Bash Shell Scripting Cheatsheet

Author: Abanoub Hanna

LinkedIn: Abanoub Hanna

Github: Gists & source code apps

YouTube: Abanoub Hanna

Twitter: Abanoub Hanna

Shell Scripting Crash Course - *Cheatsheet* for beginners

This is a bash shell scripting which can be found on **Unix**, **Linux** and **Mac**. You can install bash on the Linux subsystem on **Windows** too.

the first line is `#! /bin/bash` because the *bash* program is in */bin/bash*, you can know the path where the *bash* is by this command `which bash`

print data or text on the screen

You can use `echo Hello, World!` or `echo "Hello, world!"`

Variables & Print out them

```
NAME = "Abanoub"
echo "My name is $NAME"
```

or you can use this

```
NAME = "Abanoub"
echo "My name is ${NAME}"
```

Get Data From User

```
read -p "Enter your name: " NAME
echo "Hello, $NAME!"
```

If Statement

The simple if statement syntax of bash script:

```
if ["$NAME" == "Abanoub"]
then
    echo "Yourname is Abanoub!"
fi
```

NOTE THAT: the end of **if** is **fi** the reverse letters of **if**.

The **if else** statement syntax in bash script:

```
if ["$NAME" == "Abanoub"]
then
    echo "Yourname is Abanoub!"
else
    echo "Yourname is NOT Abanoub"
fi
```

The **else if** (*elif*) condition statement syntax in bash scripting:

```
if ["$NAME" == "Abanoub"]
then
    echo "Yourname is Abanoub!"
elif ["$NAME" == "Jack"]
then
    echo "Yourname is Jack!"
else
    echo "Yourname is NOT Abanoub NOR Jack!"
fi
```

Logic Comparisons

You can use those operators:

Logic Operator	Meaning
-eq	**equal to** (the same meaning of == in other programming languages)
-ne	**not equal** (the same meaning of != in other programming languages)
-gt	**greater then** (the same meaning of > in other programming languages)
-ge	**greater than or equal to** (the same meaning of >= in other programming languages)
-lt	**less than** (the same meaning of < in other programming languages)
-le	**less than or equal to** (the same meaning of <= in other programming language)

and use them like this:

```
NUM1 = 3
NUM2 = 5
if ["$NUM!" -gt "$NUM2"]
then
    echo "$NUM1 is greater then $NUM2"
fi
```

File Conditions

these are the **file condition flags**:

symbol	meaning
-d	is directory?
-e	exists? (usually we use -f instead)
-f	a file?
-g	is group id set?
-r	readable?
-s	non-zero size?
-u	user id is set?
-w	writable?
-x	executable?

and use them like this:

```
FILE = "test.txt"
if [ -f "$FILE" ]
then
    echo "$FILE is a file"
else
    echo "$FILE is NOT a file"
fi
```

Case Statement

Case is called **switch case** in other languages, and some modern languages call it **when case** such as *Kotlin* programming language.

Here is the case statement in bash scripting:

```
read -p "Are you 25? Y/N" ANSWER
case "$ANSWER" in
    [yY]|[yY][eE][sS])
        echo "Your age is mine :)"
        ;;
    [nN]|[nN][oO])
        echo "Nooo, your age is different than mine :("
        ;;
    *)
        echo "Please enter y/yes or n/no"
        ;;
    esac
```

Note that:

`[nN]` is a way of giving two probabilities small **n** or capital **N**.

`[yY][eE][sS]` is the word **yes** or **YES** or any combination of small and capital letters to compose a *YeS* word.

`*)` this is the default option in the case statement which is called *default* in other programming languages.

`esac` is the closing of the case statement as it is the reversed letters of `case` . This is the way of ending statements in bash script.

For Loop

```
NAMES = "Abanoub Jack John Smith"
for NAME in $NAMES
    do
    echo "Hello, $NAME"
done
```

Here is a script to rename all text files `*.txt` at once by a script:

```
FILES = $(ls *.txt)
NEW = "new"
for FILE in $FILES
    do
    echo "Renaming $FILE to new-$FILE"
    mv $FILE $NEW-$FILE
done
```

While Loop

Here is a while loop to read `nfile.txt` line by line.

```
LINE = 1
while read -r CURRENT_LINE
    do
    echo "$LINE: $CURRENT_LINE"
    ((LINE++))
done < "./nfile.txt"
```

Bash Script Function Syntax

```
function sayHello(){
    echo "Hello, World!"
}
sayHello
```

We created a function to print out `Hello, World!` and call it to occur!

Bash Script Functions with Parameters

Here is how to write a functions with params in NAMES = "Abanoub Jack John Smith"bash scripting:

```
function greet(){
    echo "Hello, I am $1 and I am $2"
}
greet "Abanoub" "25"
```

The `$1` is the first parameter, and `$2` is the second parameter. So when we call the function, we should specify the two parameters in the same order like this `greet "Abanoub" "25"`. This means that `"Abanoub"` is the first param `$1` and `"25"` is the second param `$2`.

Final Tips

Now, you learned the syntax of bash scripting, but you need to learn the **bash commands** (I will make a cheatsheet for bash commands later). You will use the bash commands in the bash scripting syntax to build the script you want to run automatically (terminal app).

License