OPERATING SYSTEM: UNIX/LINUX

Course 4

Osman SALEM
Maître de conférences - HDR
osman.salem@parisdescartes.fr

UNIVERSITÉ PARIS DESCARTES | Université de Paris

1

Conditional Command Execution

- For example:

```
ls file1 && cp file1 /tmp
cp abc xyz && echo The file was copied okay
diff fileA fileB || echo The files are different
ls file2 || exit
```

- The only problem with these constructs is that they are very limited:
  - You can only perform *one* command if the condition is met (however, it *is* possible to group commands)
  - You cannot specify a second command to be run if the condition is *not* met

2

The `if` Statement

- A much more powerful (and readable) shell programming construct is the `if` statement
- It's form is as follows:

```
if command1
then
     command2
     command3
     ...
fi
```

- `command2`, `command3`, etc will only run if `command1` completes with an exit status of `0` (*true*)

3

The `if` Statement (cont.)

- For example:

```
Var1=10; max=5
if [ $var1 -gt $max ]
  then
       echo The files are the same
       rm file2
       exit
  fi
```

```
#!/bin/bash
if  [ -f monfichier.txt ]
then
echo "File exist"
fi
```

```
 if [ $var1 -gt 100 ]
then
    echo That value is too large
fi
```

4

The `else` Clause

- The `if` statement is a powerful language construct, but we still have not seen a way to either:
  - execute commands on the condition that a given command returns a *non*-zero exit status
  - execute commands if a given condition is *not* met
- There is an optional component to the `if` statement, known as the `else` clause, that will facilitate solutions to both of these problems, as follows:

```
if command1
then
    one set of commands
else
    another set of commands
fi
```

5

The `else` Clause (cont.)

- For example:

```
if diff file1 file2 > /dev/null
then
    echo The files are the same
    rm file2
else
    echo The files are different!
    echo Please review the differences:
    diff file1 file2
fi
```

6

The `else` Clause (cont.)

- We now have a way to execute commands if a given command returns a *non*-zero exit status:

```
if ls file1 > /dev/null
then
     :          #   ":" is the "do nothing"
command
else
     echo The file does not exist – exiting...
     exit
fi
```

7

Validate Use of Script

```
# scriptname: ifcond1.sh
if [ $# = 0 ]
then
     echo "Pas de paramètres passés !"
else
     for i in $*
     do
          NBRLIGNES=`grep ^"$i:" /etc/passwd | wc -l `
          if [ $NBRLIGNES = 0 ]
          then
                    echo "$i n'est pas un login"
          else
                    echo "$i est un login"
          fi
     done
fi
```

8

The `elif` Clause

- Often we need to write a conditional code construct in which there are more than two mutually exclusive options
- The `if` statement also offers the `elif` clause (short for *else if*), as follows:

```
if command1
then
    command set 1
elif command2
then
    command set 2
else
    command set 3
fi
```

9

The `elif` Clause (cont.)

- For example (script:elif1.sh):

```
if ls $file > /dev/null 2>&1
then
    echo Sorry, the file already exists
elif who > $file
then
    echo $file now contains the user list
else
    echo Could not create $file
fi
```

- The `elif` clauses can be repeated indefinitely (however, there can only be one `else` clause)

10

elif

```
#!/bin/bash
# script name: elif2.sh
if [ $1 = "Bruno" ]
then
   echo "Salut Bruno !"
elif [ $1 = "Michel" ]
then
   echo "Bien le bonjour Michel"
elif [ $1 = "Jean" ]
then
   echo "Hé Jean, ça va ?"
else
   echo "J'te connais pas, ouste !"
fi
```

11

elif

- $touch file{1,2,3,4,5}

```
#!/bin/bash
# But: transformer les noms de fichiers en majuscule, script name: elif3.sh

    myscriptname=`basename $0` ;

    for i in `ls -A`
    do
        if [ $i = $myscriptname ]
        then
                echo "Sorry, can't rename myself!"
        else

                newname=`echo $i | tr [a-z] [A-Z]`
                mv $i $newname
        fi
    done

    #END
```

12

Conditions

- nb1 *-eq* nb2
- nb1 *-ne* nb2
- nb1 *-lt* nb2
- nb1 *-gt* nb2
- nb1 *-le* nb2
- nb1 *-ge* nb2
- expr1 *-a* expr2: and
- expr1 *-o* expr2: or
- *!* expr:not

