sourcemind

Core Concepts

Artur Davtyan
DevOps Engineer
Webb Fontaine Armenia

sourcemind

What is Bash Scripting ?

- Bash scripts are like movie scripts: A movie scripts tells actors what to say at what time, while Bash scripts tell Bash **what to do at what time**.
- Bash scripts are a **simple text file** containing a series of commands we want to automate running rather that running them.
- We need to remember to set the **execute** bit on file before we try to run the script, and that the script should start with a “**shebang**”.

sourcemind

Find a bash scripts

There are various shells with their own language syntax. Therefore, more complicated scripts will indicate a particular shell by specifying the absolute path to the interpreter as the first line, prefixed by **#!** as shown:

```
#!/bin/sh
echo "Hello, World!"
```

or

```
#!/bin/bash
echo "Hello, World!"
```

The two characters #! are traditionally called the hash and the bang respectively, which leads to the shortened form of "shebang" when they're used at the beginning of a script.

```
adavtyan@artur-lpt:~$ echo $PATH
/home/adavtyan/.sdkman/candidates/java/current/bin:/home/adavtyan/.kubectx:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
adavtyan@artur-lpt:~$ ls /usr/bin/lesspipe
/usr/bin/lesspipe
adavtyan@artur-lpt:~$ ls /usr/bin/lesspipe -l
-rwxr-xr-x 1 root root 8564 Մայ  3  2019 /usr/bin/lesspipe
adavtyan@artur-lpt:~$ file /usr/bin/lesspipe
/usr/bin/lesspipe: POSIX shell script, ASCII text executable
```

sourcemind

Editing Shell Scripts

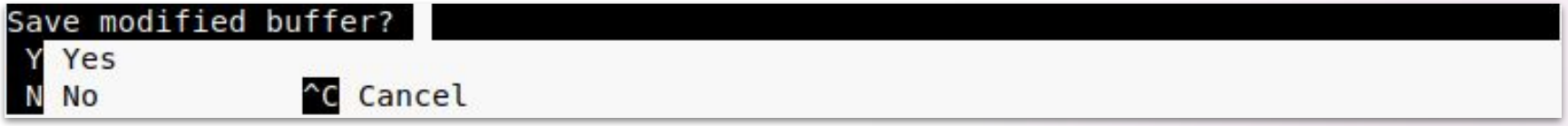
UNIX has many text editors. The GNU nano editor is a very simple editor well suited to editing small text files. The Visual Editor, vi, or its newer version, VI improved (vim), is a remarkably powerful editor but has a steep learning curve. We'll focus on nano.
Type nano test.sh and you'll see a screen similar to this:

The nano editor has few features to get you on your way. You simply type with your keyboard, using the arrow keys to move around and the delete/backspace button to delete text. Along the bottom of the screen you can see some commands available to you, which are context-sensitive and change depending on what you're doing.

sourcemind

Editing Shell Scripts

Note that the bottom-left option is ^X Exit which means “press **control** and **X** to exit”. Press **Ctrl** and **X** together and the bottom will change:



At this point, you can exit the program without saving by pressing the **N** key, or save first by pressing **Y** to save. The default is to save the file with the current file name. You can press the **Enter** key to save and exit.

You will be back at the shell prompt after saving. Return to the editor. This time press **Ctrl** and **O** together to save your work without exiting the editor. The prompts are largely the same, except that you’re back in the editor.

Other helpful commands you might need are:

Command	Description
Ctrl + **W**	search the document
Ctrl + **W**, then **Control** + **R**	search and replace
Ctrl + **G**	show all the commands possible
Ctrl + **Y/V**	page up / down
Ctrl + **C**	show the current position in the file and the file’s size

sourcemind

Working with Special Characters

Character	Function
“ “ or ‘ ’	Denotes whitespace. Single quotes preserve literal meaning; double quotes allow substitutions.
$	Denotes an expansion (for use with variables, command substitution, arithmetic substitution, etc.)
****	Escape character. Used to remove “specialness” from a special character.
#	Comments. Anything after this character isn’t interpreted
=	Assignment
[ ] or [[ ]]	Test; evaluates for either true or false
!	Negation
>>, >, <	Input/output redirection
|	Pipe. Sends the ouput of one command to the input of another
*** or ?**	Globs(aka, wildcards). ? is a wildcard for a single character.

sourcemind

Backing up a required files

Practical Work

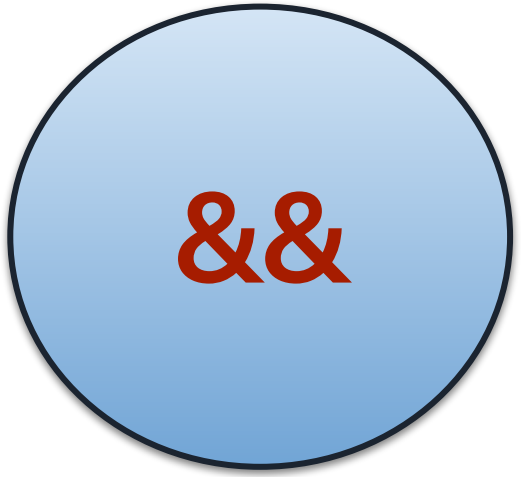
Approximate Duration: 30 minutes

Objectives:

