

Proudly sponsored by

ethical ad by CodeFund

Example	Variables	String quotes
<pre>#!/usr/bin/env bash NAME="John" echo "Hello \$NAME!"</pre>	<pre>NAME="John" echo \$NAME echo "\$NAME" echo "\${NAME}!"</pre>	<pre>NAME="John" echo "Hi \$NAME" #=> Hi John echo 'Hi \$NAME' #=> Hi \$NAME</pre>
Conditional execution	Functions	Shell execution
<pre>git commit && git push git commit echo "Commit failed"</pre>	<pre>get_name() { echo "John" } echo "You are \$(get_name)"</pre>	<pre>set -x git commit && git push git commit echo "Commit failed"</pre>
Conditionals	Brace expansion	Strict mode
<pre>if [[-z "\$string"]]; then echo "String is empty" elif [[-n "\$string"]]; then echo "String is not empty" fi</pre>	<pre>echo {A,B}.js {A,B} Same as A B {A,B}.js Same as A.js B.js {1..5} Same as 1 2 3 4 5</pre>	<pre>set -euo pipefail IFS=\$'\n\t'</pre>
See: Conditionals	See: Brace expansion	See: Strict mode

Parameter expansions

Basics	Substitution	Comments
<pre>name="John" echo \${name} echo \${name/J/} ##=> "john" (substitution) echo \${name:0:2} ##=> "Jo" (slicing) echo \${name:2} ##=> "hn" (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"</pre>	<pre>\$(FOO%suffix) \$(FOO#prefix) \$(FOO%%suffix) \$(FOO##prefix) \$(FOO/from/to) \$(FOO//from/to) \$(FOO/%from/to) \$(FOO/#from/to)</pre>	<pre># Single line comment : ' This is a multi-line comment ' # Represents a macro</pre>
<pre>length=2 echo \${name:0:length} ##=> "Jo"</pre>		<p>Substrings Replace all</p> <pre>\$(FOO:0:3) \$(FOO:-3:3)</pre> <p>Substring (position, length)</p> <p>Substring from the right</p>
See: Parameter expansion		
<pre>STR="/path/to/foo.cpp" echo \${STR%.cpp} # /path/to/foo echo \${STR%.cpp}.o # /path/to/foo.o 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</pre>	<p>Length</p> <pre>\$(#FOO)</pre>	<p>Default values</p> <pre>\$(FOO:-val) \$(FOO:=val) \$(FOO:+val) \$(FOO:?message) The : is optional (eg, \${FOO=word}) works</pre> <p>\$FOO, or val if not set</p> <p>Set \$FOO to val if not set</p> <p>val if \$FOO is set</p> <p>Show error message and exit if \$FOO is not set</p>
<pre>STR="Hello world" echo \${STR:6:5} # "world" echo \${STR:-5:5} # "world"</pre>		
<pre>SRC="/path/to/foo.cpp" BASE=\$(SRC##*/) ##=> "foo.cpp" (basepath) DIR=\${SRC%\$BASE} ##=> "/path/to/" (dirpath)</pre>		

Loops

Basic for loop	C-like for loop	Ranges
<pre>for i in /etc/rc.*; do</pre>	<pre>for ((i = 0 ; i < 100 ; i++)); do</pre>	<pre>for i in {1..5}; do</pre>

Read lines done	Forever done	Until done
< file.txt while read line; do echo \$line done	while true; do ... done	until condition; do ... done

Functions

Defining functions	Returning values	Raising errors
myfunc() { echo "hello \$1" } # Same as above (alternate syntax) function myfunc() { echo "hello \$1" } myfunc "John"	myfunc() { local myresult='some value' echo \$myresult } result=\$(myfunc)	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		
See Special parameters.		