13

Conditions

- Comparison
  - **if ["$a" -eq "$b"]**
  - **if ["$a" -ne "$b"]**
  - **if ["$a" -gt "$b"]**
  - **if ["$a" -ge "$b"]**
  - **if ["$a" -lt "$b"]**
  - **if ["$a" -le "$b"]**
- Or with double parentheses
  - **(("$a" < "$b"))**
  - **(("$a" <= "$b"))**
  - **(("$a" > "$b"))**
  - **(("$a" >= "$b"))**

14

Conditions

- Not NULL
  - **if ["$a"]**
- -z zero length string: length is equal to zero
  - **if [-z "$a"]**
- -n greater then 0
  - **if [-n "$a"]**
- String comparison for equality
  - **if ["$a" = "$b"]**
- String comparison (not equal)
  - **if ["$a" != "$b"]**

15

Conditions

- -e file: file exist
- -f file: file exist and is a regular file
- -d file: file exist and is a directory
- -r file: file is readable
- -w file: file has write permission
- -x file: file is executable
- -s file: file has length > 0
- file1 -nt file2: file1 is newer than file2

```
[ -f /etc/passwd ]
[ ! -f /etc/passwd ]
[ -f /etc/passwd -a -f /etc/shadow ]
[ -f /etc/passwd -o -f /etc/shadow ]
```

(Script name: ifoptions1.sh)

```
filename="$HOME"
if [ -e $filename ] ; then echo "$filename exists"; fi
if [ -f "$filename" ] ; then
   echo "$filename is a regular file"
elif [ -d "$filename" ] ; then
   echo "$filename is a directory"
else
   echo "I have no idea what $filename is"
fi
```

16

Conditions

Script name: ifcond3.sh

```
if [ $# -lt 2 ]
then
        echo "Nomber of arguments is smaller than 2"
fi

if [ `whoami` != "root" ]; then
   echo "error. Must be root"
   exit 1
fi
```

17

Conditions

Script name: ifcond4.sh

```
Mystring=abc
if [ $Mystring = abc ]; then
   echo "Mystring = $Mystring"
fi

if [ $Mystring != abc ]; then
   echo "$Mystring is not abc"
fi
```

18

Default value

- ```
  if [ -z "$var1" ]      # test if var1 is unset (t1.sh)
  then
      var1="Some Default Value"
  fi
  ```

- Abbreviation

  ```
  ${var1:="Some Default Value"}
  ```

- Add : before default value

  ```
  : ${var1:="Some Default Value"}
  ```

  Or

  ```
  var2=${var1:="Some Default Value"}
  ```

19

Temporary file $$

- La variable `$$`
  - PID of the script in execution
  - Unique file name
  - Example:

    ```
    tmpfile=/tmp/myscript.$$
    who > $tmpfile
    ...
    rm $tmpfile
    ```

20
    ```

set --

- To initialize arguments
  - `$1`…`$9` are read-only
  - `set --`

    `set -- file1.txt fred`
- `$1` is `file1.txt` & `$2` is `fred`
- `set -- `` `who | grep fred` ``
- Useful for default value of arguments

21

Using `test` (cont.)

- `test` is used as follows:

```
$ var1=10
$ test $var1 = 20
$ echo $?
1
```

- The sole purpose of `test` is to return an exit status appropriate to the condition being tested. This exit status is consistent with the notion of *true* and *false*. In other words, in the above example, `$var1 = 20` is considered to be *false* (`1`)

22

Using `test` (cont.)

- `test` was specifically designed for use with the `if` statement:

```
if test $var1 -gt $max
then
    echo That value is too large
fi
```

23

Using `test` (cont.)

- Notes:
  - In the test `$var1 = 20` example, you *must* have spaces around both sides of the "=" (contrast this with *Assigning Variables*!)
  - If a variable has not been set, or is set to nothing (e.g. `x=`), then checking this variable using `test` will cause a syntax error. This can be remedied by enclosing the variable in double quotes:

```
test "$var1" = 20
```

24

Using `test` (cont.)

- `test` has many useful options:
  - `test` *`value1 = value2`*
    returns *true* (`0`) if the values are equal
  - `test` *`value1 != value2`*
    returns *true* (`0`) if the values are different
  - `test` *`value1 -gt value2`*
    returns *true* (`0`) if the *`value1`* and *`value2`* are both integer (numeric) values and *`value1`* is greater than *`value2`*. Similar options include `-lt` (less than), `-ge` (greater than or equal to), and `-le` (less than or equal to)
  - `test` *`value`*
    returns *true* (`0`) if the value is non-empty
  - `test -z value`
    returns *true* (`0`) if the value is empty (zero-length)

25

Using `test` (cont.)