- Backing up in CLI required files
- *Working with special character*
- *Write first script that use backing up a files*
- *Change script files permission*
- *Run script files*
- *Using environment variables in the script*

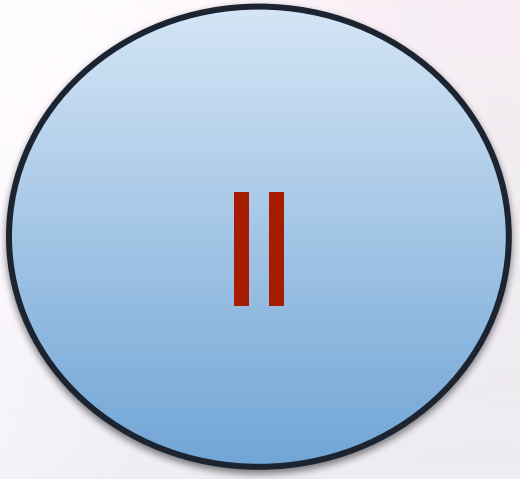
sourcemind

And/or Lists



And Lists

A string of commands where the next command is only executed if the previous command exited with a **status of zero**



Or Lists

A string of commands where the next command is only executed if the previous command exited with a **non-zero status**

sourcemind

Implementing And/or Lists

Practical Work

Approximate Duration: 15 minutes

Objectives:

- *Use and/or lists in the backup script*
- *`See status of the previous command and use it in script`*

sourcemind

Redirecting I/O

Redirecting I/O

Character	Function
>	Redirects to a file
	ls -lR > directory-tree.txt
>>	Redirects to a file, appending data
	echo mylabserver.com >> /etc/hosts
<	Redirects file as input for a command
	sort < unsorted_list.txt
< >	Redirects file as input for a command after that redirect to a file
	sort < unsorted_list.txt > sorted_list.txt

sourcemind

Utility Commands

Utility Commands

Command	Function
sort	Sorts input and prints a sorted output
uniq	Remove duplicate lines of data from the input stream
grep	Searches incoming lines for matching text
fmt	Receives incoming text and outputs formatted text
tr	Translates characters
head/tail	Outputs the first/last few lines of a file
sed	Stream Editor: More powerful than tr as a character translator
awk	An entire programing language designed for constructing filters. Very powerful and complex

sourcemind

Using Redirection I/O and Utility Commands

Practical Work

Approximate Duration: 15 minutes

Objectives:

- *Using Redirection I/O in backup script*
- *Using Utility Commands in backup script*

sourcemind

What Do Bash Variables Look Like?

An essential feature of programming is the ability to use a name or a label to refer to some other quantity: such as a value, or a command. This is commonly referred to as **variables**.

Variables can be used, at the very least, to make code more readable for humans:

What Do Bash Variables Look Like?

- **Important tip 01**

  Bash Variables do not have data types.

- **Important tip 02**

  All Bash variables start with $ when being referenced.

- **Important tip 03**

  When setting a variable, do not preface it with $.

sourcemind

What Do Bash Variables Look Like?

Practical Work

Approximate Duration: 15 minutes

Objectives:

- *Using Local and global variables in the terminal*
- *Using variables in the backup script*

sourcemind

Introducing Bash Functions

What is a Bash Function ?

A Bash function is essentially a set of commands that can be called numerous times. The purpose of a function is to help you make your bash scripts more readable and to avoid writing the same code repeatedly. Compared to most programming languages, Bash functions are somewhat limited.

The syntax for declaring a bash function is straightforward. Functions may be declared in two different formats:

First Format

```
#!/bin/bash

quit () {
    #Do Cleanup Stuff
    exit
    }
hello () { echo Hello!;}

hello
quit
```

Second Format

```
#!/bin/bash

function quit {
    #Do Cleanup Stuff
    exit
    }
function hello { echo Hello!;}

hello
quit
```

sourcemind

Introducing Bash Functions

Practical Work

Approximate Duration: 15 minutes

Objectives:

- *Using two function formats in bash script*
- *Define two format bash function in the backup script*
- *Define parameter in the backup script*

sourcemind

Working with Arrays in Bash

So like many other programming languages out there, Bash uses or can use arrays, and arrays are just a normal variable that has multiple values in that variable.

They're declared just like any other variable out there by specifying the variable name, equals, and then the value.

```
adavtyan@artur-lpt:~$ NUMBERS=(1 2 3 4 5 6)
adavtyan@artur-lpt:~$ echo $NUMBERS
1
```

If you want a specific index in the array, you have to specify or call the variable like this:

```
adavtyan@artur-lpt:~$ echo ${NUMBERS[2]}
3
```

As you can see from this example, Bash arrays start with index zero.

And finally, if you want to see everything in the array, you can use the at sign instead of an index number and that'll print out every value in the array.

```
adavtyan@artur-lpt:~$ echo ${NUMBERS[@]}
1 2 3 4 5 6
```

sourcemind

Working with Arrays in Bash

Practical Work

Approximate Duration: 15 minutes

Objectives:

- *Working with arrays in bash*
- *Define arrays*

sourcemind

Thank you for your attention !

Q&A