OPERATING SYSTEM: UNIX/LINUX

Course 5

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

UNIVERSITÉ PARIS DESCARTES
Université de Paris

1

Sub value of variable

- use portion of a variable's value via:
  `${name:offset:length}`
  - Name – the name of the variable
  - Offset – beginning position of the value
  - Length – the number of positions of the value

Example:

```
% SSN="123456789"
% password=${SSN:5:4}
% echo $password
% 6789
```

```
${name:offset:length}
echo ${p:2:2}

for ((num=1; num<=5; num++))
do
        echo $num
done
```

2

Manipulating Strings

- Bash supports a surprising number of string manipulation operations. Unfortunately, these tools lack a unified focus.
  - `${#string}` gives the string length
  - `${string:position}` extracts sub-string from `$string` at `$position`
  - `${string:position:length}` Extracts `$length` characters of sub-string from `$string` at `$position`
- Example

```
$ st=0123456789
$ echo ${#st}
10
$ echo ${st:6}
6789
$ echo ${st:6:2}
67
```

3

Special variable uses

- **`${#variable}`**
  length of the string
- **`${variable:-value}`**
  if the variable is unset, “value” is used instead of variable
  if ["${i:-0}" -ge 2] ; then ...
- **`${variable:=value}`**
  if the variable is unset, “value” is used and variable=vlaue
- **`${varname:?message}`**
  if the varname is unset, display the error message
- **`${str/sub/remp}`**
  replace only the first occurrence of sub by remp
- **`${str//sub/remp}`**
  replace all occurrences of sub by remp

4

Advanced operations on strings

- `${string/substring/replacement}`, strips the first match of substring in string with replacement.
  - `pippo=abbcaabccbcabcdbcdabab`
  - `echo ${pippo/ca/11}`

    `# abb11abccbcabcdbcdabab`
  - `echo ${pippo//ca/11}`

    `# abb11abccb11bcdbcdabab  # replaces all matches`

5

Special variable uses

- ${#string} : String Length
- **`${str#sub}`**: deletes the shortest match of $substring from front of $string
- **`${str##sub}`**: deletes the longest match of $substring from front of $string
- **`${str%sub}`**: deletes the shortest match of $substring from back of $string
- **`${str%%sub}`**: deletes the longest match of $substring from back of $string

```
$ cat shortest.sh
#! /bin/bash
filename="bash.string.txt"
echo ${filename#*.}
echo ${filename%.*}
echo ${filename##*.}
echo ${filename%%.*}
$ ./shortest.sh
After deletion of shortest match from front: string.txt
After deletion of shortest match from back: bash.string
After deletion of longest match from front: txt
After deletion of longest match from back: bash
```

6

Special variable uses

- `${PARAMETER^}`
- `${PARAMETER^^}`
- `${PARAMETER,}`
- `${PARAMETER,,}`
- `${PARAMETER~}`
- `${PARAMETER~~}`

7

Numerical Calculations

- As mentioned before, the Bourne shell has no notion of a number (only strings), and as such is incapable of doing numerical calculations
- However, there is a UNIX program called `expr` which was designed specifically for this purpose
- It works like this:

```
$ expr 3 \* 4
12
$ a=15
$ b=3
$ c=`expr $a / $b`
$ echo $c
5
```

8

Numerical Calculations

- Expr +, -, *, /, %

```
$ expr 3 \* 4
12
$ a=15
$ b=3
$ c=`expr $a / $b`
$ echo $c
5
```

```
read count
  i = 1
  while [ $i -le $count ]
  do
      echo This is loop $i of $count
      i=`expr $i + 1`
  done
```

9

Numeric variables

- Syntax:

```
let varname=value
```

- can be used for simple arithmetic:

```
let count=1
let count=$count+20
let count=count+2*4
let count+=1
```

11

Numeric operations

- Syntaxe:
  - let varname=value
  - $((expression))
  - $[expression]
  - **expr**
- Exemple:

```
let count=1
let count=$count+20
let count+=1
SUM=$[$SUM + $SCORE]
NUM=$[$NUM + 1]
AVERAGE=$[$SUM / $NUM]
 asquared=$(($adjacent ** 2))
 hsquared=$(($osquered + $asquared))
```

```
#!/bin/bash
var=12345
let var=$var+1 # let is important
echo $var
v=12345
v=$v+1 # result "12345+1"
echo $[2**3]
echo $((123 + $var))
```

12

Random

- $RANDOM:
  - 0<$RANDOM< 65536

```
#!/bin/bash
# script name: random.sh
i=0
while [ $i -lt 10 ]
do
    x=$RANDOM
    echo $x
    i=$((i+1))
 done
```

13

The `for` Loop

- There is a second type of loop available in the Bourne shell, called the `for` loop
- It causes a variable to be set to a given sequence of values, and then executes a code block once for each value, as follows:

```
for var1 in Bread Meat Dairy Vegetables Fruit
do
   echo One of the main food groups is $var1
done
```

- Note that is different from most programming languages, where the `for` loop is used to execute code a precise number of times (based on a minimum and maximum)

