Bash Cheatsheet

A quick reference for Bash scripting, covering essential syntax, commands, and best practices for writing efficient and robust scripts.

Basics & Syntax

Getting Started

Shebang:

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

This line specifies the interpreter for the script.

Variables:

```
name="John"
echo $name
echo "$name"
echo "${name} and Jane"
```

Variable assignment and usage. Always quote variables unless you need wildcard expansion or command substitution.

String Quotes:

```
name="John"
echo "Hi $name"  #=> Hi John
echo 'Hi $name'  #=> Hi $name
```

Double quotes allow variable interpolation, single quotes do not.

Command Substitution:

```
echo "I'm in $(pwd)"
echo "I'm in `pwd`"  # obsolescent
```

Executes a command and inserts its output.

Conditional Execution:

```
git commit && git push
git commit || echo "Commit failed"
```

Execute commands based on the success or failure of previous commands.

Strict Mode:

```
set -euo pipefail
IFS=$'\n\t'
```

Enable strict mode to catch errors and improve script reliability.

Comments

Single-line comment:

```
# This is a single-line comment
```

Multi-line comment:

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

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`
`{{1..3},{7..9}}`	Same as `1 2 3 7 8 9`

Parameter Expansion & Manipulation

Parameter Expansion

Basics:

```
name="John"
echo "${name}"
echo "${name/J/j}"    #=> "john" (substitution)
echo "${name:0:2}"    #=> "Jo" (slicing)
```

More basics:

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

Path manipulation:

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

More path manipulation:

```
str="/path/to/foo.cpp"
echo "${str##*.}"     # cpp (extension)
echo "${str##*/}"     # foo.cpp (basepath)
```

More string manipulation:

```
str="Hello world"
echo "${str:6:5}"    # "world"
echo "${str: -5:5}"  # "world"
```

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

String Manipulation

Case conversion:

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

More case conversion:

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

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)

Loops & Functions

Loops

Basic `for` loop:

```
for i in /etc/rc.*; do
  echo "$i"
done
```

C-style `for` loop:

```
for ((i = 0 ; i < 100 ; i++)); do
  echo "$i"
done
```

Ranges in `for` loop:

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

Ranges with step size:

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

Reading lines from a file:

```
while read -r line; do
  echo "$line"
done <file.txt
```

Infinite loop:

```
while true; do
  #...
done
```

Functions

Defining a function:

```
myfunc() {
    echo "hello $1"
}
```

Alternative syntax:

```
function myfunc {
    echo "hello $1"
}
```

Calling a function:

```
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
```

Function Arguments:

```
echo $#
echo $@
```

Arguments

`$#`	Number of arguments
`$*`	All positional arguments (as a single word)
`$@`	All positional arguments (as separate strings)
`$1`	First argument
`$_`	Last argument of the previous command

Conditionals, Arrays, Dictionaries

Conditionals

String conditions:

```
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
fi
```

Numeric conditions:

```
if (( $a < $b )); then
   echo "$a is smaller than $b"
fi
```

File conditions:

```
if [[ -e "file.txt" ]]; then
  echo "file exists"
fi
```

Conditions

`[[ -z STRING ]]`	Empty string
`[[ -n STRING ]]`	Not empty string
`[[ STRING == STRING ]]`	Equal
`[[ STRING != STRING ]]`	Not Equal
`[[ NUM -eq NUM ]]`	Equal
`[[ NUM -ne NUM ]]`	Not equal

Arrays

Defining an array:

```
Fruits=('Apple' 'Banana' 'Orange')


Fruits[0]="Apple"
Fruits[1]="Banana"
Fruits[2]="Orange"
```

Accessing array elements:

```
echo "${Fruits[0]}"           # Element #0
echo "${Fruits[-1]}"          # Last element
echo "${Fruits[@]}"           # All elements,
space-separated
```

Array operations:

```
Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits+=('Watermelon')                  # Also
Push
unset Fruits[2]                         #
Remove one item
```

Iterate array:

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

Dictionaries

Defining a dictionary:

```
declare -A sounds
sounds[dog]="bark"
sounds[cow]="moo"
```

Accessing dictionary elements:

```
echo "${sounds[dog]}" # Dog's sound
echo "${sounds[@]}"   # All values
echo "${!sounds[@]}"  # All keys
```

Dictionary operations:

```
unset sounds[dog]     # Delete dog
```

Iterating dictionary (values):

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

Iterating dictionary (keys):

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

Options & Miscellaneous

Options

Setting options:

```
set -o noclobber  # Avoid overlay files (echo
"hi" > foo)
set -o errexit    # Used to exit upon error
set -o pipefail   # Unveils hidden failures
set -o nounset    # Exposes unset variables
```

Glob options:

```
shopt -s nullglob    # Non-matching globs are
removed
shopt -s failglob    # Non-matching globs
throw errors
shopt -s nocaseglob  # Case insensitive globs
```

History

Commands:

```
history             # Show history
shopt -s histverify # Don't execute expanded
result immediately
```

Expansions:

```
!$         # Expand last parameter of most
recent command
!*         # Expand all parameters of most
recent command
!-n        # Expand nth most recent command
```

Operations:

```
!!                 # Execute last command
again
!!:s/<FROM>/<TO>/  # Replace first occurrence
in most recent command
!!:gs/<FROM>/<TO>/ # Replace all occurrences
in most recent command
```

Miscellaneous

Numeric calculations:

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

Redirection:

```
python hello.py > output.txt            #
stdout to (file)
python hello.py 2> error.log            #
stderr to (file)
python hello.py 2>&1                    #
stderr to stdout
```

Inspecting commands:

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

Case/switch statment:

```
case "$1" in
  start | up)
    vagrant up
    ;;
esac
```

Go to the previous directory:

```
cd -
```

Special Variables

`$?`	Exit status of last task
`$!`	PID of last background task
`$$`	PID of shell
`$0`	Filename of the shell script
`$_`	Last argument of the previous command