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

Conditional execution

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

Functions

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

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

See: Functions

Shell execution

```
echo "I'm in $(pwd)"
echo "I'm in `pwd`"
# Same
```

See Command substitution

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}`	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

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 ${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

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=${STR##*/}   #=> "foo.cpp" (basepath)
DIR=${SRC%$BASE}  #=> "/path/to" (dirpath)
```

Substitution

`${FOO%suffix}`	Remove suffix
`${FOO#prefix}`	Remove prefix
`${FOO%%suffix}`	Remove long suffix
`${FOO##prefix}`	Remove long prefix
`${FOO/from/to}`	Replace first match
`${FOO//from/to}`	Replace all
`${FOO/%from/to}`	Replace suffix
`${FOO/#from/to}`	Replace prefix

Comments

```
# Single line comment

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

Length

`${#FOO}`	Length of $FOO

Substrings

`${FOO:0:3}`	Substring (position, length)
`${FOO:-3:3}`	Substring from the right

Default values

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

The `:` is optional (eg, `${FOO=word}` works)

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
```

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"
}
```

```
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

`$#`	Number of arguments
`$*`	All arguments
`$@`	All arguments, starting from first
`$1`	First argument

See Special parameters.

Conditionals

Conditions

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

File conditions

`[ -e FILE ]`	Exists
`[ -r FILE ]`	Readable
`[ -h FILE ]`	Symlink
`[ -d FILE ]`	Directory
`[ -w FILE ]`	Writable
`[ -s FILE ]`	Size is > 0 bytes
`[ -f FILE ]`	File
`[ -x FILE ]`	Executable
`[ FILE1 -nt FILE2 ]`	1 is more recent than 2
`[ FILE1 -ot FILE2 ]`	2 is more recent than 1
`[ FILE1 -ef FILE2 ]`	Same files

Example

```
# String
if [ -z "$string" ]; then
  echo "String is empty"
elif [ -n "$string" ]; then
  echo "String is not empty"
fi
```

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

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

```
if (( $a < $b ))
```

```
if [ -e "file.txt" ]; then
  echo "file exists"
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, space-separated
echo ${#Fruits[@]}          # Number of elements
echo ${#Fruits}             # String length of the 1st element
echo ${#Fruits[3]}          # String length of the Nth element
echo ${Fruits[@]:3:2}       # Range (from position 3, length 2)
```

Operations

```
Fruits=("${Fruits[@]}" "Watermelon")    # 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
```

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 are removed  ('*.foo' => '')
set -o failglob    # Non-matching globs throw errors
set -o nocaseglob  # Case insensitive globs
set -o globdots    # Wildcards match dotfiles ("*.sh" => ".foo.sh")
set -o globstar    # Allow ** for recursive matches ('lib/**/*.rb' => 'lib/
```

Set `GLOBIGNORE` as a colon-separated list of patterns to be removed from glob matches.

History

Commands

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

Operations

`!!: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 recent command
`!*`	Expand all parameters of most recent command
`!-n`	Expand nth most recent command
`!n`	Expand nth command in history
`!<command>`	Expand most recent invocation of command <command>

Slices

`!!:n`	Expand only nth token from most recent command (command is 0; first param is 1)
`!!:n-m`	Expand range of tokens from most recent command
`!!:n-$`	Expand nth token to last from most recent command

`!!` can be replaced with any valid expansion i.e. `!cat`, `!-2`, `!42`, etc.

Miscellaneous

Numeric calculations

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

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

Subshells

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

Inspecting commands

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

Redirection

```
python hello.py > output.txt   # stdout to (file)
python hello.py >> output.txt  # stdout to (file), append
python hello.py 2> error.log   # stderr to (file)
python hello.py 2>&1           # stderr to stdout
python hello.py 2>/dev/null    # stderr to (null)
python hello.py &>/dev/null    # stdout and stderr to (null)

python hello.py < foo.txt
```

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
```

Case/switch

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

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

Source relative

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

printf

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

Directory of script

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

Getting options

```
while [[ "$1" =~ ^- && ! "$1" == "--" ]]; do case $1 in
  -V | --version )
    echo $version
    exit
    ;;
  -s | --string )
    shift; string=$1
    ;;
  -f | --flag )
    flag=1
    ;;
esac; shift; done
if [[ "$1" == '--' ]]; then shift; fi
```

Heredoc

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

Special variables

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

See Special parameters.

Reading input

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

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

Also see

- Bash-hackers wiki (bash-hackers.org)
- Shell vars (bash-hackers.org)
- Learn bash in y minutes (learnxinyminutes.com)