cheatsheet

Example

```
#!/bin/bash

NAME="Payam"
echo "Hello $NAME!"

exit 0
```

Variables:

```
varname=value                # defines a variable
varname=value command        # defines a variable to be in the environment o
echo $varname                # checks a variable's value
read <varname>               # reads a string from the input and assigns it
let <varname> = <equation>   # performs mathematical calculation using opera
export VARNAME=value         # defines an environment variable (will be ava
```

```
#Special shell variables
echo $$                      # prints process ID of the current shell
echo $!                      # prints process ID of the most recently invoke
echo $?                      # displays the exit status of the last command
echo $0                      # display Filename of the shell script
```

Quoting:

```
\c              #Take character c literally.
`cmd`           #Run cmd and replace it in the line of code with its output.
"whatever"      #Take whatever literally, after first interpreting $, `...`,
'whatever'     #Take whatever absolutely literally.

#Example:
match=`ls *.bak`        #Puts names of .bak files into shell variable match
echo \*                 #Echos * to screen, not all filename as in:  echo *
echo '$1$2hello'        #Writes literally $1$2hello on screen.
echo "$1$2hello"        #Writes value of parameters 1 and 2 and string hell
```

Redirection

```
python hello.py > output.txt   # stdout to (file)
python hello.py >> output.txt  # stdout to (file), append
python hello.py 2> error.log   # stderr to (file)
python hello.py 2>&1           # stderr to stdout
python hello.py 2>/dev/null    # stderr to (null)
python hello.py &>/dev/null    # stdout and stderr to (null)
python hello.py < foo.txt      # feed foo.txt to stdin for python
```

Brace expansion

```
{A,B} Same as A B
{A,B}.js  Same as A.js B.js
{1..5}  Same as 1 2 3 4 5
```

Parameter expansions

Basics

```
name="John"
echo ${name}
echo ${name/J/j}    #=> "john" (substitution)
echo ${name:0:2}    #=> "Jo" (slicing)
echo ${name::2}     #=> "Jo" (slicing)
echo ${name::-1}    #=> "Joh" (slicing)
echo ${name:(-1)}   #=> "n" (slicing from right)
echo ${name:(-2):1} #=> "h" (slicing from right)
echo ${food:-Cake}  #=> $food or "Cake"

length=2
echo ${name:0:length}  #=> "Jo"
```

```
STR="/path/to/foo.cpp"
echo ${STR%.cpp}    # /path/to/foo
echo ${STR%.cpp}.o  # /path/to/foo.o
echo ${STR%/*}      # /path/to

echo ${STR##*.}     # cpp (extension)
echo ${STR##*/}     # foo.cpp (basepath)

echo ${STR#*/}      # path/to/foo.cpp
echo ${STR##*/}     # foo.cpp

echo ${STR/foo/bar} # /path/to/bar.cpp
STR="Hello world"
echo ${STR:6:5}   # "world"
echo ${STR: -5:5}  # "world"
SRC="/path/to/foo.cpp"
BASE=${SRC##*/}   #=> "foo.cpp" (basepath)
DIR=${SRC%$BASE}  #=> "/path/to/" (dirpath)
```

Substitution

```
${FOO%suffix} Remove suffix
${FOO#prefix} Remove prefix
${FOO%%suffix}  Remove long suffix
${FOO##prefix}  Remove long prefix
${FOO/from/to}  Replace first match
${FOO//from/to} Replace all
${FOO/%from/to} Replace suffix
${FOO/#from/to} Replace prefix
```

Length

```
${#FOO} Length of $FOO
```

Default Values

```
${FOO:-val} $FOO, or val if unset (or null)
${FOO:=val} Set $FOO to val if unset (or null)
${FOO:+val} val if $FOO is set (and not null)
${FOO:?message} Show error message and exit if $FOO is unset (or null)

#Omitting the : removes the (non)nullity checks,
#e.g. ${FOO-val} expands to val if unset otherwise $FOO.
```

Comment

```
# Single line comment
: '
This is a
multi line
comment
'
```

Substrings

```
${FOO:0:3}  Substring (position, length)
${FOO:(-3):3} Substring from the right
```

