BASH script files

Nasser Giacaman
SoftEng 206: Software Engineering Design

BASH scripts

- So, what's a script file? It's essentially just a bunch of commands (that you would othewise type manually in the shell), and they get executed one after the other
- A very simple script file, e.g. myprog.sh

```
#!/bin/bash
echo Hello from `echo $BASH`
```

- `> chmod +x myprog.sh`
- `> sh` (change to another shell)
- `$ ./myprog.sh`

  `Hello from /bin/bash` (the script still executes in the bash shell, because of the first line in the script file)
- `$ bash myprog.sh` (specify like this also possible)

BASH scripts

- But for our purposes, we will only be using /bin/bash, you don't need to care about other shells, etc.
- So, inside your script files always have the following line at the top:
  - `#!/bin/bash`
- And you run your script in one of the following ways:
  - `> bash myscript.sh`
  - `> ./myscript.sh` (requires `chmod +x`)
  - `> myscript.sh` (requires `chmod +x`, & file in your PATH)
- You don't even need the `.sh` extension! It's just for convention so that you can easily distinguish it as a script file (as opposed to a binary executable)
  - `> myscript`

Putting scripts into effect

- Suppose you would like a script to run automatically when you log in, or when you start a new shell
- Execute when you log in:
  - Place it inside ~/.profile
- Execute every time you start a new bash shell:
  - Place it inside ~/.bashrc

- Try it!
  - Call myprog.sh from .bashrc

Your exercise: A backup script

- Make a backup script: it puts all the files in the current folder into a tar file, with the timestamp as part of the filename
- The resulting tar file gets moved into a backup folder in the user's home directory
  - How will you store the backup tar inside the backup folder? Can you reconstruct the folder location in there?
- consider using:
  - `> tar`
  - `> date`
  - `> mkdir`
  - `> mv`
  - Environment variables

Script arguments

- In many cases we would like to pass extra values/options to the script to make it more intelligent (and make decisions customised to the parameters we pass)
- $0
- $1 $2 $3 …
- $#
- $@

BASH reading from standard input

- ```
  echo -n "Please enter your name: "
  read name
  echo "Welcome, $name"
  ```
- ```
  read FRUITS
  FRUITS=($FRUITS)
  echo ${FRUITS[0]}
  echo ${FRUITS[1]}
  ```
- ```
  echo -n "Enter your password"
  read -s passwd
  ```
  ```

If

- Being able to make decisions is powerful
-
```
if [ $1 = "-d" ]
then
        echo "Will delete backups under $BDIR"
        rm -fr $BDIR
        exit
fi
```

-
```
if [ ! $# == 2 ]; then
     echo "Usage: $0 <num1> <num2>"
     exit
fi
```

If

- Comparing strings, surround with quotes
- `if [ “$var” = "hello world" ]`

```
then
        echo "Hello hello"
fi
```

- `if [ -z STRING ]` (true if string length zero)
- `if [ STRING ]` (true if string length non-zero)
- `if [ S1 == S2 ]` (true if strings equal)
- `if [ S1 != S2 ]` (true if strings not equal)
- `If [ ! EXPR ]` (true if EXPR is false)

getopts

- Options are a pretty standard thing, surely there is a better way?

```
> ./myprog.sh -d fish -bchips -e wrongOption
```

-
```
while getopts b:l:d: opt
do
    case $opt in
        b)      breakfast="$OPTARG";;
        l)      lunch="$OPTARG";;
        d)      dinner="$OPTARG";;
    esac
done
```

for

- ```
  for x in one two three four
  do
      echo number $x
  done
  ```
- ```
  for file in $HOME/*
  do
      if [ -d "$file" ]
      then
        echo "$(basename $file) is a folder"
      else
        echo "$(basename $file) is a file"
      fi
  done
  ```

C-style for loop

- ```
  FRUITS=(apple banana pear pineapple)
  ```
- ```
  count = 10
  for (( i = 0; i < $count; i++))
  do
      echo ${FRUITS[$i]}
  done
  ```

More `for` and more file testing

- ```
  for item in "$@"
  do
        echo arg list includes ${item}.
  done
  ```
- ```
  for f in $(ls $HOME)
  do
        echo $f
  done
  ```
- File testing
  - `-d file` (is a directory)
  - `-e file` (exists)
  - `-f file` (is a file, not a directory)
  - `-r -w -x`