Bash

This is a quick reference cheat sheet to getting started with linux bash shell scripting.

Getting Started

hello.sh

```
#!/bin/bash

VAR="world"
echo "Hello $VAR!" # => Hello world!
```

Execute the script

```
$ bash hello.sh
```

Variables

```
NAME="John"

echo ${NAME}    # => John (Variables)
echo $NAME      # => John (Variables)
echo "$NAME"    # => John (Variables)
echo '$NAME'    # => $NAME (Exact string)
echo "${NAME}!" # => John! (Variables)

NAME = "John"   # => Error (about space)
```

Comments

```
# This is an inline Bash comment.

: '
This is a
```

```
very neat comment
in bash
'
```

Multi-line comments use `:'` to open and `'` to close

Arguments

`$1 ... $9`	Parameter 1 ... 9
`$0`	Name of the script itself
`$1`	First argument
`${10}`	Positional parameter 10
`$#`	Number of arguments
`$$`	Process id of the shell
`$*`	All arguments
`$@`	All arguments, starting from first
`$-`	Current options
`$_`	Last argument of the previous command

See: Special parameters

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}`	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

Shell execution

```
# => I'm in /path/of/current
echo "I'm in $(PWD)"

# Same as:
echo "I'm in `pwd`"
```

See: Command substitution

Bash Parameter expansions

Syntax	
`${FOO%suffix}`	Remove suffix
`${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

`${FOO/#from/to}`	Replace prefix
Substrings	
`${FOO:0:3}`	Substring (position, length)
`${FOO:(-3):3}`	Substring from the right
Length	
`${#FOO}`	Length of `$FOO`
Default values	
`${FOO:-val}`	`$FOO`, or `val` if unset
`${FOO:=val}`	Set `$FOO` to `val` if unset
`${FOO:+val}`	`val` if `$FOO` is set
`${FOO:?message}`	Show message and exit if `$FOO` is unset

Substitution

```
echo ${food:-Cake}  #=> $food or "Cake"


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
```

Slicing

```
name="John"
echo ${name}           # => John
echo ${name:0:2}       # => Jo
echo ${name::2}        # => Jo
echo ${name::-1}       # => Joh
echo ${name:(-1)}      # => n
echo ${name:(-2)}      # => hn
```

```
echo ${name:(-2):2}    # => hn

length=2
echo ${name:0:length}  # => Jo
```

See: Parameter expansion

basepath & dirpath

```
SRC="/path/to/foo.cpp"

BASEPATH=${SRC##*/}
echo $BASEPATH  # => "foo.cpp"


DIRPATH=${SRC%$BASEPATH}
echo $DIRPATH   # => "/path/to/"
```

Transform

```
STR="HELLO WORLD!"
echo ${STR,}   # => hELLO WORLD!
echo ${STR,,}  # => hello world!

STR="hello world!"
echo ${STR^}   # => Hello world!
echo ${STR^^}  # => HELLO WORLD!

ARR=(hello World)
echo "${ARR[@],}" # => hello world
echo "${ARR[@]^}" # => Hello World
```

Bash Arrays

Defining arrays

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

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

```
Fruits[2]="Orange"

ARRAY1=(foo{1..2}) # => foo1 foo2
ARRAY2=({A..D})    # => A B C D

# Merge => foo1 foo2 A B C D
ARRAY3=(${ARRAY1[@]} ${ARRAY2[@]})

# declare construct
declare -a Numbers=(1 2 3)
Numbers+=(4 5) # Append => 1 2 3 4 5
```

Indexing

`${Fruits[0]}`	First element
`${Fruits[-1]}`	Last element
`${Fruits[*]}`	All elements
`${Fruits[@]}`	All elements
`${#Fruits[@]}`	Number of all
`${#Fruits}`	Length of 1st
`${#Fruits[3]}`	Length of nth
`${Fruits[@]:3:2}`	Range
`${!Fruits[@]}`	Keys of all

Iteration

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

for e in "${Fruits[@]}"; do
    echo $e
done
```

With index

```
for i in "${!Fruits[@]}"; do
  printf "%s\t%s\n" "$i" "${Fruits[$i]}"
done
```

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
Fruits=("${Fruits[@]}" "${Veggies[@]}")  # Concatenate
lines=(`cat "logfile"`)                  # Read from file
```

Arrays as arguments

```
function extract()
{
    local -n myarray=$1
    local idx=$2
    echo "${myarray[$idx]}"
}
Fruits=('Apple' 'Banana' 'Orange')
extract Fruits 2     # => Orangle
```

Bash Dictionaries

Defining

```
declare -A sounds

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

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

