DEVHINTS.IO

Edit

Bash scripting cheatsheet

Proudly sponsored by

MongoDB Atlas is the most reliable cloud database service available. Try now!

ethical ad by CodeFund

Example

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

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

Conditional execution

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

Conditionals

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

See: Conditionals

Variables

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

Functions

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

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

Brace expansion

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

`{A,B}`
`{A,B}.js`
`{1..5}`

See: Brace expansion

St

Sh

St

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 rig
echo ${name:(-2):1} #=> "h" (slicing from rig
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##*.}     # 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

`${FOO%suffix}`

`${FOO#prefix}`

`${FOO%%suffix}`

`${FOO##prefix}`

`${FOO/from/to}`

`${FOO//from/to}`

`${FOO/%from/to}`

`${FOO/#from/to}`

Length

`${#FOO}`

Default values

`${FOO:-val}`

`${FOO:=val}`

`${FOO:+val}`

`${FOO:?message}`

The `:` is optional (eg, `${FOO=word}` works)

C

Su

M

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
```

Ra

Reading lines

```
< 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)"
```

Ra

Arguments

`$#`

`$*`

`$@`

`$1`

See Special parameters.

Conditionals

Conditions

Note that `[[` is actually a command/program that retur
the same logic (like all base utils, such as `grep(1)` or `pi`

`[[ -z STRING ]]`	
`[[ -n STRING ]]`	
`[[ STRING == STRING ]]`	
`[[ STRING != STRING ]]`	
`[[ NUM -eq NUM ]]`	
`[[ NUM -ne NUM ]]`	
`[[ NUM -lt NUM ]]`	
`[[ NUM -le NUM ]]`	
`[[ NUM -gt NUM ]]`	
`[[ NUM -ge NUM ]]`	Greater than or equal
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions
`[[ -o noclobber ]]`	If OPTIONNAME is enabled
`[[ ! EXPR ]]`	Not
`[[ X ]] && [[ Y ]]`	And
`[[ X ]] || [[ Y ]]`	Or

File conditions

`[[ -e FILE ]]`
`[[ -r FILE ]]`
`[[ -h FILE ]]`
`[[ -d FILE ]]`
`[[ -w FILE ]]`
`[[ -s FILE ]]`
`[[ -f FILE ]]`
`[[ -x FILE ]]`
`[[ FILE1 -nt FILE2 ]]`
`[[ FILE1 -ot FILE2 ]]`
`[[ FILE1 -ef FILE2 ]]`

Ex

Arrays

Defining arrays

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

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

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
```

Working with arrays

```
echo ${Fruits[0]}
echo ${Fruits[@]}
echo ${#Fruits[@]}
echo ${#Fruits}
echo ${#Fruits[3]}
echo ${Fruits[@]:3:2}
```

Iteration

```
for i in "${arrayName[@
  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
```

It

Options

Options

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

Glob options

```
set -o nullglob    # No
set -o failglob    # No
set -o nocaseglob  # Ca
set -o globdots    # Wi
set -o globstar    # Al
```

Set `GLOBIGNORE` as a colon-s

History

Commands

`history`	
`shopt -s histverify`	Don't execute exp

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
`!$:t`	Expand only basename from last parameter
`!$:h`	Expand only directory from last parameter

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

Expansions

`!$`
`!*`
`!-n`

Slices

`!!:n`	Expa
`!^`	
`!$`	
`!!:n-m`	
`!!:n-$`	

!! can be replaced with any

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"
```

Trap errors

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

or

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

set -o errtrace
trap traperr ERR
```

Source relative

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

Directory of script

Subshells

```
(cd somedir; echo "I'm
pwd # still in first di
```

Redirection

```
python hello.py > outpu
                     tp
                     ro

python hello.py 2>/dev/
python hello.py &>/dev/

                     .t
```

Case/switch

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

  *)
    echo "Usage: $0 {st
    ;;
esac
```

printf

```
printf "Hello %s, I'm %
#=> "Hello Sven, I'm Ol
```

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

Heredoc

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

Reading input

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

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

Go to previous directory

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

Getting options

```
while [[ "$1" =~ ^- &&
```

```
  -s | --string )
    shift; string=$1
    ;;
```

Special variables

```
$?
```

Also see

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

15 Comments for this cheatsheet. Write yours!

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