Bash Shell Scripting

Bash is a version of the classic Unix shell with many enhancements. Bash is the default shell installed on GNU/Linux distributions and many other Unix-style systems. This cheat sheet covers some useful concepts in Bash scripting.
If a command in the examples produces output, the output is shown on the same line, separated from the command by a hash or pound sign (#).

BASH SCRIPT HEADER

```
#!/usr/bin/env bash
echo "Hello World"
```

VARIABLES & STRINGS

```
#!/usr/bin/env bash
MSG="Hello World"
echo "$MSG Alex" # Hello World Alex
echo '$MSG Alex' # $MSG Alex
```

String Manipulation

```
MSG="hello world"
# Replace
echo ${MSG/w/W} #  hello World
echo ${MSG//[a-zA-Z]/X} #  XXXXX XXXXX
# Substring
echo ${MSG:0:5} # hello
echo ${MSG%world} # hello
echo ${MSG#hello} # world
# Uppercase
echo ${MSG^} # Hello world
echo ${MSG^^} # HELLO WORLD
MSG="HELLO WORLD"
echo ${MSG,} # hELLO WORLD
echo ${MSG,,} # hello world
# Alternative
echo ${MSG:-val} # HELLO WORLD
echo ${FOO:-val} # val
```

FUNCTIONS

```
helloworld() {
  echo "Number of arguments $#" # 2
  echo "Hello World $1 from $2" # Hello World Alex from Bash
}
helloworld "Alex" "Bash"
```

Returning Values

```
helloworld () {
  return 46
}
helloworld
echo $? # 46
```

Bash can return only a status code. To return a string, use command substitution:

```
helloworld() {
echo 'My return string!'
}

msg=$(helloworld)
echo $msg
```

COLLECTIONS

Arrays

```
names=('Alex' 'Ada' 'Alexandra')
names+=('Soto') # Appends element
unset names[3] # Removes element

echo ${names[0]} # Alex
echo ${names[@]} # Alex Ada Alexandra
echo ${#names[@]} # 3
```

Maps

```
declare -a score
score[alex]="1"
score[edson]="2"
score[sebi]="3"
unset score[alex]   # Delete alex entry

echo ${!score[@]}  # alex edson sebi
echo ${score[@]}   # 2 1 3
echo ${#score[@]}  # 3
```

CONDITIONALS

```
if [[ $a -gt 4 ]]; then
  echo "$a is greater than 4"
elif [[ $a -lt 4 ]]; then
  echo "$a less than 4"
else
  echo "$a is equal 4"
fi
```

Numeric Conditions

`[[ NUM -eq NUM ]]`	Equal
`[[ NUM -ne NUM ]]`	Not equal
`[[ NUM -lt NUM ]]`	Less than
`[[ NUM -le NUM a]]`	Less than or equal to
`[[ NUM -gt NUM ]]`	`[[ NUM -ge NUM ]]` Greater than or equal to

Greater than

String Conditions

[[STRING == STRING]]	Equal
[[STRING != STRING]]	Not Equal
[[-z STRING]]	Empty string
[[-n STRING]]	Not empty string
[[STRING =~ STRING]]	Regular expression match

File Conditions

[[-f FILE]]	Is a file
[[-d FILE]]	Is a directory
[[-e FILE]]	Exists
[[-r -w -x FILE]]	Is readable, Writable, executable
[[-h FILE]]	Is symbolic link

Boolean conditions:
a|[[! EXPR]] |Not
a|[[BOOL && BOOL]] |And
a|[[BOOL || BOOL]] |OR

LOOPS

```
for ((i = 0 ; i < 10 ; i++)); do
  echo "Hello World $i"
done
```

Range

```
for i in {1..5}; do
    echo "Hello World $i"
done
```

Collections

Print all elements from a plain array:

```
for i in "${names[@]}"; do
  echo "Hello $i"
done
```

Print keys and values of all elements from a key/value array:

```
for key in "${!score[@]}"; do
  echo $key
done

for val in "${score[@]}"; do
  echo $val
done
```

Files

```
for i in /tmp/*.txt; do
  echo $i
done

cat /tmp/hello.txt | while read line; do
  echo $line
done
```

While

```
x=1;
while [ $x -le 5 ]; do
  echo "Hello World"
done
```

EXECUTING COMMANDS

Execute a command and check the exit status:

```
cat /tmp/hello.txt

if [ $? -eq 0 ]
then
  echo "OK"
else
  echo "KO"
fi
```

To get the output of a command, surround the call with "\`" character:

```
lines=(`cat "/tmp/hello.txt"`)
lines="$(cat "/tmp/hello.txt)"
```

USEFUL SNIPPETS

Getting the Script Directory

```
DIR="${0%/*}"
```

Reading CLI Arguments:

```
echo "$1 $2"
########
execute.sh "Hello" "Alex"
# Hello Alex
```

Print Output

```
printf "\n\n######## Deploying ########\n"
```

Read Input

```
echo -n "Enter name: "
read ans
echo $ans
```

Create File with Content

```
echo "
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
    providers:
```

```
    - aescbc:
        keys:
        - name: key1
          secret: b6sjdRWAPhtacXo8mO1cfgVYWXzwuls3T3NQOo4TBhk=
    - identity: {}
"  | tee /var/lib/minikube/certs/encryptionconfig.yaml
```

Subshell

A shell script can launch subshells. These subshells let the script do parallel processing, in effect executing multiple subtasks simultaneously.

```
(
# Inside parentheses, and therefore a subshell . . .
while [ 1 ]   # Endless loop.
do
  echo "Subshell"
done
)
```

Run the following command in a new terminal:

```
ps -ef | grep execute.sh
501  4286   641   0 11:17AM ttys007    0:00.00 /bin/bash ./execute.sh
501  4287  4286   0 11:17AM ttys007    0:07.67 /bin/bash ./execute.sh
```

Two processes are started. The first one (4286) as parent of the second.