

Computer Systems and Architecture

UNIX Scripting

Ruben Van den Bossche

Original slides by Bart Sas

University of Antwerp



Outline

Basics

Conditionals

Loops

Advanced

Exercises

Outline

Basics

Conditionals

Loops

Advanced

Exercises

Shell scripts

- ▶ Grouping commands into a single file
 - Reusability
- ▶ Possible to use programming constructs
 - ▶ Variables
 - ▶ Conditionals
 - ▶ Loops
 - ▶ ...
- ▶ No compilation required

Creating a shell script

1. Save the script as a (.sh) file
2. Add the line `#!/bin/bash` (or `#!/usr/local/bin/bash` on radix) to the beginning of the script
 - ▶ `#!` indicates that the file is a script
 - ▶ `/bin/bash` is the shell that is used to execute the script
 - ▶ When the script is executed, the program after the `#!` is executed and the name of the script is passed to it
 - ▶ Since the line starts with a `#` it is ignored by the shell
3. Make the script executable using `chmod +x`
4. Execute the script by calling it
 - ▶ Put `./` in front of the name in order to avoid confusion with commands

Comments

- ▶ Comments are placed behind a # and last until the end of the line
- ▶ There are no multiline comments
- ▶ The #! line is a comment

Variables

- ▶ Setting variables
 - ▶ `VARIABLE=value`
 - ▶ No spaces before and after the '='
- ▶ Using variables
 - ▶ Place a '\$' before the name
 - ▶ If the variable name is followed by text → place the name between braces
 - ▶ E.g.: `echo "Today is the ${DAY}th day of the week"`
- ▶ Waiting for keyboard input
 - ▶ `read VARIABLE`
- ▶ Exporting variables
 - ▶ To make them accessible from other programs
 - ▶ Place 'export' before the name of the variable
 - ▶ E.g.: `export PATH='/bin:/usr/bin'`

Special variables

- `$@` Expands to the list of positional parameters, separated by commas
- `$#` The number of positional parameters
- `$0` The name of the script
- `$1, ..., $9` The nine first positional parameters
- `$?` The exit status of the last executed command
- `$!` The PID of the last process that was started in the script
- `$RANDOM` A positive random integer

Example

- ▶ pico script.sh

```
#!/usr/local/bin/bash
name='whoami'
echo Hello $name !
```
- ▶ Execute:

```
chmod +x script.sh
./script.sh
```

Basics

Conditionals

Loops

Advanced

Exercises

Conditions

- ▶ Between [...]
- ▶ Spaces before and after []
- ▶ Examples
 - ▶ [-d dir] returns true if dir is a directory
 - ▶ [\$var -eq 2] returns true if \$var equals 2
 - ▶ [\$var -eq 1] || [\$var -eq 2] returns true if \$var equals 1 or 2

Conditions - Files

- e File exists
- d Is a directory
- f Is a regular file
- r Is readable
- w Is writeable

Conditions - Strings

`-n` Length of string is nonzero

`-z` Length of string is zero

`s1 = s2` s1 and s2 are identical

`s1 != s2` s1 and s2 are not identical

Conditions - Numbers

`i1 -eq i2` `i1` and `i2` variables are equal

`i1 -ne i2` `i1` and `i2` variables are not equal

`i1 -gt i2` `i1` is greater than `i2`

`i1 -ge i2` `i1` is greater than or equal to `i2`

`i1 -lt i2` `i1` is less than `i2`

`i1 -le i2` `i1` is less than or equal to `i2`

Conditions - And, or, not

! negation (NOT) operator

&& AND operator

|| OR operator

If statements

```
if [ $# -ne 1 ]
then
    echo Please specify your name
elif id $1 > /dev/null
then
    echo Hello $1
else
    echo I don\'t know you
fi
```


If statements

- ▶ Zero or more elif clauses are possible
- ▶ The else clause is optional
- ▶ The if body is executed if the exit status of the condition is 0

Case statements

```
case $NUMBER
in
  11|12|13)
    echo ${NUMBER}th
    ;;
  *1)
    echo ${NUMBER}st
    ;;
  *2)
    echo ${NUMBER}nd
    ;;
  *3)
    echo ${NUMBER}rd
    ;;
  *)
    echo ${NUMBER}th
    ;;
esac
```

Case statements

- ▶ Executes code based on which pattern matches a word
- ▶ Multiple cases can be specified per block by separating them using ' | '
- ▶ Each block has to be terminated by a ' ; ; '
- ▶ Use '*' to match 'the rest'
- ▶ If multiple cases match, the first one is executed

Outline

Basics

Conditionals

Loops

Advanced

Exercises

For loops

```
for FILE in `ls /bin`  
do  
    echo "Creating link to $FILE..."  
    ln -s /bin/$FILE  
done
```

For loops

- ▶ The list can be
 - ▶ A literal list: `a b c`
 - ▶ A glob pattern: `*.jpeg`
 - ▶ The output of a command: `'ls -a'`
- ▶ The body is executed for each element in the list
- ▶ The Loop variable is set to the value of the current word

While and until loops

```
while [ -f file.txt ]  
do  
    echo file.txt still exists... Please remove it  
    sleep 5  
done
```

While and until loops

- ▶ The condition is evaluated on each iteration
- ▶ While loops are executed as long as the exit status of the condition is zero
- ▶ Until loops are executed as long as the exit status of the condition is not zero

Break and continue

```
for I in `seq 10`  
do  
  if [ $I -eq 3 ]  
  then  
    echo Skipping 3...  
    continue  
  fi  
  
  if [ $I -eq 7 ]  
  then  
    echo Stopping at 7...  
    break  
  fi  
  
  echo The square of $I is  $$(($I*$I))$   
done
```

Break and continue

- ▶ `break` causes a loop to be exited immediately
- ▶ `continue` causes a loop to continue with the next iteration
- ▶ An integer parameter can be specified to continue or break from the n th enclosing loop
 - ▶ `'break 2'` will break from the second enclosing loop
 - ▶ `'continue 1'` is the same as `'continue'`

Outline

Basics

Conditionals

Loops

Advanced

Exercises

Arithmetic

- ▶ Arithmetic can be performed between `((and))`
- ▶ Only operations on integers are possible
- ▶ The exit status is 0 when the result of the expression is not zero and 1 if the result of the expression is zero
- ▶ An expression between `$((and))` expands to the result of the expression.
- ▶ For more advanced calculations `bc` can be used.

```
A=$RANDOM
B=$RANDOM
C=$A
D=$B

while ((D != 0))
do
    TEMP=$D
    D=$((C % D))
    C=$TEMP
done

echo "The GCD of $A and $B is $C"
```

Functions

- ▶ Functions behave the same as commands
- ▶ The exit status of the function is the exit status of the last executed process
- ▶ Parameters are placed in variables \$1, ..., \$9
- ▶ Use 'return' to exit from the function early
- ▶ Use the 'local' keyword to make local variables

Further reading

- ▶ The Bash Manual
www.gnu.org/software/bash/manual/bashref.html
- ▶ Advanced Bash-Scripting Guide
tldp.org/LDP/abs/html/

Outline

Basics

Conditionals

Loops

Advanced

Exercises

- ▶ <http://msdl.cs.mcgill.ca/people/hv/teaching/ComputerSystemsArchitecture/#csw4>