

Table of contents:

- [Getting started](#)
 - [Introduction](#)
 - [Example](#)
 - [Variables](#)
 - [String quotes](#)
 - [Shell execution](#)
 - [Conditional execution](#)
 - [Functions](#)
 - [Conditionals](#)
 - [Strict mode](#)
 - [Brace expansion](#)
- [Parameter expansions](#)
 - [Basics](#)
 - [Substitution](#)
 - [Comments](#)
 - [Substrings](#)
 - [Length](#)
 - [Manipulation](#)
 - [Default values](#)
- [Loops](#)
 - [Basic for loop](#)
 - [C-like for loop](#)
 - [Ranges](#)
 - [With step size](#)
 - [Reading lines](#)
 - [Forever](#)
- [Functions](#)
 - [Defining functions](#)
 - [Returning values](#)
 - [Raising errors](#)
 - [Arguments](#)
- [Conditionals](#)
 - [Conditions](#)
 - [More conditions](#)
 - [File conditions](#)
 - [Example](#)
- [Arrays](#)
 - [Defining arrays](#)
 - [Working with arrays](#)
 - [Operations](#)
 - [Iteration](#)
- [Dictionaries](#)
 - [Defining](#)
 - [Working with dictionaries](#)

- Iteration
 - Iterate over values
 - Iterate over keys
- Options
 - Options
 - Glob options
- History
 - Commands
 - Expansions
 - Operations
 - Slices
- Miscellaneous
 - Numeric calculations
 - Subshells
 - Redirection
 - Inspecting commands
 - Trap errors
 - Case/switch
 - Source relative
 - printf
 - Transform strings
 - Example
 - Directory of script
 - Getting options
 - Heredoc
 - Reading input
 - Special variables
 - Go to previous directory
 - Check for command's result
 - Grep check
- [Download Pdf File](#)

Getting started

Introduction

This is a quick reference to getting started with Bash scripting.

- [Learn bash in y minutes](#) (*learnxinyminutes.com*)
- [Bash Guide](#) (*mywiki.woledge.org*)

Example

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

```
NAME="John"
echo "Hello $NAME!"
```

Variables

```
NAME="John"
echo $NAME
echo "$NAME"
echo "${NAME}!"
```

String quotes

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

Shell execution

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

See [Command substitution](#)

Conditional execution

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

Functions

```
get_name() {
  echo "John"
}

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

See: [Functions](#)

Conditionals

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

See: [Conditionals](#)

Strict mode

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

See: [Unofficial bash strict mode](#)

Brace expansion

```
echo {A,B}.js
```

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

See: [Brace expansion](#)

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"
```

See: [Parameter expansion](#)

```
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

Code	Description
<code>\${F00%suffix}</code>	Remove suffix
<code>\${F00#prefix}</code>	Remove prefix
---	---
<code>\${F00%%suffix}</code>	Remove long suffix
<code>\${F00##prefix}</code>	Remove long prefix
---	---
<code>\${F00/from/to}</code>	Replace first match
<code>\${F00//from/to}</code>	Replace all
---	---

Code	Description
<code>\${FOO/%from/to}</code>	Replace suffix
<code>\${FOO/#from/to}</code>	Replace prefix

Comments

```
# Single line comment
```

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

Substrings

Expression	Description
<code>\${FOO:0:3}</code>	Substring (<i>position, length</i>)
<code>\${FOO:(-3):3}</code>	Substring from the right

Length

Expression	Description
<code>\${#FOO}</code>	Length of <code>\$FOO</code>

Manipulation

```
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)
```

Default values

Expression	Description
<code>\${FOO:-val}</code>	<code>\$FOO</code> , or <code>val</code> if unset (or null)

Expression	Description
<code>\${F00:=val}</code>	Set <code>\$F00</code> to <code>val</code> if unset (or null)
<code>\${F00:+val}</code>	<code>val</code> if <code>\$F00</code> is set (and not null)
<code>\${F00:?message}</code>	Show error message and exit if <code>\$F00</code> is unset (or null)

Omitting the `:` removes the (non)nullity checks, e.g. `${F00-val}` expands to `val` if unset otherwise `$F00`.

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

