DEVHINTS.IO

Edit

Bash scripting cheatsheet

Introduction

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

Learn bash in y minutes
(learnxinyminutes.com)

Bash Guide
(mywiki.wooledge.org)

Bash Hackers Wiki
(wiki.bash-hackers.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 executio

```
echo "I'm in $
echo "I'm in `
# Same
```

See Command su

Functions

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

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

Conditional execution

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

Conditionals

```
if [[ -z "$str
  echo "String
elif [[ -n "$s
```

Strict mode

Brace expansion

See: Functions

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

`{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

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

Unofficial bash strict mode

```
ho "String
```

Conditionals

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)
```

Substitution

`${FOO%suffix}`	Remove s
`${FOO#prefix}`	Remove p
`${FOO%%suffix}`	Remove long s
`${FOO##prefix}`	Remove long p

Comments

```
# Single line

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

```
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)
```

Default values

`${FOO:-val}`	`$FOO`, or `val` if unset (or null)
`${FOO:=val}`	Set `$FOO` to `val` if unset (or null)
`${FOO:+val}`	`val` if `$FOO` is set (and not null)
`${FOO:?message}`	Show error message and exit if `$FOO` is unset (or null)

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

Substring

```
"HELLO WOR
${STR,}
${STR,,}

"hello wor
${STR^}
${STR^^}
```

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 "Welc
done
```

With step size

```
for i in {5..5
    echo "Welc
```

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"
}
```

Returning values

```
myfunc() {
    local myresult='some value'
    echo $myresult
}
```

```
result="$(myfunc)"
```

Raising errors

```
myfunc() {
  return 1
}
```

```
if myfunc; the
  echo "succes
else
  echo "failur
fi
```

Arguments

```
myfunc "John"
```

`$#`	Number of arguments
`$*`	All positional arguments (as a single word)
`$@`	All positional arguments (as separate strings)
`$1`	First argument
`$_`	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 base utils, such as `grep(1)` or `ping(1)`) can be used as conditio see examples.

`[[ -z STRING ]]`	Empty s

File conditions

`[[ -e FILE ]]`	Exists
`[[ -r FILE ]]`	Readable
`[[ -h FILE ]]`	Symlink
`[[ -d FILE ]]`	Directory

Example

`[[ -n STRING ]]`	Not empty s
`[[ STRING == STRING ]]`	E
`[[ STRING != STRING ]]`	Not E
`[[ NUM -eq NUM ]]`	E
`[[ NUM -ne NUM ]]`	Not e
`[[ NUM -lt NUM ]]`	Less
`[[ NUM -le NUM ]]`	Less than or e
`[[ NUM -gt NUM ]]`	Greater than
`[[ NUM -ge NUM ]]`	Greater than or equal
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions

More conditions

`[[ -o noclobber ]]`	If OPTIONNAME is enabled
`[[ ! EXPR ]]`	Not
`[[ X && Y ]]`	And
`[[ X || Y ]]`	Or

`[[ -w FILE ]]`	Writ
`[[ -s FILE ]]`	Size is > 0 b
`[[ -f FILE ]]`	
`[[ -x FILE ]]`	Execut
`[[ FILE1 -nt FILE2 ]]`	1 is more recent th
`[[ FILE1 -ot FILE2 ]]`	2 is more recent th
`[[ FILE1 -ef FILE2 ]]`	Same

```
# String
if [[ -z "$str
  echo "String
elif [[ -n "$s
  echo "String
else
  echo "This n
fi
```

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

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

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

```
if (( $a < $b
   echo "$a is
fi
```

```
if [[ -e "file
  echo "file e
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 elemer
echo ${Fruits[@]}           # All element
echo ${#Fruits[@]}          # Number of e
echo ${#Fruits}             # String leng
echo ${#Fruits[3]}          # String leng
echo ${Fruits[@]:3:2}       # Range (from
echo ${!Fruits[@]}          # Keys of all
```

Operations

```
Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits+=('Watermelon')                  # Also Push
Fruits=( ${Fruits[@]/Ap*/} )            # Remove by regex
unset Fruits[2]                         # Remove one item
Fruits=("${Fruits[@]}")                 # Duplicate
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate
lines=(`cat "logfile"`)                 # Read from file
```

Iteration

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

Dictionaries

Defining

Working with dictionaries

Iteration

```
declare -A sounds
```

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

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

```
echo ${sounds[dog]} # Dog's sound
echo ${sounds[@]}   # All values
echo ${!sounds[@]}  # All keys
echo ${#sounds[@]}  # Number of elements
unset sounds[dog]   # Delete dog
```

Iterate over values

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

Iterate over keys

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

Options

Options

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

Glob options

```
shopt -s nullglob    # Non-matching globs
shopt -s failglob    # Non-matching globs
shopt -s nocaseglob  # Case insensitive g
shopt -s dotglob     # Wildcards match do
shopt -s globstar    # Allow ** for recur
```

Set `GLOBIGNORE` as a colon-separated list of patte
removed from glob matches.

History

Commands

`history`	Show history
`shopt -s histverify`	Don't execute expanded result immediately

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
`!$:t`	Expand only basename from last parameter of most recent command
`!$:h`	Expand only directory from last parameter of most recent command

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

Expansions

`!$`	Expand last parameter of most
`!*`	Expand all parameters of most
`!-n`	Expand nth most
`!n`	Expand nth co
`!<command>`	Expand most recent invoca

Slices

`!!:n`	Expand only nth token from most (command is `0`; fi
`!^`	Expand first argument from most
`!$`	Expand last token from most
`!!:n-m`	Expand range of tokens from most
`!!:n-$`	Expand nth token to last from most

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

Miscellaneous

Numeric calculations

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

```
$(($RANDOM%200))  # Random number 0..199
```

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
```

Subshells

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

Redirection

```
python hello.py > output.txt   # stdout t
python hello.py >> output.txt  # stdout t
python hello.py 2> error.log   # stderr t
python hello.py 2>&1           # stderr t
python hello.py 2>/dev/null    # stderr t
python hello.py &>/dev/null    # stdout a
```

```
python hello.py < foo.txt      # feed foo
diff <(ls -r) <(ls)            # Compare
```

Case/switch

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

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

Conditionals based on a command's result

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

`$$`	PID of shell
`$0`	Filename of the shell script
`$_`	Last argument of the previous command
`${PIPESTATUS[n]}`	return value of piped commands (array)

See Special parameters.

`[:space:]`	All whitespace
`[:alpha:]`	All letters
`[:alnum:]`	All letters and digits

Example

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

Conditionals based on the existence of files

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

```
pwd # /home/user/foo
```

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

developers for developers.

Devhints home

cheatsheet

Debug Bridge)
cheatsheet

composer
cheatsheet

Fish shell
cheatsheet

cheatsheet

cheatsheet

Vim
cheatsheet

Vim scripting
cheatsheet