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 pipefail
```

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"
```

Substitution

`${FOO%suffix}`	
`${FOO#prefix}`	
`${FOO%%suffix}`	
`${FOO##prefix}`	
`${FOO/from/to}`	
`${FOO//from/to}`	
`${FOO/%from/to}`	

Comments

```
# Single line com
```

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

Substrings

`${FOO:0:3}`	

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)
```

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

Length

```
${#FOO}
```

```
${FOO:-3:3}
```

Default values

```
${FOO:-val}
```

```
${FOO:=val}
```

```
${FOO:+val}
```

```
${FOO:?message}
```

The `:` is optional (e

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 "Welcome
done
```

With step size

```
for i in {5..50..
```

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)"
```

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 ei
utils, such as `grep(1)` or `ping(1)`) can be used as condition,

```
[[ -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 ]]
```

Grea

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

```
if ping -c 1 goog
  echo "It appear
fi
```

```
if grep -q 'foo'
  echo "You appea
fi
```

```
# String
if [[ -z "$string
  echo "String is
elif [[ -n "$stri
  echo "String is
fi
```

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

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

`(( NUM < NUM ))`	Nu
`[[ -o noclobber ]]`	If OPTIONI
`[[ ! EXPR ]]`	
`[[ X ]] && [[ Y ]]`	
`[[ X ]] || [[ Y ]]`	

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

```
if (( $a < $b ));
   echo "$a is sm
fi
```

```
if [[ -e "file.tx
  echo "file exis
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[@]}           # All elements,
echo ${#Fruits[@]}          # Number of elem
echo ${#Fruits}             # String length
echo ${#Fruits[3]}          # String length
echo ${Fruits[@]:3:2}       # Range (from po
```

Operations

```
Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits+=('Watermelon')                  # Also Push
```

Iteration

```
for i in "${arrayName[@]}"; do
  echo $i
```

```
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
```

```
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 "${sou
  echo $val
done
```

Iterate over keys

```
for key in "${!sc
  echo $key
done
```

Options

Options

Glob 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
```

```
set -o nullglob    # Non-matching globs are
set -o failglob    # Non-matching globs thro
set -o nocaseglob  # Case insensitive globs
set -o globdots    # Wildcards match dotfile
set -o globstar    # Allow ** for recursive
```

Set GLOBIGNORE as a colon-separated list of pattern

History

Commands

`history`

`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 occurrences of <FROM> to <TO> in most recent command
`!$:t`	Expand only basenam
`!$:h`	Expand only director

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

Expansions

`!$`

`!*`

`!-n`

`!n`

Slices

`!!:n`

`!^`

`!$`

```
!!:n-m
```

```
!!:n-$
```

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

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

Subshells

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

Redirection

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

```
                                         tx
```

Case/switch

```
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%/*}"
```

Heredoc

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

Reading input

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

```
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
```

Getting options

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

```
    shift; string=$1
    ;;
  -f | --flag )
```

```
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
```

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)

▶ **10 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

Fish shell

Top cheatsheets

Elixir
cheatsheet

ES2015+
cheatsheet

React.js
cheatsheet

Vimdiff
cheatsheet

Vim
cheatsheet

Vim scripting
cheatsheet

cheatsheet

cheatsheet