Conditionals

Conditions	File conditions	Example
Note that [[is actually a command/program that returns either true or false. It is a program that obeys the same logic (like all base utils, such as grep) and can be used as condition, see examples.	[[-e FILE]]	if ping -c 1 google.com; then echo "It appears you have a working internet connection" fi
	[[-r FILE]]	
	[[-h FILE]]	
	[[-d FILE]]	if grep -q 'foo' ~/.bash_history; then echo "You appear to have typed 'foo' in the past" fi
	[[-w FILE]]	
	[[-s FILE]]	# String if [[-z "\$string"]]; then echo "String is empty" elif [[-n "\$string"]]; then echo "String is not empty" fi
	[[-f FILE]]	
	[[-x FILE]]	
	[[FILE1 -nt FILE2]]	# Combinations if [[X]] && [[Y]]; then ... fi
	[[FILE1 -ot FILE2]]	
	[[FILE1 -ef FILE2]]	
	Greater than	# Equal if [["SA" == "SB"]]
	Greater than or equal	# Regexp if [["A" =~ "."]]
	Regexp	
	Numeric conditions	if ((\$a < \$b)); then echo "\$a is smaller than \$b" fi
	if OPTIONNAME is enabled	if [[-e "file.txt"]]; then echo "file exists" fi
	Not	
	And	
	Or	

Arrays

Defining arrays	Working with arrays
Fruits=('Apple' 'Banana' 'Orange') Fruits[0]="Apple" Fruits[1]="Banana" Fruits[2]="Orange"	echo \${Fruits[0]} echo \${Fruits[@]} echo \${#Fruits[@]} echo \${#Fruits} echo \${#Fruits[3]} echo \${Fruits[@]:3:2} # Element #0 # All elements, space-separated # Number of elements # String length of the 1st element # String length of the Nth element # Range (from position 3, length 2)

```
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
```

```
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"
```

Declares sound as a Dictionary object (aka associative array).

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
```

Options

Options

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

Glob options

```
set -o nullglob # Non-matching globs are removed ('*.foo' => '')
set -o failglob # Non-matching globs throw errors
set -o nocaseglob # Case insensitive globs
set -o globdots # Wildcards match dotfiles ("*.sh" => ".foo.sh")
set -o globstar # Allow ** for recursive matches ('lib/**/*.rb' => 'lib/
```

Set GLOBIGNORE as a colon-separated list of patterns to be removed from glob matches.

History

Commands

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

Operations

!!	Execute last command again
!!:s/<FROM>/<TO>/	Replace first occurrence of <FROM> to <TO> in most recent command
!!:gs/<FROM>/<TO>/	Replace all occurrences of <FROM> to <TO> in most recent command
!\$:t	Expand only basename from last parameter of most recent command
!\$:h	Expand only directory from last parameter of most recent command
!! and !\$	can be replaced with any valid expansion.

Expansions

!\$	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 <command>

Slices

!!:n	Expand only nth token from most recent command (command is 0; first argument is 1)
!^	Expand first argument from most recent command
!\$	Expand last token from most recent command
!!:n-m	Expand range of tokens from most recent command
!!:n-\$	Expand nth token to last from most recent command
!!	can be replaced with any valid expansion i.e. !cat, !-2, !42, etc.

Miscellaneous

Numeric calculations

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

Inspecting commands

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

Subshells

```
(cd somedir; echo "I'm now in $PWD")
pwd # still in first directory
```

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
```

Displaying file directory

```
pwd # /home/user/foo
cd bar/
pwd # /home/user/foo/bar
cd -
pwd # /home/user/foo

echo "ERROR: ${BASH_SOURCE[1]} at about ${BASH_LINENO[0]}"
}

set -o erretrace
trap traperr ERR
```

Python variables

```
python hello.py 2>/dev/null # stderr to (null)
python hello.py &>/dev/null # stdout and stderr to (null)
```

\$?	Exit status of last task
\$!	PID of last background task
\$\$	PID of shell


See Special parameters.

```
-f | --flag )
  flag=1
;;
esac; shift; done
if [[ "$1" == '-' ]] ; then shift; fi
```

Also see

Bash-hackers wiki (bash-hackers.org)
Shell vars (bash-hackers.org)
Learn bash in y minutes (learnxinyminutes.com)
Bash Guide (mywiki.woledge.org)
ShellCheck (shellcheck.net)

devhints.io / Search 381+ cheatsheets



Over 381 curated cheatsheets,
by developers for developers.

Devhints home

Other CLI cheatsheets

Cron cheatsheet	Homebrew cheatsheet
httpie cheatsheet	adb (Android Debug Bridge) cheatsheet
composer cheatsheet	Fish shell cheatsheet

Top cheatsheets

Elixir cheatsheet	ES2015+ cheatsheet
React.js cheatsheet	Vimdiff cheatsheet
Vim cheatsheet	Vim scripting cheatsheet