40 JAAR 1971 2011

Introduction to bash scripting:
create your own commands & job scripts

Walter Lioen Walter.Lioen@sara.nl
Group Leader Supercomputing
Senior Consultant

What is bash / Why learn bash

- A shell is a piece of software that provides an interface for users of an operating system which provides access to the services of a kernel.
- `bash` is the Bourne again shell

- Where do you encounter the bash
  - your command prompt: `joeuser@login3:~$`
  - batch scripts
  - bash scripts (see below)
- They are all the very same environment.
  Everything you learn about this environment makes your Lisa life more easy.

- bash scripts: “Script” complex / repetitive tasks similar to “bat” files on Windows

Hello World

- Create a file using some editor
  (Alternatively, on a Windows machine, you might use WinSCP)
  ```
  $ pico hello.sh
  ```
- Check: show content of file
  ```
  $ cat hello.sh
  #!/bin/bash
  echo Hello World
  ```
- Make file executable
  ```
  $ chmod +x hello.sh
  ```
- Execute file
  ```
  $ ./hello.sh
  Hello World
  ```

Hello World – Common errors 1/2

- I get an error message when I execute my script:

```
$ ./hello.sh
-bash: ./hello.sh: Permission denied
```

- Check the file permissions

```
$ ls -l hello.sh
-rw------- 1 joeuser joeuser 29 Oct  3 14:48 hello.sh
```

- Problem: missing “execute permission bit”; Fix: set it:

```
$ chmod +x hello.sh
$ ls -l hello.sh
-rwx------ 1 joeuser joeuser 29 Oct  3 14:48 hello.sh
$ ./hello.sh
Hello World
```

Hello World – Common Errors 2/2

- I get an error message when I execute my script:

```
$ hello.sh
-bash: hello.sh: command not found
```

- Fix: use (full) path name:

```
$ ./hello.sh
Hello World
```

- I get an error message when I execute my script:

```
$ ./hello.sh
-bash: ./hello.sh: /bin/bash^M: bad interpreter: No such
file or directory
```

- You most probably copied your file from a Windows machine to Lisa (a UNIX machine where the end of line character(s) differ). Fix by using the `dos2unix` command:

```
$ dos2unix hello.sh
dos2unix: converting file hello.sh to UNIX format ...
$ ./hello.sh
Hello World
```

Command line arguments

- Repetitive command sequences will probably act on different files, so you want to pass a file name as argument
-
```
$ cat arg.sh
#!/bin/bash
echo argument 1: $1
echo argument 2: $2
echo all arguments: $@
$ ./arg.sh foo bar
argument 1: foo
argument 2: bar
all arguments: foo bar
```

Loops

```
for f in file1 file2 file3
do
    echo "processing file $f"
    # do something on $f
done
```

```
for arg in $@
do
    echo "processing arg $arg"
    # do something with $arg
done
```

```
for i in {1..10}
do
    echo "processing element # $i"
    # do something with $i
done
```

Conditional expression

```
#!/bin/bash
if [ $# -lt 1 ]
then
    echo ERROR: The script $0 expected at least one argument
    exit 1
elif [ $# -eq 1 ]
then
    echo The script $0 is invoked with exactly 1 argument
else
    echo The script $0 is invoked with more than 1 argument
fi
```

- Notes:
  - $# clearly denotes the number of arguments
  - $0 denotes the script's filename
  - use spaces around [and]

(Environment) Variables

- **Variables:**

```
$ foo='foo bar'
$ echo $foo
foo bar
```

- **Notes:**
  - don't use spaces around the = sign
  - the value of foo contains a space, so it requires quoting

- **Environment Variables:**
  - variables inherited from/set by parent process
  - a process cannot change its parent environment
  - set in current process using the export command

```
export FOOBAR=foobar
```

Environment Variables

- ```
  $ cat env.sh
  #!/bin/bash
  echo FOOBAR=$FOOBAR
  ```
- ```
  $ unset FOOBAR
  $ ./env.sh
  FOOBAR=
  ```
- ```
  $ FOOBAR=foobar
  $ ./env.sh
  FOOBAR=
  ```
- ```
  $ export FOOBAR
  $ ./env.sh
  FOOBAR=foobar
  ```

- **Notes:**
  - the `unset` command does what its name suggests
  - we can also use: `export FOOBAR=foobar`

“Sourcing” files

- ```
  $ cat foo.sh
  #!/bin/bash
  foo=bar
  ```
- Execute file:
  ```
  $ unset foo
  $ ./foo.sh
  $ echo foo=$foo
  foo=
  ```
- Source file:
  ```
  $ . ./foo.sh
  $ echo foo=$foo
  foo=bar
  ```
- Note: the “`. filename [arguments]`” or `source` command reads and executes commands from `filename` in the current shell environment
  ```

Pattern matching

- `*` Matches any string, including the null string.
- `?` Matches any single character.
- `[...]` Matches any one of the enclosed characters. A pair of characters separated by a hyphen denotes a range. If the first character following the `[` is a `!` or a `^` then any character not enclosed is matched.

- Example file name matching:

```
for f in file[1-3]
do
    echo "processing file $f"
    # do something on $f
done
```

Quoting

- Enclosing characters in single quotes preserves the literal value of each character within the quotes. A single quote may not occur between single quotes, even when preceded by a backslash.
- Enclosing characters in double quotes preserves the literal value of all characters within the quotes, with the exception of `$`, `` ` ``, and `\`.The characters `$` and `` ` `` retain their special meaning within double quotes.

```
$ foo=bar
$ foobar='foo $foo'
$ echo $foobar
foo $foo
```

```
$ foobar="foo $foo"
$ echo $foobar
foo bar
```

More information

- This Introduction is far from complete and was clearly meant as an appetizer. For more information see:

- bash(1) – the bash manual page
  - `man bash`
- The bash command: help
  - `help help`
- SARA tutorials on:
  - http://www.sara.nl/systems/lisa/tutorial
- Advanced Bash-Scripting Guide
  (An in-depth exploration of the art of shell scripting)
  (Current PDF version 891 pages …)
  - http://www.tldp.org/LDP/abs/html/

Thank you for listening!