Bash Scripting Essentials Cheatsheet

A concise reference for Bash scripting, covering essential syntax, commands, and best practices. Includes variables, loops, conditionals, functions, and more, to help you write effective and maintainable scripts.

Basics & Variables

Script Structure

Every bash script starts with a shebang line, specifying the interpreter.

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

Use comments to explain your code.

```
# This is a comment
```

Best practice: use strict mode to catch errors early.

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

Variables

Defining a variable:

```
name="John"
```

Accessing a variable:

```
echo $name
echo "$name"
echo "${name} has a value"
```

String quotes:

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

Variable scope (local):

```
local myvar="local value"  # Function scope
```

Readonly variables:

```
declare -r readonly_var="cannot be changed"
```

Brace Expansion

Generate combinations of strings.

```
echo {A,B}.js   # A.js B.js
echo {1..5}     # 1 2 3 4 5
echo {{1..3},{7..9}} # 1 2 3 7 8 9
```

Loops & Conditionals

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
```

Looping through a range:

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

Looping 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
```

Conditionals

Basic `if` statement:

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

Using `test` command (alternative):

```
if test -e "file.txt"; then
  echo "File exists"
fi
```

Numeric conditions:

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

File existence check:

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

Case Statements

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

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

Functions & Arrays

Functions

Defining a function:

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

# Alternate 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
```

Passing arguments:

```
myfunc() {
  echo "Argument 1: $1, Argument 2: $2"
}

myfunc "arg1" "arg2"
```

Arrays

Defining arrays:

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

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

Accessing elements:

```
echo "${Fruits[0]}"   # Element #0
echo "${Fruits[-1]}"  # Last element
```

Iterating through elements:

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

Getting array length:

```
echo "${#Fruits[@]}"  # Number of elements
```

Adding elements:

```
Fruits=("${Fruits[@]}" "Watermelon")  # Push
Fruits+=('Watermelon')                # Also Push
```

Parameter Expansion & Redirection

Parameter Expansion

Basics:

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

String manipulation:

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

Default values:

```
echo "${food:-Cake}"  #=> $food or "Cake"
```

Redirection

Stdout to file:

```
python hello.py > output.txt
```

Stdout to file (append):

```
python hello.py >> output.txt
```

Stderr to file:

```
python hello.py 2> error.log
```

Stderr to stdout:

```
python hello.py 2>&1
```

Both stdout and stderr to file:

```
python hello.py > output.txt 2>&1
# or
python hello.py &> output.txt
```

Feed file to stdin:

```
python hello.py < foo.txt
```

Advanced Features

History Expansion

Show history:

```
history
```

Expand last parameter of most recent command:

```
!$
```

Expand all parameters of most recent command:

```
!*
```

Expand nth most recent command:

```
!-n
```

Execute last command again:

```
!!
```

Replace in last command:

```
!!:s/<FROM>/<TO>/
```

Options and Globbing

Disable file overwriting:	`set -o noclobber`
Exit on error:	`set -e`
Fail on pipe errors:	`set -o pipefail`
Unset variables are errors:	`set -u`
Wildcards match dotfiles:	`shopt -s dotglob`
Recursive globbing	`shopt -s globstar`

String Manipulation

Uppercase and Lowercase Transformations:

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

Transforming strings using `tr`:

```
echo "Welcome To Devhints" | tr '[:lower:]' '[:upper:]'
#=> WELCOME TO DEVHINTS
```

Getting the directory of the script:

```
dir=${0%/*}
```