(BA)SH Scripting Basics

Josh Lange
CPLUG President
Cal Poly, SLO

BASH Scripting

Overview:

- Almost everything (syntactically) is a command
- NO types, declarations, etc
- Quick 'N Dirty
- Most scripts can be inline (they don't have to be in their own file!)

BASH Scripting

Before you start:

Put “#!/bin/bash” at the top of your file

- *nix kernels will look for #!, and they will execute the specified shell
- Result: you can have you script executed from anything equivalent to the exec() call in c.

BASH Scripting

Comments:

```
TEST=testing123 #here is is
#and also here

#note that file4 will be a comment
cat  file1 '#file2'  “#file3”  #file4
.......
```

BASH Scripting

Quoting:
Single quotes=exact output

```
TEST=testing 123

TEST=”testing 123”
echo $TEST
TEST2=”this is $TEST”
echo $TEST2
TEST2='this is $TEST'
echo $TEST2
.......
```

OUTPUT:

```
-bash: 123: command not fo

testing 123

this is testing 123

this is $TEST
```

BASH Scripting

Quoting (contd):

Quoting is important in some cases

- unset variables will not be passed
  - ./a.out $SOME_RANDOM_VAR => argc == 1

- ./a.out “a b c” => argv[1] == a b c
- ./a.out a b c => argv[1] == a, [2] == b....

ABC=”a b c”

- ./a.out $ABC => argv[1] == a, [2] == b....

BASH Scripting

If statements:

```
if <expression>; then
   echo success
else
   echo failed
fi
```

- Any command can be used, 0 an exit code represents “success”
- Think in terms of success/failure not true/false!

BASH Scripting

expressions:

./a.out
vim
startx
[“$i” = 1]

false
true
sleep

.....basically any command

BASH Scripting

Compound Expressions:

./a.out || echo “something failed me!”
vim && exit 0
startx || mail -s “we have a problem”
president@whitehouse.gov

[-f “/tmp/myfile”] || [-f “/tmp/myfile2”] || echo “ahh”

[-f “/tmp/myfile”] || exit 1

false || true

BASH Scripting

If statements:

```
if <expression>; then
   echo success
else
   echo failed
fi
```

- Any command can be used, 0 an exit code represents “success”
- Think in terms of success/failure not true/false!

BASH Scripting

If statements (contd):

```
if [ -f “/tmp/file” ]; then
   echo /tmp/file is a regular file
else
   echo /tmp/file is NOT a regular file
fi
```

- [is a program (/usr/bin/[ ), it takes similar arguments to the program test(1m)

BASH Scripting

If statements (contd):

```
if [ -L “/tmp/file” ]; then
   echo /tmp/file is a symlink

elif [ -f “/tmp/file” ]; then
   echo /tmp/file is a regular file

else
   echo /tmp/file is not a link of a file
fi
```

BASH Scripting

If statements (contd):

```
if ! [ -d “/tmp/directory” ]; then
   echo /tmp/directory is NOT a directory
fi
```

- Expressions can be negated when using if or while

BASH Scripting

While statements:

```
while <expresssion>; do
   echo “doSomething”
done
```

BASH Scripting

While statements:

```
while true; do
   if ! [ -f “/tmp/file” ]; then
      echo /tmp/file does not exist yet!
   fi
   sleep 10
done
```

- optionally you can use “sleep 10” as the expression!

BASH Scripting

While statements:

```
x=0
while [ “$x” -lt 10 ]; do
   echo $x
   let x=x+1
done
```

- FYI: continue and break keywords work too!

BASH Scripting

For statements:

```
for VAR_NAME in <collection>; do
   echo “I got $VAR_NAME”
done
```

- Imagine a collection like command line arguements

BASH Scripting

For statements (contd):

```
for VAR_NAME in a b c; do
   echo “I got $VAR_NAME”
done
```

BASH Scripting

Collections of stuff:

/tmp/* (almost everything in /tmp)
`ls /tmp` (redirecting ls to command line arguments)

also check out: seq 1 10
  (a command that prints numbers 1-10)

BASH Scripting

For statements (contd):

```
for VAR_NAME in `seq 1 10`; do
   echo “I got $VAR_NAME”
done
```

BASH Scripting

Switch/case:

```
case “input string” of
    1)  echo string is 1; break
    ;;

    input*) echo “input what?”; break
    ;;

    *) echo “I dunno”
    ;;
esac
```

BASH Scripting

Functions:

```
#!/bin/bash

someFunc() {
   echo you called somefunc!
   return 0
}

someFunc
```

BASH Scripting

Functions:

- WATCH OUT!

```
#!/bin/bash

someFunc() {
   FILE=myLocalFile
   cat “$FILE”
}

FILE=myGlobalFile
someFunc
cat “$FILE”       #every variable is global!
```

BASH Scripting

Redirection:

stdout of a program to command line: ``(backticks)

- echo `ls -l`

BASH Scripting

Redirection:

piping in:

echo “abc” | ./a.out

./a.out <<END
abc
END

./a.out <<<”abc” #doesn't work in SH!

BASH Scripting

Redirection:

```
someFunc() {
   cat >/tmp/file
}

echo “abc” |someFunc
```

BASH Scripting

User input:

read -p “Please enter some text: ” VAR_NAME

echo “$VAR_NAME”

BASH Scripting

Other useful commands:

```
zenity   #make gui popups in bash!
true     #always returns “success”
false    #always returns “success”
seq      #prints a sequence
cat
read
echo
sed
awk
grep
lockfile   #semaphore locking in your bash script!
..others....
```

Other references

- Comprehensive:
- http://en.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.ht
-
- ~7 page quick reference:
- http://www.digilife.be/quickreferences/QRC/Bash%20
-
- (Some of) the variable manipulation in bash:
- http://www.xaprb.com/blog/2007/03/29/bash-parame
-
- Get these links off the cplug website. Bug mgius/icco in #CPLUG if it doesn't show up there :)