Bash scripting cheatsheet

Introduction

This is a quick reference to getting started with Bash scr

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!"
```

String quotes

```
name="John"
echo "Hi $name"  #=> Hi John
echo 'Hi $name'  #=> Hi $name
```

Shell execution

See Command substitution

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

Strict mode

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

See: Unofficial bash strict mode

# 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

```
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)
```

Substitution

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

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
```

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
```

Reading lines

Forever

```
while read -r line; do
  echo "$line"
done <file.txt
```

```
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)
```

Arguments

`$#`

`$*`

`$@`

`$1`

`$_`

Note: `$@` and `$*` must be quoted in order to perform as descri
as separate strings).

See Special parameters.

Conditionals

Conditions

Note that `[[` is actually a command/program that returns either
(like all base utils, such as `grep(1)` or `ping(1)`) can be used as

`[[ -z STRING ]]`

`[[ -n STRING ]]`

File conditions

`[[ -e FILE ]]`

`[[ -r FILE ]]`

`[[ -h FILE ]]`

`[[ -d FILE ]]`

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

`[[ -w FILE ]]`
`[[ -s FILE ]]`
`[[ -f FILE ]]`
`[[ -x FILE ]]`
`[[ FILE1 -nt FILE2 ]]`
`[[ FILE1 -ot FILE2 ]]`
`[[ FILE1 -ef FILE2 ]]`

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
```

Working with arrays

```
echo "${Fruits[0]}"
echo "${Fruits[-1]}"
echo "${Fruits[@]}"
echo "${#Fruits[@]}"
echo "${#Fruits}"
echo "${#Fruits[3]}"
echo "${Fruits[@]:3:2}"
echo "${!Fruits[@]}"
```

Iteration

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

```
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
```

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

```
shopt -s nullglob    # N
shopt -s failglob    # N
shopt -s nocaseglob  # C
shopt -s dotglob     # W
shopt -s globstar    # A
```

Set GLOBIGNORE as a colon-se

History

Commands

`history`	
`shopt -s histverify`	Don’t execute ex

Operations

Expansions

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

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

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

Slices

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

`!!` can be replaced with any v

Miscellaneous

Numeric calculations

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

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

```
declare -i count  # Declare as type integer
count+=1          # Increment
```

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]}"
}
```

Subshells

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

Redirection

```
python hello.py > output
python hello.py >> outpu
python hello.py 2> error
python hello.py 2>&1
python hello.py 2>/dev/r
                        ut.
                        /r
                        o m
```

```
python hello.py < foo.tx
diff <(ls -r) <(ls)
```

Case/switch

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

  *)
```

```
set -o errtrace
trap traperr ERR
```

Source relative

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

Transform strings

-c	Operations apply to characters not in the given set
-d	Delete characters
-s	Replaces repeated characters with single occurrence
-t	Truncates
[:upper:]	All upper case letters
[:lower:]	All lower case letters
[:digit:]	All digits
[:space:]	
[:alpha:]	All letters
[:alnum:]	All letters and digits

Example

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

Heredoc

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

Reading input

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

The -r option disables a peculiar legacy behavior with backslashes.

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

```
    echo "Usage: $0 {sta
    ;;
esac
```

printf

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

```
ou
ri
/ba
li
 2
```

Directory of script

```
dir=${0%/*}
```

Getting options

```
while [[ "$1" =~ ^- && !
  -V | --version )
    echo "$version"
    exit
    ;;
  -s | --string )
    shift; string=$1
    ;;
```

```
if [[ "$1" == '--' ]]; t
```

Special variables

$?
$!
$$
$0

Go to previous directory

```
$_
```

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

Check for command's r

```
if ping -c 1 google.com;
  echo "It appears you h
fi
```

Grep check

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

Over 358 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 ●