# Bash

Cheatsheet

P J

# Contents

# 1   Bash Cheatsheet

A Bash script is a plain text file which contains a series of commands.

These commands are a mixture of commands we would normally type ouselves on the command line (such as `ls` or `cp` for example).

> Anything you can run normally on the command line can be put into a script and it will do exactly the same thing. Similarly, anything you can put into a script can also be run normally on the command line and it will do exactly the same thing.

You don't need to change anything. Just type the commands as you would normally and they will behave as they would normally. It's just that instead of typing them at the command line we are now entering them into a plain text file. In this sense, if you know how to do stuff at the command line then you already know a fair bit in terms of Bash scripting.

It is convention to give files that are Bash scripts an extension of `.sh`.

As you would be aware, Linux is an extensionless system so a script doesn't necessarily have to have this characteristic in order to work.

## 2 Variables

Variables are one of those things that are actually quite easy to use but are also quite easy to get yourself into trouble with if you don't properly understand how they work. As such there is a bit of reading in this section but if you take the time to go through and understand it you will be thankful you did later on when we start dabbling in more complex scripts.

### 2.1 Define Variables

Variables may have their value set in a few different ways. The most common are to set the value directly and for its value to be set as the result of processing by a command or program.

```bash
#!/bin/bash

VarName=$(ls /tmp)
variable_name="Data List"
echo $VarName
echo $variable_name
```

**Note there is no space on either side of the equals ( = ) sign.**

Variable names may be uppercase or lowercase or a mixture of both but Bash is a case sensitive environment so whenever you refer to a variable you must be consistent in your use of uppercase and lowercase letters. You should always make sure variable names are descriptive. This makes their purpose easier for you to remember.

To read the variable we then place its name (preceded by a $ sign) anywhere in the script we would like. Before Bash interprets (or runs) every line of our script it first checks to see if any variable names are present. For every variable it has identified, it replaces the variable name with its value. Then it runs that line of code and begins the process again on the next line.

### 2.2 Command-line Argumets

When we run a program on the command line you would be familiar with supplying arguments after it to control its behaviour. For instance we could run the command `ls -l /etc`. `-l` and `/etc` are both command line arguments to the command `ls`. We can do similar with our bash scripts. To do this we use the variables `$1` to represent the first command line argument, `$2` to represent the second command line argument and so on. These are automatically set by the system when we run our script so all we need to do is refer to them.

```bash
#!/bin/bash
# A simple copy script
cp $1 $2
# Let's verify the copy worked
echo Details for $2
ls -lh $2
```

## 2.3   Other Special Variables

- $0 - The name of the Bash script.
- $1, $2, $3, etc.. - The first, second, third, etc.. argument of the script.
- $# - How many arguments were passed to the script.
- $@ - All the arguments supplied to the Bash script.
- $? - The exit status of the most recently run process.
- $$ - The process ID of the current script.
- $USER - Username of the user running the script.
- $HOSTNAME - Hostname of the machine the script is running on.
- $SECONDS - Number of seconds since the script was started.
- $RANDOM - Returns a different random number each time is it referred to.
- $LINENO - Returns the current line number in the Bash script.

Some of these variables may seem useful to you now. Others may not. As we progress to more complex scripts in later sections you will see examples of how they can be useful.

If you type the command env on the command line you will see a listing of other variables which you may also refer to.

Here is a simple example to illustrate their usage:

```
1  #!/bin/bash
2
3  # A simple variable example
4  myvariable=Hello
5  anothervar=Fred
6
7  echo $myvariable $anothervar
8  echo
9
10 sampledir=/etc
11 ls $sampledir
```

```
$ ./simplevariables.sh
Hello Fred

a2ps.cfg aliases alsa.d ...
$_
```

Variables can be useful for making our scripts easier to manage. Maybe our script is going to run several commands, several of which will refer to a particular directory. Rather than type that directory out each time we can set it once in a variable then refer to that variable. Then if the required directory changes in the future we only need to update one variable rather than every instance within the script.

## 2.4   Quotes

In the example above we kept things nice and simple. The variables only had to store a single word. When we want variables to store more complex values however, we need to make use of quotes. This is because under normal circumstances Bash uses a space to determine separate items.

```
$ myvar=Hello World
-bash: World: command not found
$_
```

**Remember, commands work exactly the same on the command line as they do within a script.**

> Because commands work exactly the same on the command line as in a script it can sometimes be easier to experiment on the command line.

When we enclose our content in quotes we are indicating to Bash that the contents should be considered as a single item. You may use single quotes (') or double quotes (").

- Single quotes will treat every character literally.
- Double quotes will allow you to do substitution (that is include variables within the setting of the value).

```
1  myvar='Hello World'
2  echo $myvar # -> Hello World
3  newvar="More $myvar"
4  echo $newvar # -> More Hello World
5  newvar='More $myvar'
6  echo $newvar # -> More $myvar
```

## 2.5   Command Substitution

Command substitution allows us to take the output of a command or program (what would normally be printed to the screen) and save it as the value of a variable. To do this we place it within practices, preceded by a $ sign.

```
1  myvar=$( ls /etc | wc -l )
2  echo There are $myvar entries in the directory /etc
3  # -> There are 126 entries in the directory /etc
```

```
$ ls
bin Documents Desktop ...
Downloads public_html ...
$ myvar=$( ls )
$ echo $myvar
bin Documents Desktop Downloads public_html ...
$_
```