DEVHINTS.IO

Edit

Bash scripting cheatsheet

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"
echo 'Hi $NAME'
```

Shell execution

Conditional execution

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

Functions

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

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

Conditionals

```
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
```

Strict mode

```
  echo "String is not empty"
fi
```

See: Conditionals

Brace expansion

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

```
{A,B}
```

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

```
{1..5}
```

See: Brace expansion

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

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

Substitution

```
${FOO%suffix}
```

```
${FOO#prefix}
```

```
${FOO%%suffix}
```

```
${FOO##prefix}
```

```
${FOO/from/to}
```

```
${FOO//from/to}
```

```
${FOO/%from/to}
```

```
${#FOO}
```

Comments

```
# Single line co
```

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

Substrings

```
${FOO:0:3}
```

Length

```
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
```

Manipulation

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

STR="hello world!"
echo ${STR^}   #=> "Hello world!" (uppercase 1st
echo ${STR^^}  #=> "HELLO WORLD!" (all uppercase)
```

Default values

```
${FOO:-val}
${FOO:=val}
${FOO:+val}
${FOO:?message}
```

The : is optional (eg

```
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)
```

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};
    echo "Welcom
done
```

With step size

Reading lines

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

Forever

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

```
for i in {5..50.
    echo "Welcom
```

♯ 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

```
$#
```

```
$*
```

```
$@
```

```
$1
```

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 e

```
[[ -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 ]]
[[ STRING =~ STRING ]]
(( NUM < NUM ))
[[ -o noclobber ]]
```

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 ]]
```

Example

```
# String
if [[ -z "$strin
  echo "String i
elif [[ -n "$str
  echo "String i
fi

# Combinations
if [[ X ]] && [[
  ...
fi

# Equal
if [[ "$A" == "$

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

if (( $a < $b ))
   echo "$a is s
fi

if [[ -e "file.t
  echo "file exi
fi
```

Gre

N

If OPTION

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[@]}           # All elements
echo ${#Fruits[@]}          # Number of el
echo ${#Fruits}             # String lengt
echo ${#Fruits[3]}          # String lengt
echo ${Fruits[@]:3:2}       # Range (from
```

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
```

Working with dictionaries

```
echo ${sounds[dog]} # Dog's sound
echo ${sounds[@]}   # All values
```

Iteration

Iterate over values

```
[[ ! EXPR ]]
```

Not

```
[[ X ]] && [[ Y ]]
```

And

```
[[ X ]] || [[ Y ]]
```

Or

```
sounds[dog]="bark"
sounds[cow]="moo"
sounds[bird]="tweet"
sounds[wolf]="howl"
```

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

```
echo ${!sounds[@]}  # All keys
echo ${#sounds[@]}  # Number of elements
unset sounds[dog]   # Delete dog
```

```
for val in "${so
  echo $val
done
```

Iterate over keys

```
for key in "${!s
  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 ar
set -o failglob    # Non-matching globs th
set -o nocaseglob  # Case insensitive glob
set -o globdots    # Wildcards match dotfi
set -o globstar    # Allow ** for recursiv
```

Set GLOBIGNORE as a colon-separated list of pattern

History

Commands

```
history
```

Expansions

```
!$
```

```
shopt -s histverify
```

Operations

`!!`	Execute last command again
`!!:s/<FROM>/<TO>/`	Replace first occurrence of `<FROM>` to `<TO>` in most recent command
`!!:gs/<FROM>/<TO>/`	Replace all occurre
`!$:t`	Expand only basena
`!$:h`	Expand only direc

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

Slices

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

`!!` can be replaced with any valid expansion i.e. `!cat`

Miscellaneous

Numeric calculations

```
$((a + 200))      # Add 200 to $a

$((RANDOM%=200))  # Random number 0..200
```

Inspecting commands

Subshells

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

Redirection

```
python hello.py > output.txt   # stdout to
python hello.py >> output.txt  # stdout to
```

```
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

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

Getting options

```
while [[ "$1" =~ ^- && ! "$1" == "--" ]]; do case $1 in
  -V | --version )
```

Case/switch

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

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

printf

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

```
printf "This is how you print a float: %f"
#=> "This is how you print a float: 2.000(
```

Heredoc

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

Special variables

`$?`	Exit status of last task
`$!`	PID of last background task
`$$`	
`$0`	

See Special parameters.

Check for command's result

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

```
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
```

Grep check

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

♯ 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)

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