14

Boucle

```
for variable in liste_de_valeurs
do
    commandes
done
```

```
$ for file in *.txt;
> do
>   mv -v $file $file.old;
> done
barbie.txt -> barbie.txt.old
food.txt -> food.txt.old
quirks.txt -> quirks.txt.old
```

```
for NUMBER in 0 1 2 3 4 5 6 7 8 9
do
   echo The number is $NUMBER
done
```

```
for FILE in `/bin/ls`; do echo $FILE; done
```

```
for NAME in jean ali julie
do
        MESSAGE='Bonjour !'
        echo $MESSAGE
done
```

```
for num in $(seq 1 10)
for file in *.txt
```

```
for i in {1..10}
do
        ./something
done
```

15

Examples

- ```
  for f in hw1.*; do
    mv $f ${f//hw1/hw2};
  done
  ```

16

Functions

- Syntax

```
Function_name()
{
   commandes
}
```

```
function DisplayHello ( ) {
echo "hello"
}
DisplayHello
```

- To call a function
  - function_name [arguments]
  - $1 à $9 et $#
  - `$* & $@` : whole arguments
  - "$*" = "$1 $2 $3 ... $n" = all parameters in single word
  - "$@"="$1" "$2" "$3" ... "$n" = all parameters in array of words
  - $0 contains the name of the script
- Local variables
  - **local** inside the function

17
```

Defining and Calling a Function

- A *function* is a named code block that may be run from any point in the program, simply by invoking its name
- Functions must be defined before they can be used
- A function is defined as follows:

```
function_name()
{
    code block
    ...
}
```

- Once a function has been defined in this way, it may be used at any time as follows:

```
function_name
```

18

Function

```
MyFunction()
{
local maVariable="coucou"
echo $maVariable
}
maVariable="Bonjour"
echo $maVariable
MyFunction
echo $maVariable
```

19

Function : Usage

- Exemple: `$grep`

```
   Usage: grep [OPTION]... PATTERN [FILE]...
```

-
```
if [ $# -lt 2 ]
then
   echo Usage: myscript username filename ...
   exit 2
fi

 echo Usage: $0 username filename ...
 echo Usage: `basename $0` username filename …
```

23

Array variables

- Syntaxe:

`varname=(list of words)`

- Acces through index:

`${varname[index]}`

`${varname[0]}`	1ier element
`${varname[*]}`	whole elements

25

Arrays

- To extract all the elements, use an asterisk as:
  `echo ${arraynames[*]}`
- To see how many elements are in the array:
  `echo ${#arraynames[*]}`
- We can combine arrays with loops using a **for** loop:
  - ```
    for x in ${arrayname[*]}
    do
     echo $x
    done
    ```

26

ARRAY

Examples:

```
$ ml=(Farida Ahmed Osman Lynda Alice)
$ echo ${ml[*]}
Farid Ahmed Osman Lynda Alice
$ echo ${ml[2]}
Osman
$ echo ${#ml}
6
echo ${!array[*]}
0 1 2 3 4
```

27
```

Array

- ARRAY[INDEX]=value
- ARRAY=(value1 value2 ... valueN)
- read –a arrayname
- ARRAY=*(one two three)*
- echo ${ARRAY[*]} or
  - echo ${ARRAY[@]}
- echo ${ARRAY[2]}
- ARRAY[3]=*four*
- echo ${ARRAY[*]}

echo ${ARRAY[*]}

unset ARRAY[1]

unset ARRAY

Nb of elements:

echo ${#var}
ARRAY=*(one two three)*
echo ${#ARRAY[*]}

28

ARRAY

```
echo "Enter your favorite fruits: "
read -a fruits
echo You entered ${#fruits[@]} fruits
for f in "${fruits[@]}"
do
        echo "$f"
done
for i in ${!fruits[@]} # all indexes
do
        echo fruits[$i]=${fruits[i]}
done
$ array=( "${fruits[@]}" "grapes" )      # add to end
$ copy="${fruits[@]}"                    # copy an array
$ unset fruits[1]                        # delete one element
$ unset fruits                           # delete array
```

29

Array

```
read -p "Enter total number of entries: " total
for ((n=0; n < $total; n++))
do
      read -p "Enter value: " entries[$n]
done

for ((n=0; n < $total; n++))
do
      echo "Value $n is  ${entries[n]}"
done
```

30

Array

```
declare -a Tool
Tool[0]="Wrench"
Tool[1]="Hammer"
Tool[2]="Saw"
for num in 0 1 2
do
      echo ${Tool[num]}
done
```

```
#!/bin/bash

farm_hosts=(web03 web04 web05 web06 web07)
for i in ${farm_hosts[@]};
do
            echo $i
done
exit 0
```

```
#!/bin/bash
arr=(aa bb cc dd)
n=${#arr[@]}
echo $n
arr=( "${arr[@]}" "newElem" )
arr=( "newElem" "${arr[@]}" )
unset arr[${#arr[@]}-1]
```

```
#!/bin/bash
arr=(aa bb cc dd ee ff gg)
echo ${arr[*]} # all array
echo ${arr[@]:0} # aa bb cc dd ee ff gg
echo ${arr[@]:1} # bb cc dd ee ff gg
echo ${arr[@]:2:3} # cc dd ee
for i in ${arr[*]}
do
            echo $i
done
```

31