```
cat file.txt | while read line; do
  echo $line
done
```

Forever

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

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

Expression	Description
<code>\$#</code>	Number of arguments
<code>\$*</code>	All positional arguments (as a single word)
<code>\$@</code>	All positional arguments (as separate strings)
<code>\$1</code>	First argument
<code>\$_</code>	Last argument of the previous command

Note: `$@` and `$*` must be quoted in order to perform as described. Otherwise, they do exactly the same thing (arguments as separate strings).

See [Special parameters](#).

Conditionals

Conditions

Note that `[[` is actually a command/program that returns either `0` (true) or `1` (false). Any program that obeys the same logic (like all base utils, such as `grep(1)` or `ping(1)`) can be used as condition, see examples.

Condition	Description
<code>[[-z STRING]]</code>	Empty string
<code>[[-n STRING]]</code>	Not empty string
<code>[[STRING == STRING]]</code>	Equal
<code>[[STRING != STRING]]</code>	Not Equal
---	---
<code>[[NUM -eq NUM]]</code>	Equal
<code>[[NUM -ne NUM]]</code>	Not equal
<code>[[NUM -lt NUM]]</code>	Less than
<code>[[NUM -le NUM]]</code>	Less than or equal
<code>[[NUM -gt NUM]]</code>	Greater than

Condition	Description
<code>[[NUM -ge NUM]]</code>	Greater than or equal
---	---
<code>[[STRING =~ STRING]]</code>	Regex
---	---
<code>((NUM < NUM))</code>	Numeric conditions

More conditions

Condition	Description
<code>[[-o noclobber]]</code>	If OPTIONNAME is enabled
---	---
<code>[[! EXPR]]</code>	Not
<code>[[X && Y]]</code>	And
<code>[[X Y]]</code>	Or

File conditions

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

Example

```
# String
if [[ -z "$string" ]]; then
    echo "String is empty"
elif [[ -n "$string" ]]; then
    echo "String is not empty"
else
    echo "This never happens"
fi
```

```
# Combinations
if [[ X && Y ]]; then
    ...
fi
```

```
# Equal
if [[ "$A" == "$B" ]]
```

```
# Regex
if [[ "A" =~ . ]]
```

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

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

Arrays

Defining arrays

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

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

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

```

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

```

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"

```

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

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

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

History

Commands

Command	Description
<code>history</code>	Show history
<code>shopt -s histverify</code>	Don't execute expanded result immediately

Expansions

Expression	Description
<code>!\$</code>	Expand last parameter of most recent command
<code>!*</code>	Expand all parameters of most recent command
<code>!-n</code>	Expand <code>n</code> th most recent command
<code>!n</code>	Expand <code>n</code> th command in history
<code>!<code>command</code></code>	Expand most recent invocation of command <code><command></code>

Operations

Code	Description
<code>!!</code>	Execute last command again
<code>!!:s/<FROM>/<TO>/</code>	Replace first occurrence of <code><FROM></code> to <code><TO></code> in most recent command
<code>!!:gs/<FROM>/<TO>/</code>	Replace all occurrences of <code><FROM></code> to <code><TO></code> in most recent command
<code>!\$:t</code>	Expand only basename from last parameter of most recent command
<code>!\$:h</code>	Expand only directory from last parameter of most recent command

`!!` and `!$` can be replaced with any valid expansion.

Slices

Code	Description
<code>!!:n</code>	Expand only <code>n</code> th token from most recent command (command is <code>0</code> ; first argument is <code>1</code>)
<code>!^</code>	Expand first argument from most recent command
<code>!\$</code>	Expand last token from most recent command
<code>!!:n-m</code>	Expand range of tokens from most recent command
<code>!!:n-\$</code>	Expand <code>n</code> th 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..199
```

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  
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  
diff <(ls -r) <(ls)              # Compare two stdout without files
```

Inspecting commands

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

Trap errors

```
trap 'echo Error at about $LINENO' ERR
```

or

```
traperr() {
  echo "ERROR: ${BASH_SOURCE[1]} at about ${BASH_LINENO[0]}"
}

set -o erretrace
trap traperr ERR
```

Case/switch

```
case "$1" in
  start | up)
    vagrant up
    ;;
  *)
    echo "Usage: $0 {start|stop|ssh}"
    ;;
esac
```

Source relative

```
source "${0%/*}/../share/foo.sh"
```

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"
```

Transform strings

Command option	Description
-c	Operations apply to characters not in the given set
-d	Delete characters
-s	Replaces repeated characters with single occurrence
-t	Truncates

Command option	Description
<code>[:upper:]</code>	All upper case letters
<code>[:lower:]</code>	All lower case letters
<code>[:digit:]</code>	All digits
<code>[:space:]</code>	All whitespace
<code>[:alpha:]</code>	All letters
<code>[:alnum:]</code>	All letters and digits

Example

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

Directory of script

```
DIR="${0%/*}"
```

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
```

Heredoc

```
cat <<END  
hello world  
END
```

Reading input

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

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

Special variables

Expression	Description
\$?	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

See [Special parameters](#).

Go to previous directory

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

Check for command's result

```
if ping -c 1 google.com; then  
    echo "It appears you have a working internet connection"  
fi
```

Grep check

```
if grep -q 'foo' ~/.bash_history; then  
    echo "You appear to have typed 'foo' in the past"  
fi
```

[Download Pdf File](#)