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}!"
```

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

Brace expansion

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

{A,B}	Sam

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

See: Brace expansion

Parameter expansions

Basics

```
name="John"
echo ${name}
echo ${name/J/j}    #=> "john" (substit
echo ${name:0:2}    #=> "Jo" (slicing)
echo ${name::2}     #=> "Jo" (slicing)
echo ${name::-1}    #=> "Joh" (slicing)
echo ${name:(-1)}   #=> "n" (slicing fr
echo ${name:(-2):1} #=> "h" (slicing fr
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
```

Substitution

`${FOO%suffix}`	Remov
`${FOO#prefix}`	Remov
`${FOO%%suffix}`	Remove lon
`${FOO##prefix}`	Remove lon
`${FOO/from/to}`	Replace firs
`${FOO//from/to}`	Rep
`${FOO/%from/to}`	Replac
`${FOO/#from/to}`	Replac

Length

`${#FOO}`	Length

```
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=${SRC##*/}   #=> "foo.cpp" (basepa
DIR=${SRC%$BASE}  #=> "/path/to/" (dirp
```

Loops

Basic for loop

```
for i in /etc/rc.*; 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
```

Forever

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

Functions

Defining functions

```
myfunc() {
```

Returning values

```
myfunc() {
```

```
    echo "hello $1"
}

# Same as above (alternate syntax)
function myfunc() {
    echo "hello $1"
}

myfunc "John"
```

```
    local myresult='some value
    echo $myresult
```

Arguments

`$#`	Number of arg
`$*`	All arg
`$@`	All arguments, starting fr
`$1`	First ar

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.

`[[ -z STRING ]]`	Empty string
`[[ -n STRING ]]`	Not empty string

File conditions

`[[ -e FILE ]]`	
`[[ -r FILE ]]`	Re
`[[ -h FILE ]]`	
`[[ -d FILE ]]`	Di
`[[ -w FILE ]]`	W
`[[ -s FILE ]]`	Si

`[[ STRING == STRING ]]`	Equal
`[[ STRING != STRING ]]`	Not Equal
`[[ NUM -eq NUM ]]`	Equal
`[[ NUM -ne NUM ]]`	Not equal
`[[ NUM -lt NUM ]]`	Less than
`[[ NUM -le NUM ]]`	Less than or equal
`[[ NUM -gt NUM ]]`	Greater than
`[[ 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

`[[ -f FILE ]]`	
`[[ -x FILE ]]`	Exe
`[[ FILE1 -nt FILE2 ]]`	1 rece
`[[ FILE1 -ot FILE2 ]]`	2 rece
`[[ FILE1 -ef FILE2 ]]`	Sa

Arrays

Defining arrays

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

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

Work

```
ech
ech
ech
ech
ech
ech
```

Operations

Itera

```
Fruits=("${Fruits[@]}" "Watermelon")
Fruits+=('Watermelon')
Fruits=( ${Fruits[@]/Ap*/} )
unset Fruits[2]
Fruits=("${Fruits[@]}")
Fruits=("${Fruits[@]}" "${Veggies[@]}")
lines=(`cat "logfile"`)
```

```
for
  e
don
```

Options

Options

```
set -o noclobber  # Avoid overlay files
set -o errexit    # Used to exit upon e
set -o pipefail   # Unveils hidden fail
set -o nounset    # Exposes unset varia
```

Glob

```
set
set
set
set
set
```

Set
of p
mat

History

Commands

`history`	Show history
`shopt -s histverify`	Don’t execute expanded

Operations

Expa

```
!$
```

```
!*
```

`!!`	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

Miscellaneous

Numeric calculations

!! and !$ can be replaced with any valid

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

```
$((RANDOM%=200))  # Random number 0..20
```

Inspecting commands

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

Slice

```
!!:
```

```
!^
```

```
!$
```

```
!!:
```

```
!!:
```

```
!! c
exp
```

Subs

```
(cd
pwd
```

Redi

```
pyt
pyt
pyt
pyt
```

Directory

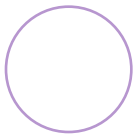
```
pwd # /home/user/foo
cd bar/
pwd # /home/user/foo/bar
cd -
pwd # /home/user/foo
```

```
  echo "ERROR: ${BASH_SOURCE[1]} at abo
}

set -o errtrace
trap traperr ERR
```

```
$?

$!

$$

See

esa
if
```

♯ 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)



Over 380 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