

Shell Script Programming

David Morgan

© David Morgan 2004

Shell scripts

- Files
- Containing sequences of comman-line commands
- Executed collectively in sequence by giving the filename to the shell instead of the individual commands

© David Morgan 2004

Getting script together with shell

- Give script file's name on comand line
 - # myscript
- Give a executable shell on command line with script file's name as parameter
 - # sh myscript
 - # ksh myscript
 - # csh myscript
 - # bash myscript

© David Morgan 2004

Executability

- Script file's name on comand line
 - Current shell executes the file (as child process)
 - File must therefore be executable (use chmod)
- Executable shell on command line with script file's name as parameter
 - Called shell executes the file
 - File need not be executable

© David Morgan 2004

Executing in current shell

- dot on comand line followed by script's name
 - Current shell executes the file (but not as child)
- Commands are executed in current shell
 - e.g., variables created in current shell's space
 - e.g., exit collapses current shell instead of returning to it

© David Morgan 2004

Variables

- Create: DAY=Monday
- Destroy: unset DAY, or terminate script
- List: set

© David Morgan 2004

Special shell variables

- \$0 – script name from command line
- \$1, \$2, etc – command line parameters
- \$* - command line parameters collectively
- \$\$ - process ID (PID) of current process
- \$? – exit status of last command

© David Morgan 2004

Getting user input

- read command
- Variable name list as parameters
- Creates variables, assigns input to each word-by-word
- Final variable in list gets all remaining words

© David Morgan 2004

for – looping through a set of values

```
for variable in values  
do  
    statements  
done
```

© David Morgan 2004

for loop with fixed strings

```
for foo in bar fud 43  
do  
    echo $foo  
done
```

© David Morgan 2004

for loop with wildcard expansion

```
for file in `ls lab*`  
do  
    echo $file  
done
```

© David Morgan 2004

if – conditional execution

```
if condition  
then  
    statements  
fi
```

© David Morgan 2004

conditions

- exit status of condition expression
- test command often used
 - string comparison
 - arithmetic comparison
 - file conditionals

© David Morgan 2004

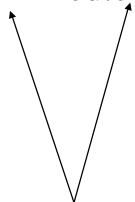
test command

- 2 forms of syntax

– test -f fred.c

– [-f fred.c]

whitespace required!



```
if test -f fred.c
```

```
then
```

```
...
```

```
fi
```

OR

```
if [ -f fred.c ]
```

```
then
```

```
...
```

```
fi
```

© David Morgan 2004

conditions – string comparison

True if:

- `string` string is not an empty string
- `-z string` string is an empty string
- `string1 = string2` strings are same
- `string1 != string2` strings are not same

© David Morgan 2004

conditions – arithmetic comparison

True if:

- `exp1 -eq -exp2` expressions equal
- `exp1 -ne -exp2` expressions not equal
- `exp1 -gt -exp2` exp1 greater than exp2
- `exp1 -lt -exp2` exp1 less than exp2
- `! expression` expression is false

© David Morgan 2004

conditions – file conditionals

True if:

- -e file file exists
- -d file file is a directory
- -f file file is a regular file
- -r file file is a readable
- -w file file is a writeable
- -x file file is a executable

© David Morgan 2004

if

```
echo "Is it morning? Answer yes or no"
read timeofday
if [ "$timeofday" = "yes" ];then
    echo "Good morning"
else
    echo "Good afternoon"
fi
```

© David Morgan 2004

while – conditional repetition

```
while condition do  
    statements  
done
```

© David Morgan 2004

while

```
read trythis  
while [ "$trythis" != "secret" ]  
do  
    echo "Sorry, try again"  
    read trythis  
done
```

© David Morgan 2004

while

```
foo=1
while [ "$foo" -le 20 ]
do
    echo "foo is: " $foo
    foo=$((foo+1))
done
```

© David Morgan 2004

while – ways to increment

```
foo=$((foo+1))

let foo=foo+1

foo=$((foo+1))
```

© David Morgan 2004

Until – conditional repetition

```
until condition  
do  
    statements  
done
```

© David Morgan 2004

until

```
until who | grep "$1" > /dev/null  
do  
    sleep 5  
done  
echo "*** $1 has just logged in ***"
```

© David Morgan 2004

case

```
case variable in  
    pattern) statements;;  
    pattern) statements;;  
    ...  
esac
```

© David Morgan 2004

looping thru a file

```
while read LINE  
do  
    echo $LINE  
done < /home/joe/myfile
```

© David Morgan 2004

looping thru command output

```
who |
(while read LINE
do
    echo $LINE
done)
```

© David Morgan 2004

functions

```
# function definition precedes
myfunction()
{
    <statements>
}

# main program follows
echo "Now about to call the function"
myfunction
```

© David Morgan 2004

here documents

```
# Call as "birthday Lincoln" to print  
the Lincoln record
```

```
grep -i "$1" <<+  
Washington Feb 22  
Lincoln Feb 12  
King Jan 17  
+
```

here document

an embedded "pseudo-file"
because script takes input
from within the script file
itself instead of resorting
to a real, external file

© David Morgan 2004