Manipulations

```
STR="HELLO WORLD!"
echo ${STR,}   #=> "hELLO WORLD!" (lowercase 1st letter)
echo ${STR,,}  #=> "hello world!" (all lowercase)

STR="hello world!"
echo ${STR^}   #=> "Hello world!" (uppercase 1st letter)
echo ${STR^^}  #=> "HELLO WORLD!" (all uppercase)
```

Conditionals:

Test Operators

In Bash, the `test` command takes one of the following syntax forms:

- **test EXPRESSION**
- **[EXPRESSION]**
- **[[EXPRESSION]]**

To make the script portable, prefer using the old test `[` command which is available on all POSIX shells. The new upgraded version of the `test` command `[[` (double brackets) is supported on most modern systems using Bash, Zsh, and Ksh as a default shell. To negate the test expression, use the logical `NOT` (`!`) operator.

Checking Numbers

Note that a shell variable could contain a string that represents a number. If you want to check the numerical value use one of the following:

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

Checking Strings

```
[[ -z STRING ]] Empty string
[[ -n STRING ]] Not empty string
[[ STRING == STRING ]]  Equal
[[ STRING != STRING ]]  Not Equal

```

Checking files

```
[[ -e FILE ]] Exists
[[ -r FILE ]] Readable
[[ -h FILE ]] Symlink
[[ -d FILE ]] Directory
[[ -w FILE ]] Writable
[[ -s FILE ]] Size is > 0 bytes
[[ -f FILE ]] File
[[ -x FILE ]] Executable
[[ FILE1 -nt FILE2 ]] 1 is more recent than 2
[[ FILE1 -ot FILE2 ]] 2 is more recent than 1
[[ FILE1 -ef FILE2 ]] Same files
```

More conditions:

```
[[ -o noclobber ]]  If OPTIONNAME is enabled
[[ ! EXPR ]]  Not
[[ X && Y ]]  And
[[ X || Y ]]  Or
```

if statement:

```
#if Statement

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
fi
```

```
#if..else Statement

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
else
  echo "The variable is equal or less than 10."
fi
```

```
#if..elif..else Statement

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
elif [[ $VAR -eq 10 ]]
then
  echo "The variable is equal to 10."
else
  echo "The variable is less than 10."
fi
```

```
# Nested if Statements
echo -n "Enter the first number: "
read VAR1
echo -n "Enter the second number: "
read VAR2
echo -n "Enter the third number: "
read VAR3

if [[ $VAR1 -ge $VAR2 ]]
then
  if [[ $VAR1 -ge $VAR3 ]]
  then
    echo "$VAR1 is the largest number."
  else
    echo "$VAR3 is the largest number."
  fi
else
  if [[ $VAR2 -ge $VAR3 ]]
  then
    echo "$VAR2 is the largest number."
  else
    echo "$VAR3 is the largest number."
  fi
fi
```

Loops:

for:

```
#basic for loop
for i in 1 2 3 4 5
do
   echo "Welcome $i times"
done
```

```
#Basic for loop
for i in /etc/rc.*; do
  echo $i
done
```

```
#Ranges
for i in {1..5}; do
    echo "Welcome $i"
done
```

```
#C-Like for loop
for ((i = 0 ; i < 100 ; i++)); do
  echo $i
done
```

```
#with step size
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

while:

```
n=1

while [ $n -le 5 ]
do
  echo "Welcome $n times."
  n=$(( n+1 ))
done
```

```bash
#Using ((expression)) Format With The While Loop
n=1
while (( $n <= 5 ))
do
  echo "Welcome $n times."
  n=$(( n+1 ))
done
```

```bash
#for ever
while true; do
  ...
done
```

```bash
# Reading a test file:
###example1/2:
cat /etc/resolv.conf | while read line; do
  echo $line
done

###example2/2:
file=/etc/resolv.conf
while IFS= read -r line
do
  echo $line
done < "$file"

### Reading A Text File With Separate Fields:
file=/etc/resolv.conf
# set field separator to a single white space
while IFS=' ' read -r f1 f2
do
  echo "field # 1 : $f1 ==> field #2 : $f2"
done < "$file"
```

Until:

```bash
#!/bin/bash

counter=0

until [ $counter -gt 5 ]
do
```

```
  echo Counter: $counter
  ((counter++))
done
```

Case:

