Shell Programming 1

By the end of this lecture you should...

- Know what a shell program is
- Be able to run a shell program
- Be able to write simple shell programs

Overview

- What are shell scripts?
- Creating and running shell scripts.
- Using Variables.
- The `.profile` file.
- Input and Output.

You can download all the sample programs from in this lecture from the course home page.

UNIX Command Line

- In UNIX you type commands at the keyboard and the system responds.
- Every operating system has some sort of command interface.
- In UNIX this is a separate program. Shells are different versions of this program.
- Originally there were two shells, `sh` and `csh`.
- `sh` (by Bourne) was best for programming – writing *shell scripts* which executed a sequence of commands.
- `csh` (by Joy) was best for interactive work
- `bash` (Bourne again shell) incorporates combines aspects of both and is a successor of `sh`.

What is a Shell Script

- The shell is a command interpreter – it read commands and then executes them.
- It can work interactively or from a text file.
- A shell program is simply a text file which contains commands you would normally type at the prompt.
- Major differences between shell scripts and other files:
  - First line is usually
    `#!/usr/bin/sh`
    - NB. # is also used for comments.
  - It is normally executable.
- MOST IMPORTANT: A shell script is some “glue” with which you stick together other program.

Why use Shell?

- It is always there!!
- Anything you can type in a shell script you can type at the command line.
- Lots of legacy code is written in Shell.
- Its concepts underpin many other scripting languages
- Lots of UNIX applications make more sense if you know a bit of Shell.

Using a Shell Script

- Create the text file.
- Make it executable (optional): `chmod u+x filename`
- Run it
  - `sh filename`
  - `filename` – only works if file is executable and your `PATH` is set correctly.

A Simple Shell Script

```
#!/usr/bin/sh

ls
echo "done"
```

Each command appears on a separate line.

The Simple Shell Script in Action

```
[lad@bartok examples]$ ls
done.sh
[lad@bartok examples]$ chmod u+x done.sh
[lad@bartok examples]$ done.sh
done.sh
done
[lad@bartok examples]$ sh done.sh
done.sh
done
[lad@bartok examples]$
```


Quick Quiz

Shell commands are the same as UNIX commands so:

1. Write a shell script to print a file called `shell1.pdf`
2. Write a shell script to change the name of a file called `shell1.pdf` to a file called `simple_shell.pdf`
3. Write a shell script to list the files a directory and redirect the output to a file `ls.txt`
4. Write a shell script to make the file, `done.sh` executable.

Answers handed to me at the end of the lecture with your name and CS login on them will be eligable for the course prize.

Answers to the quiz will go up on the website after the lecture.

Some Fundamentals of Programming (Revision)

- Assignment – giving a variable a value
- Input and Output.
- Conditionals – if-then-else and case
- Loops – for, while and do

For small programs things like objects classes and methods get in the way. Shell should only be used for small programs.

Variables and Assignment

- Assigned by `=` – there must be *NO SPACES* round the `=` symbol.
  `TMP_FILE=/tmp/junk`
- Called by `$VarName`
  `lpr $TMP_FILE`
- To set the variable so its available to other shell scripts and at the command line – i.e. make it an Environment Variable use `export`:
  `export TMP_FILE`

Your `.profile` is a shell script

```
# set up personal bin directorys
PATH=$HOME/bin:$PATH:
EDITOR=emacs
LASER=het
export PATH TERM EDITOR LASER
DEFTERM=vt100
ASKTERM=false
```


Some Details

- If you want to use whitespace then use `"`
  `MESSAGE="Program ends OK"`
- If you actually need a $ sign use `\$`
  `echo "A meal in USA costs \$"`
- Similarly you can us `\` instead of quotes
  `MESSAGE=Program\ ends\ OK`

Input and Output

- The first argument to a shell script is called `$1`.
- The second argument to a shell script is called `$2`.
- ... and so on.
- Shell uses `echo` like Java's `println`

Input and Output: An Example

```
#!/usr/bin/sh

echo $1


[lad@bartok bin]$ simple.sh hello
hello
```


Special Input Variables

- `$#` Records the number of arguments passed to the shell, not counting the first command.
  - `simple.sh a b c` sets `$#` to 3.
  - One of its primary uses is to check that enough arguments have been specified
- `$*` All the arguments given to the shell
  - Useful in loops when you want to do something to every input.

More Simple Shell Programs

```
#!/usr/bin/sh

echo $#

[lad@bartok examples]$ count_input.sh
0
[lad@bartok examples]$ sh count_input.sh hello goodbye hello
3
[lad@bartok examples]$ sh count_input.sh hello goodbye hello pink
4
```

Sequences of Commands

- Commands are generally terminated by end-of-line.
- Several commands can be on one line, separated by semi-colons.
- To spread a command over more than one line, end the first line with an escape \.
- Leading white space (e.g. tabs) is ignored.
- Arguments to commands are separated by white space.

Examples of semi-colons and backlashes

```
#!/usr/bin/sh

echo $1
echo $2; echo \
$3


[lad@bartok examples]$ sh echo_three.sh red green blue
red
green
blue
```


Sequences at Work

```
#!/usr/bin/sh

emacs $1
chmod u+x $1

[lad@bartok examples]$ edit_shell.sh new.sh
[lad@bartok examples]$ ls -lt
total 52
-rwxr--r--    1 lad      staff           8 Nov 26 16:10 new.sh
[lad@bartok examples]$
```


Quotes

Single Quotes Treat as a string.

Double Quotes evaluate variables

No Quotes evaluate variables and wild cards

Back Quotes treat as a command

Examples

```
#!/usr/bin/sh

FILE=hello1.java
echo ’ls *.sh $FILE’
echo "ls *.sh $FILE"
echo ls *.sh $FILE
echo ‘ls *.sh $FILE‘

[lad@bartok lad]$ quotes.sh
[lad@bartok examples]$ quotes.sh
ls *.sh $FILE
ls *.sh hello1.java
ls edit_shell.sh list_shell.sh make_executable.sh quotes.sh hello1.java
edit_shell.sh hello1.java list_shell.sh make_executable.sh quotes.sh
```


Summary

- Variables: `$NAME`
- Assignment: `=`
- Input Arguments: `$1`, `$#`, `$*`.
- Output: `echo`.
- Sequences of commands.