```
for val in "${sounds[@]}"; do
    echo $val
done
```

```
for key in "${!sounds[@]}"; do
    echo $key
done
```

Bash Conditionals

Integer conditions	
`[[ 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
`(( NUM < NUM ))`	Less than
`(( NUM <= NUM ))`	Less than or equal
`(( NUM > NUM ))`	Greater than
`(( NUM >= NUM ))`	Greater than or equal

String conditions	
`[[ -z STR ]]`	Empty string
`[[ -n STR ]]`	Not empty string
`[[ STR == STR ]]`	Equal
`[[ STR = STR ]]`	Equal (Same above)

`[[ STR < STR ]]`	Less than (ASCII)
`[[ STR > STR ]]`	Greater than (ASCII)
`[[ STR != STR ]]`	Not Equal
`[[ STR =~ STR ]]`	Regexp

Example

String

```
if [[ -z "$string" ]]; then
    echo "String is empty"
elif [[ -n "$string" ]]; then
    echo "String is not empty"
else
    echo "This never happens"
fi
```

Combinations

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

Equal

```
if [[ "$A" == "$B" ]]; then
    ...
fi
```

Regex

```
if [[ '1. abc' =~ ([a-z]+) ]]; then
    echo ${BASH_REMATCH[1]}
fi
```

Smaller

```
if (( $a < $b )); then
   echo "$a is smaller than $b"
fi
```

Exists

```
if [[ -e "file.txt" ]]; then
    echo "file exists"
```

```
fi
```

File conditions

`[[ -e FILE ]]`	Exists
`[[ -d FILE ]]`	Directory
`[[ -f FILE ]]`	File
`[[ -h FILE ]]`	Symlink
`[[ -s FILE ]]`	Size is > 0 bytes
`[[ -r FILE ]]`	Readable
`[[ -w FILE ]]`	Writable
`[[ -x FILE ]]`	Executable
`[[ f1 -nt f2 ]]`	f1 newer than f2
`[[ f1 -ot f2 ]]`	f2 older than f1
`[[ f1 -ef f2 ]]`	Same files

More conditions

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

logical and, or

```
if [ "$1" = 'y' -a $2 -gt 0 ]; then
    echo "yes"
fi

if [ "$1" = 'n' -o $2 -lt 0 ]; then
    echo "no"
fi
```

Bash 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}; do
    echo "Welcome $i"
done
```

With step size

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

Auto increment

```
i=1
while [[ $i -lt 4 ]]; do
    echo "Number: $i"
    ((i++))
done
```

Auto decrement

```
i=3
while [[ $i -gt 0 ]]; do
    echo "Number: $i"
    ((i--))
done
```

Continue

```
for number in $(seq 1 3); do
    if [[ $number == 2 ]]; then
        continue;
    fi
    echo "$number"
done
```

Break

```
for number in $(seq 1 3); do
    if [[ $number == 2 ]]; then
        # Skip entire rest of loop.
        break;
    fi
    # This will only print 1
    echo "$number"
done
```

Until

```
count=0
until [ $count -gt 10 ]; do
    echo "$count"
    ((count++))
done
```

Forever

```
while true; do
    # here is some code.
done
```

Forever (shorthand)

```
while :; do
    # here is some code.
done
```

Reading lines

```
cat file.txt | while read line; do
    echo $line
done
```

Bash 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
```

Bash Options

Options

```
# Avoid overlay files
# (echo "hi" > foo)
set -o noclobber

# Used to exit upon error
# avoiding cascading errors
set -o errexit

# Unveils hidden failures
set -o pipefail

# Exposes unset variables
set -o nounset
```

Glob options

```
# Non-matching globs are removed
# ('*.foo' => '')
shopt -s nullglob

# Non-matching globs throw errors
shopt -s failglob

# Case insensitive globs
shopt -s nocaseglob

# Wildcards match dotfiles
# ("*.sh" => ".foo.sh")
shopt -s dotglob

# Allow ** for recursive matches
# ('lib/**/*.rb' => 'lib/a/b/c.rb')
shopt -s globstar
```

Bash History

Commands

`history`	Show history
`sudo !!`	Run the previous command with sudo
`shopt -s histverify`	Don't execute expanded result immediately

Expansions

`!$`	Expand last parameter of most recent command
`!*`	Expand all parameters of most recent command
`!-n`	Expand `n`th most recent command
`!n`	Expand `n`th command in history
`!<command>`	Expand most recent invocation of command `<command>`

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.

Slices

`!!:n`	Expand only `n`th token from most recent command (command is `0`; first argument is `1`)
`!^`	Expand first argument from most recent command
`!$`	Expand last token from most recent command
`!!:n-m`	Expand range of tokens from most recent command
`!!:n-$`	Expand `n`th 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..199
```

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      # feed foo.txt to stdin for python
```

Source relative

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

Directory of script

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

Case/switch

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

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

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
```

printf

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

printf "1 + 1 = %d" 2
#=> "1 + 1 = 2"

printf "Print a float: %f" 2
#=> "Print a float: 2.000000"
```

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
```

Check for command's result

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

Special variables

$?	Exit status of last task
$!	PID of last background task
$$	PID of shell
$0	Filename of the shell script

See Special parameters.