```
Case/switch
case "$1" in
  start | up)
    vagrant up
    ;;

  *)
    echo "Usage: $0 {start|stop|ssh}"
    ;;
esac
```

Functions:

```
# Defining functions:
myfunc() {
    echo "hello $1"
}
# Same as above (alternate syntax)
function myfunc() {
    echo "hello $1"
}
myfunc "John"
```

```
#Returning values:
myfunc() {
    local myresult='some value'
    echo $myresult
}
result="$(myfunc)"
```

```
#Raising errors:
myfunc() {
  return 1
}
if myfunc; then
  echo "success"
else
  echo "failure"
fi
```

```
#Arguments:
$#  Number of arguments
$*  All arguments
$@  All arguments, starting from first
$1  First argument
$_  Last argument of the previous command
```

Arrays

```
Defining arrays
Fruits=('Apple' 'Banana' 'Orange')
Fruits[0]="Apple"
Fruits[1]="Banana"
Fruits[2]="Orange"
```

```
Operations
Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits+=('Watermelon')                  # Also Push
Fruits=( ${Fruits[@]/Ap*/} )            # Remove by regex match
unset Fruits[2]                         # Remove one item
Fruits=("${Fruits[@]}")                 # Duplicate
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate
lines=(`cat "logfile"`)                 # Read from file
```

```
Working with arrays
echo ${Fruits[0]}           # Element #0
echo ${Fruits[-1]}          # Last element
echo ${Fruits[@]}           # All elements, space-separated
echo ${#Fruits[@]}          # Number of elements
echo ${#Fruits}             # String length of the 1st element
echo ${#Fruits[3]}          # String length of the Nth element
echo ${Fruits[@]:3:2}       # Range (from position 3, length 2)
echo ${!Fruits[@]}          # Keys of all elements, space-separated
```

```
Iteration
for i in "${arrayName[@]}"; do
  echo $i
done
```

Dictionaries:

```
Defining
declare -A sounds
sounds[dog]="bark"
sounds[cow]="moo"
sounds[bird]="tweet"
sounds[wolf]="howl"
```

```
Working with dictionaries
echo ${sounds[dog]} # Dog's sound
echo ${sounds[@]}   # All values
echo ${!sounds[@]}  # All keys
echo ${#sounds[@]}  # Number of elements
unset sounds[dog]   # Delete dog
```

```
Iteration
Iterate over values
for val in "${sounds[@]}"; do
  echo $val
done
Iterate over keys
for key in "${!sounds[@]}"; do
  echo $key
done
```

Debugging

```
bash -n scriptname  # don't run commands; check for syntax errors only
set -o noexec       # alternative (set option in script)

```

```
bash -v scriptname  # echo commands before running them
set -o verbose      # alternative (set option in script)

bash -x scriptname  # echo commands after command-line processing
set -o xtrace       # alternative (set option in script)
```

Miscellaneous:

```
#Numeric calculations
$((a + 200))      # Add 200 to $a
$(($RANDOM%200))  # Random number 0..199
```

```
#Inspecting commands
command -V cd
#=> "cd is a function/alias/whatever"
```

```
#Heredoc:
cat <<END
hello world
END
```

```
#printf:
printf "Hello %s, I'm %s" Sven Olga
#=> "Hello Sven, I'm Olga

printf "1 + 1 = %d" 2
#=> "1 + 1 = 2"

printf "This is how you print a float: %f" 2
#=> "This is how you print a float: 2.000000"
```

```
#Reading input
echo -n "Proceed? [y/n]: "
read ans
echo $ans
```

```

#Reading  Just one character:
read -n 1 ans
```

```
#Getting options
while [[ "$1" =~ ^- && ! "$1" == "--" ]]; do case $1 in
  -V | --version )
    echo $version
    exit
    ;;
  -s | --string )
    shift; string=$1
    ;;
  -f | --flag )
    flag=1
    ;;
esac; shift; done
if [[ "$1" == '--' ]]; then shift; fi
```

```
#Check for command’s result
if ping -c 1 google.com; then
  echo "It appears you have a working internet connection"
fi
```

Payam Borosan.Goodluck