Shell bash

Enrique Arias
Universidad de Castilla–La Mancha

Contents

Main objectives

- Introduce the shell
- Know redirections and pipes
- Understand Shell and environment variables
- Know ` operator
- Know internal commands of the shell

Introduction to Shell

- What is the Shell?
  - Interface berween user and operating system
  - Translate order given by user and execute his/her programs
- $ → Shell indicator
- Writing bash a new process is created → Son process
- Writing exit back to the previous Shell
- ps commnad → active processes

Introduction to Shell

- Exercise 1
  - Login to the system (remember practice 1)
  - Write bash and then ps command. How many Shell do you have ?
  - Write exit and then ps command. How many Shell do you have now?
  - What would happen if you write exit?

Introduction to Shell

- Predefined variables
  - HOME: contains the absolute pathname of your home directory
  - PATH: This is a list of directories in which the shell looks to find commands
  - PS1: contains the primary prompt ($ or #)
  - PS2: contains the secondary prompt (> and /Enter)
  - IFS: This variable determines how bash recognizes fields, or word boundaries, when it interprets character strings
  - SHELL: contains the absolute pathname of your login shell
  - TERM: contains the name of your terminal type
  - env VS set
  - echo $variable_name

Introduction to Shell

- Predefined variables

$?	Specifies the exit value of the last command executed. 0 indicates successful completion.
$$	Identifies the process number of the current process
$!	Specifies the process number of the last process run in the background using the & terminator.
$#	Specifies the number of positional parameters
$@	Expands the positional parameters, beginning with **$1**
$0	Command name
$i	Positional paramenter i>0

Introduction to Shell

- Defining variables
  - variable_name= value
  - variable_name=   /* empty variable */
  - variable="$dir" then variable gets the value stored in variable dir
  - variable='$dir' then variable gets value $dir

Introduction to Shell

- Exercise 2
  - Create variable test=MYTEST; echo $test
  - Create variable my=MY; test_1=$myTEST ; echo $test_1
  - Create variable test_2="$myTEST"; echo $test_2
  - Create variable test_3=´$myTEST'; echo $test_3
  - Compare the results

Introduction to Shell

- Variables and substitution formats
  - new_variable=${variable} → new_variable takes the value of variable.
  - new_variable=${variable:-word}

```
if variable is empty
   then
      new_variable=word
   else
      new_variable=variable
```

Introduction to Shell

- Variables and substitution formats
  - new_variable=${variable:=word}

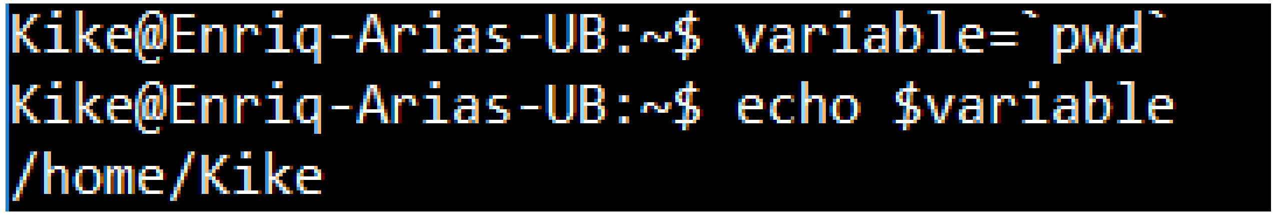
```
if variable is empty
    then
        new_variable=word
        variable=word
    else
        new_variable=variable
```

Introduction to Shell

- Export command makes visible the defined variables in a parent process to a child process
- Use: export variable
- Child process can modify the exported variable and remains unchanged for the parent process

Introduction to Shell

- It is possible to create a variable with the exit of a command by using `operator
- Use: variable=`command`
- Example:



```
Kike@Enriq-Arias-UB:~$ variable=`pwd`
Kike@Enriq-Arias-UB:~$ echo $variable
/home/Kike
```

Introduction to Shell

- Exercise 3
  - Use env command to show environment variables. Could you identify some of the variables introduced in this practice?
  - Show the content of HOME variable. Could you explain the meaning of that action?
  - Create a varible with your name and show it. After that, delete the variable and show it again. Is it really empty?
  - Use the previous variable. What would happen with the following command test_4=´$my$name'; echo $test_4
  - Create test_5=${test_4:-'Hello world'}. What happens?. Now créate test_6=${test_4:='Hello world'}. What happens now? Is there any difference?
  - Export a variable to child process. Create the child process (bash) and verify it. Modify the variable in child process. Back to parent process (exit). Which is the value of the exported varibla in parent and child process.
  - Create a variable called directory with the content of HOME directory

Defining complex Shell commands

- **;** : Will run one command after another has finished, irrespective of the outcome of the first
- **&** : This will run a command in the background, allowing you to continue working in the same shell
- **&&** : Used to build AND lists, it allows you to run one command only if another exited successfully
- **||** : Used to build OR lists, it allows you to run one command only if another exited unsuccessfully

Defining complex Shell commands

- Exercise 4
  - Create a file called new_file, list it and verify the errorlevel (using predefined variable ?)
  - If the new_file exists, explain the following actions (no_file means a file that has not been créate, it is just a name)
    - 1. ls new_file || echo “errorlevel$?”
    - 2. ls new_file && echo “errorlevel$?”
    - 3. ls no_file || echo “errorlevel $?”
    - 4. ls no_file && echo “errorlevel $?”

Redirections and pipes

- Standard input <
- Standard output > or >> (appending output)
  - Redirect to /dev/null is the output is not useful
- Error redirection 2>
- Pipe → command_1 | command_2

Internal Shell commands

- : → returns errorlevel 0
- . script_file → executes the commands in script_file
- exit → exits the Shell
- read variable → reads a variable from standard input
- shift → shifts the positional parameters in a Shell script

Internal Shell commands

- Exercise 5
  - Create a file with the content (ls) of the current directory (.) using output redirection (>)
  - Create a single command showing the number of directories and files in a given directory. Use: read, ls, wc -l and pipe

Shell bash

Enrique Arias
Universidad de Castilla–La Mancha