- `test` also offers many useful options for checking files and directories:
  - `test -f` *`filename`*
    returns *true* (`0`) if the given file exists and is a regular file (i.e. not a directory, device, etc)
  - `test -d` *`filename`*
    returns *true* (`0`) if the given file exists and is a directory
  - `test -s` *`filename`*
    returns *true* (`0`) if the given file exists and has a file-size greater than 0
  - `test -r|w|x` *`filename`*
    returns *true* (`0`) if the given file exists and is readable | writable | executable by the current process

26

Using `test` (cont.)

- The following options can be used in combination with the options mentioned above:
  - `test ! expression`
    returns *true* (`0`) if the expression is considered *false* (`expression` is one of the options mentioned above) (the "`!`" character is read as "not")
  - `test expression1 -a expression2`
    returns *true* (`0`) if the *both* expressions are true (and)
  - `test expression1 -o expression2`
    returns *true* (`0`) if the *either* expression is true (or)

27

Using `test` (cont.)

- Interestingly, another name (an alias) for `test` is "`[`", meaning that our earlier example could have been written:

```
if [ $var1 -gt $max ]
then
    echo That value is too large
fi
```

- Note the (mandatory) use of the closing "`]`"
- This is the way `test` is primarily used

28

The `case` Statement

- Once common use of the `if`/`elif`/`else` statement is to compare the value of a variable to a known set of values. If there are more than two or three values, this can involve considerable code:

```
if [ $var1 = val1 ]
then
    code for case 1
elif [ $var1 = val2 ]
then
    code for case 2
elif [ $var1 = val3 ]
then
    code for case 3
elif [ $var1 = val4 ]
then
    etc
```

29

The `case` Statement (cont.)

- A language construct called the `case` statement was created to make writing this sort of code easier
- The `case` statement looks like this:

```
case $var1 in
    val1)
        code for case 1
        ;;
    val2)
        code for case 2
        ;;
    val3)
        code for case 3
        ;;
esac
```

30

The `case` Statement (cont.)

- The `case` statement has a similar construct to the `else` clause of the `if` statement – simply create a case called *

```
case $var1 in
     val1)
          code for case 1
          ;;
     ...
     *)
          code for any case that
          is not covered above
          ;;
  esac
```

31

The `case` Statement (cont.)

- It is possible to cause the same code to be executed in many different cases, as follows:

```
case $var1 in
     val1|val2|val3)
          code for cases 1-3
          ;;
     ...
```

- It is possible to use wildcard characters to match values to variables, as follows:

```
case $var1 in
     d*)
          code for anything that starts with d
          ;;
     ...
```

32

The `case` Statement (cont.)

- Notes on using `case`:
  - Syntactically speaking, the `case` statement is complicated – there are many language elements to remember and get right:
    - `case`
    - `in`
    - `esac`
    - `)`
    - `;;`
    - `|`
    - wildcards

    Nevertheless, it is usually preferable to using a series of `if…elif…elif…else` clauses
  - Unlike the `if` statement, the `case` statement has nothing to do with *true*/*false* and exit statuses.

33

Example

- `#!/bin/bash` (**case.sh**):

```
echo -n "Enter a number 1 < x < 10: "
read x
case $x in
    1) echo "Value of x is 1.";;
    2) echo "Value of x is 2.";;
    3) echo "Value of x is 3.";;
    4) echo "Value of x is 4.";;
    5) echo "Value of x is 5.";;
    6) echo "Value of x is 6.";;
    7) echo "Value of x is 7.";;
    8) echo "Value of x is 8.";;
    9) echo "Value of x is 9.";;
    0 | 10) echo "wrong number.";;
    *) echo "Unrecognized value.";;
esac
```

34

Switch case

- case value in
  - expression1) commandes ;;
  - expression2) commandes ;;
- esac

```
case $var1 in
 val1|val2|val3)
        code for case 1
        ;;
    ...
    *)
        code for any case that
        is not covered above
        ;;
esac
```

```
(Script name: switch2.sh)
echo -n "Your answer:"
read REPONSE
case $REPONSE in
Y* | y*) REPONSE="OUI" ;;
N* | n*) REPONSE="NON" ;;
*) REPONSE="PEUT-ETRE !" ;;
esac
```

```
case $var1 in
    val1)
        code for case 1
        ;;
    val2)
        code for case 2
        ;;
    val3)
        code for case 3
        ;;
esac
```

35

Switch case

```
#!/bin/bash
#script name: switch3.sh
case $1 in
        "Osman")
                echo "Buenos días M. SALEM !" ;;
        "Ahmed")
                echo "Bien le bonjour M. MEHAOUA" ;;
        "RAMON")
                echo "Hola Senior RAMON, Como está usted? " ;;
        *) echo "M. $1, je ne te connais pas !" ;;
esac
```

