UNIX

Lecture 7 : Shell Programming (bash) (Part 1)

Filipe Vasconcelos[1]
[1]ESME, Lille, filipe.vasconcelo@esme.fr

ESME
PARIS•BORDEAUX•LILLE•LYON

Outline

Intended Learning Outcomes

ILO4

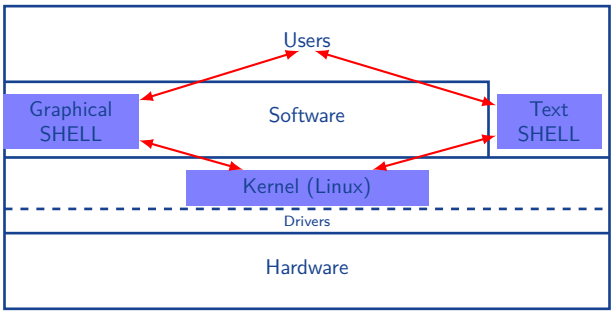
Produce, test, and verify the results of Shell scripts (in Bash) to perform tasks ranging from simple to complex algorithms.

Outline

Graphical Shell

GUI: Graphical User Interface

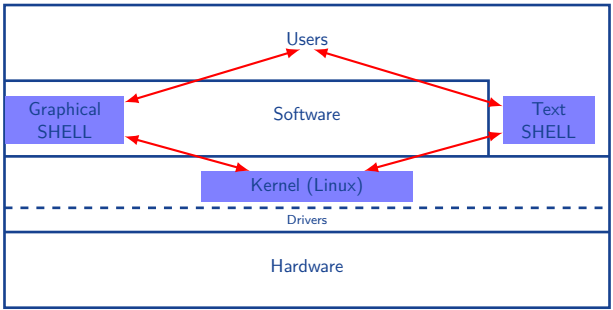
- There are several graphical environments in GNU/Linux:
  - GTK: Gnome, Unity, MATE, Xfce, etc.
  - Qt: KDE, LXQt, etc.
  - WSL (Windows): Xming



Text Shell

CLI: Command Line Interface

- There are different versions of command interpreters:
  - **bash: Bourne Again Shell**



Outline

SHELL Script

Definition

- Instructions are written in a file called a "shell script" or simply a script.
- Sequences of commands (batch) as opposed to the interactive mode of the command prompt.

Usage

- Direct access to the system and all commands.
- Automation of complex and repetitive tasks.
- Creating new commands (an advanced alias).
- System administration.

Special Considerations

- Caution! Strict syntax required!
- Caution! Readability is important!

Outline

First Script

Script Name

- By convention, shell scripts typically have the extension `.sh`.
- Example: `my_first_script.sh`

The First Line of the Script

- A script starts with the name of the script interpreter.
- The first two characters are `#!` (called "shabang").
- Then comes the path to the interpreter. Example: `#!/bin/bash`
- Fun Fact: You can start a script with other interpreters.
  - awk: `#!/usr/bin/awk`
  - perl: `#!/usr/bin/perl`
  - python: `#!/usr/bin/python3`

First Script

Comments

Commented lines begin with #.

```
#!/bin/bash
# This is my first bash script
# It displays the message "Hello World!"
echo "Hello World!"
```

Running the Script

- Give execution permission to the script using the command: `chmod u+x your_script_name`
- Then type `./` before the script name or enter the absolute path to the script to execute it.

Outline

Bash Variables

A Particular Syntax

- Every variable has a name and a value: `name=value` Example: `myVariable="Hello"`
- Note!!: There should be no space on either side of the equals sign.
- Caution: You can use uninitialized variables, which are then equal to an empty string.

Initializing a Variable from the Result of a Command

- Using backticks (AltGr+7). Example: `myVariable=‘ls‘`
- Using parentheses. Example: `myVariable=$(ls)` (more readable)

Accessing the Variable

- Use with the $ symbol or ${} Example: `echo $myVariable`
- Example: `echo $myVariable` (more readable)
- Display with `echo`

“Quotes”

- Apostrophes or “single quotes”: '
- Quotation marks or “ double quotes”: ""
- Backticks “ back quotes ”: `

Different Behaviors

- “ Single quotes” -> the variable is not displayed, and the $ is shown as is.
- “ Double quotes” -> ask bash to parse the message’s content. If it finds special symbols (like variables), it interprets them.
- “ Back quotes ” -> the entire string is interpreted.

`read` Command

It is possible to ask the user to enter text with the `read` command. This text will be immediately stored in a variable.
Example: `test.sh`

```
read first_name
echo "Your first name is:  $first_name"
```

`./test.sh`

```
Filipe
Your first name is:  Filipe
```

`read` Command

It is possible to simultaneously assign a value to multiple variables with `read`.

```
read last_name first_name
echo "Your last name is $last_name, and your first name is
$first_name"
```

`read -p "Message"`

```
read -p "Please enter your last name and first name:" last_name first_name
echo "Your last name is $last_name, and your first name is
$first_name"
```

Other options for `read`:

- `-n`: truncate the input string (e.g., "5 characters max").
- `-s`: hide the entered string (e.g., for passwords).

Mathematical Operations

In bash, variables are all strings of characters. Therefore, bash is not capable of performing operations directly.

`let` Command

Perform operations `+`, `-`, `*`, `/`, `**`, and `%` using `let`.

```
let "a=5"
let "a+=3"
let "b=3"
let "c=a+b"
echo "a = $a b = $b c = $c"
```

Mathematical Operations

$(())

Use `$(())` for mathematical operations.

```
a=5
b=3
c=$((a+b))
echo "a = $a b = $b c = $c"
```

For calculations involving decimal numbers, you can use the `bc` command. For example: `c=$(echo "scale=2;$a/$b"|bc)`.

Parameter Variables

Like all commands, bash scripts can also accept parameters. Thus, we could call our script like this. The different parameters are separated by spaces.

```
./params.sh param1 param2 param3
```

The values of these parameters can be retrieved in the script using predefined variables in bash:

- `$#` : Number of received parameters
- `$0` : Name of the script used
- `$*` : All parameters received as a single string
- `$@` : All parameters received as an array
- `$1, $2, $3, ...` : Refers to the i-th received parameter

Environment Variables

Currently, the variables you create in your bash scripts exist only within those scripts. Global variables from the `env` can be used in a script.

Useful Global Variables (in uppercase)

- `SHELL` : Indicates which type of shell is currently in use.
- `PATH` : A list of directories containing executables you want to run without specifying their directory.
- `HOME` : The location of your home directory.
- `PWD` : The directory you are currently in.
- `OLDPWD` : The directory you were in previously.

```
echo "Your home directory is: $HOME"
```