Bash Scripting Day 2

Mohamed Emary

January 28, 2025

1 Shell Scripting Basics

1.1 Checking Installed Shells

To know all the shells installed in your system, you can use the following command:

```
cat /etc/shells
```

1.2 Checking Current Shell

To know the shell you are currently using:

```
echo $SHELL
```

1.3 Command Location

Commands are stored in `/usr/bin` directory. To know the location of a command e.g. `chmod`:

```
type chmod
```

1.4 Environment Variables

`set` command displays all the environment variables. `$SHELL` is an environment variable.

1.5 Script Permissions

After creating any bash script add `r` and `x` permissions to it, so that it can be executed:

```
chmod +rx script.sh
```

Bash scripts should have `.sh` extension. Although it is not necessary, it is a good practice because it makes it easier to identify bash scripts if you have a lot of files in a directory or you are working with a team.

1.6 Shebang

The first line of any bash script should be:

```
#!/usr/bin/bash
```

This line is called `shebang`.

1.7 `PATH` Variable

`PATH` is all the directories where the system looks for commands. To know the `PATH`:

```
echo $PATH
```

1.8 Adding Directories to `PATH`

To add a directory to the `PATH`:

```
PATH=$PATH:/path/to/directory
```

This is a temporary change. To make it permanent, add the above line to `.bashrc` file.

2 Script Execution

2.1 Process Creation

Each script when executed, creates a new process. We can see running processes using `ps` command.

If you add `ps` command to a script, it will show the process of the script itself next to other processes.

2.2 Source Command

If you don't want the script to create a new process, you can use `source` command to run the script.

```
source script.sh
# or
. script.sh
```

3 Shell Compatibility

Some commands work on shells and don't work on others:

Bash	Ksh (Korn shell)
`echo`	`print`
`type`	`whence`

4 Variables

4.1 Variable Assignment

To create a variable:

```
variable=value
```

This will output `california`:

```
state=cal
echo ${state}ifornia # california
```

If you want to assign a value that contains spaces, you should use quotes:

```
variable="value with spaces"
```

Example:

```
name='Mohamed Emary'
echo $name # Mohamed Emary
```

4.2 Empty Variables

If we define an empty variable `x=`, and try to `echo` it, it will output nothing.

4.3 Integer Variables

If we define an empty variable that contains a calculated value, we should use `typeset -i num`:

```
typeset -i result
result=5+5
echo $result # 10
```

And if you want to have spaces in your calculation, you should use quotes:

```
typeset -i result
result="5 * 5"
echo $result # 25
```

If we put a string value inside that calculation, it will output `0`:

```
typeset -i result
result=myString
echo $result # 0
```

4.4 `let` and Calculation

To make bash understand that this is a number without using `typeset -i`, you can `let`:

```
i=1
let i=i+1
echo $i # 2
```

We can also use `((The calculation))`:

```
((i=i+1))
echo $i # 3
```

4.5 Common Environment Variables

Other example Environment variables include

- `PATH` - The directories where the system looks for commands.
- `HOME` - The home directory of the user.
- `PS1` - The prompt.

- `LOGNAME` - The login name of the user.
- `PS2` - The secondary prompt which is used when a command is continued on the next line.

4.6 Quotes

Difference between single quotes and double quotes:

- Single quotes: All characters inside it are treated as string.
- Double quotes: It treats everything inside it as string except `$`, `` ` ``, and `\`.
  - `$` is used to reference a variable.
  - `` ` `` is used to execute a command.
  - `\` is used to escape a character, for example if the character after it is `$`, it will be treated as a string not a variable.

4.7 Backticks

Anything inside backticks is executed as a command:

```
# Ex 1
echo "Date today is `date`"
# date today is Tue Jan 28 05:25:05 PM EET 2025
echo 'Date today is `date`'
# Date today is `date`

# Ex 2
echo \$HOME
# $HOME
```

5 Script Arguments

5.1 Accessing Arguments

We can pass arguments to our bash scripts in one of those ways:

- `$#` - Number of arguments
- `$*` - List of all arguments
- `$0` - Script name
- `$1`, `$2`, ... - first argument, second argument, ...
- `$?` - Return code of the last command

5.2 Passing Arguments

To pass arguments to a script:

```
script.sh arg1 arg2 arg3
```

5.3 Example

For example if we have the following script:

```
#!/usr/bin/bash
echo hello $*
echo the number of arguments is $#
echo the script name is $0
```

And we run it with the following command:

```
./script.sh Mohamed Ahmed
```

The output will be:

```
hello Mohamed Ahmed
the number of arguments is 2
the script name is ./script.sh
```

6 User Input

6.1 `read` Command

To create a script that takes user name and displays it:

```
#!/usr/bin/bash
echo "Enter your name:"
read name
echo "Hello $name"
```

6.2 `REPLY` Variable

If you don't give the variable a name in `read` command, it will be stored in `REPLY` variable:

```
#!/usr/bin/bash
echo "Enter your name:"
read
echo "Hello $REPLY"
```

7 Conditional Statements

7.1 If Statements

To test for conditions in bash scripts, we use `if` statement with one of the following operators:

- `-eq` - equal
- `-ne` - not equal
- `-gt` - greater than
- `-lt` - less than
- `-ge` - greater than or equal
- `-le` - less than or equal

7.2 Logical Operators

Logical operators:

- `-a` - AND operator

- `-o` - OR operator
- `!` - NOT operator

7.3 File Testing

Testing files:

- `-f` - file exists
- `-d` - directory exists
- `-l` - symbolic link

7.4 File Permissions

Checking file permissions:

- `-r` - readable
- `-w` - writable
- `-x` - executable

7.5 String Operators

String operators:

- `-z` - empty string
- `-n` - not empty string

7.6 Example If Statement

Example:

```
if [ $# -eq 0 ]; then
  echo $0 must take an argument
else
  echo hello $*
fi
```

The script above will check if the number of arguments is equal to 0, it will output `$0 must take an argument`, otherwise it will output `hello $*`.

- `$0` is the script name.
- `$*` is the list of all arguments.

8 File and Directory Checks

To check if a parameter is a file or a directory we use `-f` and `-d` operators:

```
echo Enter a file or directory name:
read name

if [ -f $name ]; then
  echo $name is a file
elif [ -d $name ]; then
```

```
  echo $name is a directory
else
  echo $name is neither a file nor a directory
fi
```

8.0.1 Test Command

In old bash scripts we used to use `test` command instead of `[ ]`:

```
if test $# -eq 0; then
  echo $0 must take an argument
else
  echo hello $*
fi
```

8.1 Multiple Parameters

To accept multiple parameters:

```
echo Enter your full name:
read first middle last
echo Your first name is $first
```