36

read

- Plusieurs commandes sur la même ligne:
  - `echo Please enter your Name: ; read name`
- Une commande sur plusieurs lignes
  - `echo This command is split \`
    `over several lines`
- read var1 var2
  - Chaque mot en entrée est enregistré dans une variable
  - Les derniers mots sont stockés dans la dernière variable
- -p pour afficher un message avant la lecture
  - `read -p "Enter a filename: " FILE`

37

Menu

```
#!/bin/bash
# script name: menu1.sh
echo -e "\n COMMAND MENU\n"
echo " a. Current date and time"
echo " b. Users currently logged in"
echo -n "Enter a, ou b:"
read answer
case "$answer" in
    a) date;;
    b) who;;
    *) echo "There is no selection: $answer";;
esac
```

38

The `while` Loop

- The `while` loop *repeats* the execution of a code block in the same way that the `if` statement *conditionally* executes a code block
- The syntax for the while loop is as follows:

```
while command
do
     code block
done
```

- The given code block will be repeatedly executed until the given command returns an exit status that is non-zero

39

Loop `while`

- Example 1 (script name: while1.sh):

```
read answer
while [ "$answer" != "epita" ]
do
     echo Please try again
     read answer
done
```

- Example 2 (script name: while2.sh):

```
who | while read user term time
do
     echo $user has been on $term since $time
done
```

40

while loop and Shift

```
(Script name : while3.sh)
while [ $# -ne 0 ]
do
   echo $1
   if [ $# -ge 2 ]
   then
       shift 2
   else
       shift
   fi
done
```

41

break and continue

- The `break` and `continue` statements can be used to further control the execution of any loop (not just the `while` loop)
- The `break` statement will cause the loop to immediately terminate. Execution will recommence with the next line after the `done`
- The `continue` statement will cause the loop to abandon the current iteration of the loop and begin the next one (the loop condition is retested)

42

`break` and `continue` (cont.)

- For example (cont1.sh):

```
while  [ "$filename" ]
do
   if [ ! -d $filename ]
   then
      echo Must be a directory
      continue
   fi
   if [ `ls $filename | wc -l` -gt 100 ]
   then
      echo stopping - There was a huge directory
      break
   fi
   # process the directory

   read filename
done
```

43

Break et continue

- **Break**
- **Break n**
- **continue**

44

Until

```
#!/bin/bash (scriptname: until1.sh)
selection=
until [ "$selection" = "0" ]; do
        echo ""
        echo "PROGRAM MENU"
        echo "1 - display free disk space"
        echo "2 - display free memory"
        echo ""
        echo "0 - exit program"
        echo ""
        echo -n "Enter selection: "
        read selection
        echo ""
        case $selection in
                1 ) df ;;
                2 ) free ;;
                0 ) exit ;;
                * ) echo "Please enter 1, 2, or 0"
        esac
done
```

45

Until (scriptname: until2.sh)

```
PS3="Please make a selection => " ; export PS3
select COMPONENT in comp1 comp2 comp3 all none
do
        case $COMPONENT in
                comp1|comp2|comp3) CompConf $COMPONENT ;;
                all) CompConf comp1
                     CompConf comp2
                     CompConf comp3
                  ;;
                none) break ;;
                *) echo "ERROR: Invalid selection, $REPLY." ;;
        esac
done
```

The menu presented by the select loop looks like the following:

1) comp1
2) comp2
3) comp3
4) all
5) none

```
Please make a selection ?
```

46

A C-like for loop

- An alternative form of the **for** structure is
  - ```
    for (( EXPR1 ; EXPR2 ; EXPR3 ))
    do
       statements
    done
    ```
- First, the arithmetic expression EXPR1 is evaluated. EXPR2 is then evaluated repeatedly until it evaluates to 0. Each time EXPR2 is evaluates to a non-zero value, statements are executed and EXPR3 is evaluated.
- ```
  $ cat for2.sh
  #!/bin/bash
  echo -n "Enter a number: "; read x
  let sum=0
  for (( i=1 ; $i<=$x ; i=$i+1 )) ; do
    let "sum = $sum + $i"
  done
  echo "the sum of the first $x numbers is: $sum"
  ```

47

Iteration Statements: <list>

- if the `list` part is left off, `var` is set to each parameter passed to the script ($1, $2, $3,…)
  - ```
    $ cat for3.sh ( for1.sh )
    #!/bin/bash
    for x
    do
      echo "The value of variable x is: $x"
      sleep 1
    done
    $ ./for3.sh alba chiara
    The value of variable x is: alba
    The value of variable x is: chiara
    ```

48
    ```