.g. "ls -l simple.sh" runs the "ls" command with "-l" and "simple.sh" as arguments
- Some things are treated specially
  - variables, quotes, "globbing", ...

Globbing

- You will almost certainly have seen this before.
- "*" matches any character or characters in a file name; "?" matches any single character
  - so "rm *.sh" removes all files ending in .sh
  - "ls sh?rk" matches "shark" and "shirk" but not "shaaaaaark" or "shrk"

Shell Variables

- Assign a value to a variable with "VARIABLE_NAME=value"; e.g. "MY_NAME=cameron"
- Insert the value of a variable with "$VARIABLE_NAME"
  - Or "${VARIABLE_NAME}"
- Some variables are special
- Example: vars.sh

Special Variables

- “$#” - number of command line arguments
- “$@” - all command line arguments (behaves slightly weirdly)
- “$?” - exit status of last command (usually 0 means success)
- “$$” - process ID of this shell

Environment Variables

- Some variables are inherited from the environment when the shell starts
  - The values of these variables are passed on to any programmes the shell starts
  - The “env” command lists them all
- To make sure a variable is placed in the external environment, use “export VARIABLE”

Quoting

- Things in “double” or ‘single’ quotes are treated as one argument.
  - In double quotes, references to variables are expanded
- `back quotes` expand to the output of a command
  - e.g. FOO=“`date`” sets the variable FOO to the current date and time

Conditionals: If/Then

```
if some_command
then
        stuff to be run if true
else
        stuff to be run if false
fi
```

Here “true” means returned an exit status of 0

Conditionals: If/Then

- There is a built-in shell command called "[" which makes shell **if** statements look like normal ones.
  - Allows testing for strings being equal, numbers but less than/more than/etc, also existence of files and directories.
  - See bash "help ["
- Example: ifthen.sh

Conditionals: && and ||

- These are “and” and “or” tests.
- “a && b” runs command b if a returned true (zero)
- “a || b” runs command b if b returned false (non-zero)
- Example: grab-photos.sh

Others

- The shell also supports a “while” loop and a “case” statement much like in C.
- Also a “for” loop to go through all items in a list
- Won’t be covered in any detail tonight

Pipes and redirection

- Redirection: taking the input or output of a programme to or from a file
  - “some_command >output.txt”
  - “some_command <input.txt”
- Pipes: taking the output of one programme, and connecting it to the input of another
  - “some_command | another_command”

Text manipulation: grep, sed and awk

- Grep: print out the lines in a file that match a particular string
- Sed: alter lines of a file according to certain rules
- Awk: a small programming language for processing text files; can do anything that grep or sed can, and more
- Examples: rotate-photos.sh (grep); mangle.sh (sed)

Looking for files: find and xargs

- find will recursively look for files which match some set of criteria, and do things to them
  - e.g. files with a certain name; or directories only; or symbolic links; or files that haven't been modified in the last few days; or which have certain permissions set; or whatever
- xargs reads a list of files on standard input and executes a programme on that list
- Example: xargs.sh

Functions

- Much like other programming languages, the shell supports defining functions, which can be called later as if they were external commands.
- Example: mangle.sh

Miscellaneous

- Running a shell script with “sh -x something.sh” will display every command being executed.
  - useful for debugging
  - equivalent to placing “set -x” at the top of the script
- Running a script with “sh -e” will cause it to abort whenever a command exits with an error
  - equivalent to “set -e” inside the script

Conclusion

- I’ve only really scratched the surface of what can be done by a shell script.
- Reading other people’s scripts can be enlightening.
  - Sometimes as examples of how **not** to do things.

Questions?