Bash Scripting Cheatsheet

A comprehensive cheat sheet covering Bash scripting essentials, from basic syntax to advanced techniques. Includes variables, functions, conditionals, loops, and more, with practical examples for quick reference.

Basics & Variables

Script Header & Execution

Shebang:

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

Recommended header for portability. Specifies the interpreter for the script.

Execution:

```
bash script.sh
./script.sh  # if executable
```

Make script executable: `chmod +x script.sh`

Exit Status:
`$?` contains the exit status of the last command. `0` indicates success, non-zero indicates failure.

Comments:

```
# Single-line comment
: 'Multi-line
comment'
```

Variables

Declaration:

```
name="John"
age=30
```

Usage:

```
echo $name
echo "$name"
echo "${name}!"
```

Always quote variables to prevent word splitting and globbing.

String Quotes:

```
echo "Hi $name"  # Interpolation
echo 'Hi $name'  # No interpolation
```

Environment Variables:

`echo $PATH`

Access system-defined variables.

Unsetting Variables:

```
unset name
```

Command Substitution

Capture output of a command:

```
date=$(date +%Y-%m-%d)
echo "Today is $date"
```

Backticks (obsolescent):

```
echo "I'm in `pwd`"
```

Functions & Conditionals

Functions

Definition:	`get_name() { echo "John" }`
	
	`# Alternate syntax`
	`function my_func { ... }`

Calling:	`echo "You are $(get_name)"`

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:	`myfunc() { echo "Hello, $1" }`
	
	`myfunc "World"`

Special Parameters:	`$#`: Number of arguments `$*`: All arguments as a single word `$@`: All arguments as separate strings `$1`: First argument

Conditionals

Basic Syntax:	`if [[ condition ]]; then # statements elif [[ condition ]]; then # more statements else # final statements fi`
String Conditions:	`[[ -z STRING ]]`: Empty string `[[ -n STRING ]]`: Not empty string `[[ STRING == STRING ]]`: Equal `[[ STRING != STRING ]]`: Not Equal
Numeric Conditions:	`[[ 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
File Conditions:	`[[ -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
Combining Conditions:	`[[ ! EXPR ]]`: Not `[[ X && Y ]]`: And `[[ X || Y ]]`: Or
Example:	`if [[ -e "file.txt" ]]; then echo "file exists" fi`

Loops & Arrays

Loops

Basic For Loop:	`for i in /etc/rc.*; do`
	`  echo "$i"`
	`done`

C-like For Loop:	`for ((i = 0 ; i < 100 ; i++)); do`
	`  echo "$i"`
	`done`

Ranges:	`for i in {1..5}; do`
	`  echo "Welcome $i"`
	`done`
	
	`# With step size`
	`for i in {5..50..5}; do`
	`  echo "Welcome $i"`
	`done`

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

Forever Loop:	`while true; do`
	`  # statements`
	`done`

Arrays

Defining Arrays:	`Fruits=('Apple' 'Banana' 'Orange')`
	
	`# Or`
	`Fruits[0]="Apple"`
	`Fruits[1]="Banana"`
	`Fruits[2]="Orange"`

Accessing Elements:	`echo "${Fruits[0]}"`: Element #0 `echo "${Fruits[-1]}"`: Last element `echo "${Fruits[@]}"`: All elements, space-separated
Array Length:	`echo "${#Fruits[@]}"`: Number of elements `echo "${#Fruits}"`: String length of the 1st element
Range/Slice:	`echo "${Fruits[@]:1:2}"`: Range (from position 1, length 2)
Keys:	`echo "${!Fruits[@]}"`: Keys of all elements, space-separated

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`

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

Dictionaries & Options

Dictionaries (Associative Arrays)

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

Accessing Values:	`echo "${sounds[dog]}"`: Dog's sound `echo "${sounds[@]}"`: All values
Accessing Keys:	`echo "${!sounds[@]}"`: All keys
Number of Elements:	`echo "${#sounds[@]}"`: Number of elements
Deleting Elements:	`unset sounds[dog]`

Iteration (Values):	`for val in "${sounds[@]}";`
	`do`
	`  echo "$val"`
	`done`

Iteration (Keys):	`for key in "${!sounds[@]}";`
	`do`
	`  echo "$key"`
	`done`

Options (set)

`set -o noclobber`	Avoid overlaying files (e.g., `echo "hi" > foo` will fail if foo exists).
`set -o errexit`	Exit immediately if a command exits with a non-zero status (helps prevent cascading errors).
`set -o pipefail`	If a command in a pipeline fails, the pipeline's exit status is that of the failed command.
`set -o nounset`	Attempting to use an unset variable results in an error, exposing potential bugs.

Glob Options (shopt)

`shopt -s nullglob`	If a glob pattern doesn't match any files, it's removed from the argument list (e.g., `'*.foo'` becomes '').
`shopt -s failglob`	If a glob pattern doesn't match, an error is thrown.
`shopt -s nocaseglob`	Glob patterns are case-insensitive.
`shopt -s dotglob`	Wildcards match dotfiles (e.g., `"*.sh"` matches `.foo.sh`).
`shopt -s globstar`	Allows `**` for recursive matches (e.g., `lib/**/*.rb` matches `lib/a/b/c.rb`).

Redirection and miscellaneous

Redirection

- `command > file`: Redirect stdout to file
- `command >> file`: Redirect and append stdout to file
- `command 2> file`: Redirect stderr to file
- `command 2>&1`: Redirect stderr to stdout
- `command > file 2>&1`: Redirect both stdout and stderr to file
- `command &> file`: Redirect both stdout and stderr to file (shorthand)
- `command < file`: Read stdin from file
- `command <<< "string"`: Here string

Heredoc

```
cat <<EOF
This is a heredoc.
It allows multi-line input.
EOF
```

```
cat <<-EOF
  This is a heredoc.
  It allows multi-line input with leading
tabs.
EOF
```

History expansions

`!!`	Execute last command again
`!$`	Expand last parameter of most recent command
`!*`	Expand all parameters of most recent command
`!-n`	Expand nth most recent command
`!n`	Expand nth command in history
`! <command>`	Expand most recent invocation of command