Grep check

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

Backslash escapes

	!	"	#
&	'	(	)
,	;	<	>
[	|	\	]
^	{	}	`
$	*	?	

Escape these special characters with \

Heredoc

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

Go to previous directory

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

Reading input

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

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

Conditional execution

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

Strict mode

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

See: Unofficial bash strict mode

Optional arguments

```
args=("$@")
args+=(foo)
args+=(bar)
echo "${args[@]}"
```

Put the arguments into an array and then append

Also see

Devhints (devhints.io)
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)
shell - Standard Shell (devmanual.gentoo.org)

Related Cheatsheet

Awk Cheatsheet
Quick Reference

Hook Cheatsheet
Quick Reference

Powershell Cheatsheet
Quick Reference

Python Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

Perl

The perl quick reference cheat sheet that aims at providing help on writing basic syntax and methods.

Getting Started

Unix and Linux Installation

- Open a Web browser and go to https://www.perl.org/get.html.
- Follow the link to download zipped source code available for Unix/Linux.
- Download perl-5.x.y.tar.gz file and issue the following commands at $ prompt.

```
$tar -xzf perl-5.x.y.tar.gz
$cd perl-5.x.y
$./Configure -de
$make
$make test
$make install
```

Windows Installation

- Follow the link for the Strawberry Perl installation on Windows http://strawberryperl.com
- Download either 32bit or 64bit version of installation.
- Run the downloaded file by double-clicking it in Windows Explorer. This brings up the Perl install wizard, which is really easy to use. Just accept the default settings, wait until the installation is finished, and you're ready to roll!

Macintosh Installation

- Open a Web browser and go to https://www.perl.org/get.html.

- Follow the link to download zipped source code available for Mac OS X.
- Download perl-5.x.y.tar.gz file and issue the following commands at $ prompt

```
$tar -xzf perl-5.x.y.tar.gz
$cd perl-5.x.y
$./Configure -de
$make
$make test
$make install
```

Running Perl

```
# Unix/Linux
$perl  -e <perl code>
# Windows/DOS
C:>perl -e <perl code>
```

Available command line options

-d[:debugger]	Runs program under debugger
-Idirectory	Specifies @INC/#include directory
-T	Enables tainting warnings
-U	Allows unsafe operations
-w	Enables many useful warnings
-W	Enables all warnings
-X	Disables all warnings
-e program	Runs Perl script sent in as program
file	Runs Perl script from a given file

Script from the Command-line

```
# Unix/Linux
$perl  script.pl
# Windows/DOS
C:>perl script.pl
```

First Perl Program

```
$perl -e 'print "Hello World\n"'

# #!/usr/bin/perl

# This will print "Hello, World"
print "Hello, world\n";
$chmod 0755 hello.pl
$./hello.pl
```

Comments in Perl

```
# This is a comment in perl
=begin comment
This is all part of multiline comment.
You can use as many lines as you like
These comments will be ignored by the
compiler until the next =cut is encountered.
=cut
```

Whitespaces in Perl

```
#!/usr/bin/perl

# This would print with a line break in the middle
print "Hello
          world\n";
#output
#Hello
#         world
```

Single and Double Quotes in Perl

```
#!/usr/bin/perl

print "Hello, world\n";
print 'Hello, world\n';

#Hello, world
#Hello, world\n$
```

Datatypes

Creating Variables

```
$age = 25;             # An integer assignment
$name = "John Paul";   # A string
$salary = 1445.50;     # A floating point
```

Scalar Variables

```
#!/usr/bin/perl

$age = 25;             # An integer assignment
$name = "John Paul";   # A string
$salary = 1445.50;     # A floating point

print "Age = $age\n";
print "Name = $name\n";
print "Salary = $salary\n";
```

Array Variables

```
#!/usr/bin/perl

@ages = (25, 30, 40);
@names = ("John Paul", "Lisa", "Kumar");

print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
```

Hash Variables

```
#!/usr/bin/perl

%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);

print "\$data{'John Paul'} = $data{'John Paul'}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
```

```
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

Variable Context

```
#!/usr/bin/perl

@names = ('John Paul', 'Lisa', 'Kumar');

@copy = @names;
$size = @names;

print "Given names are : @copy\n";
print "Number of names are : $size\n";
```

Numeric Scalars

```
#!/usr/bin/perl

$integer = 200;
$negative = -300;
$floating = 200.340;
$bigfloat = -1.2E-23;

# 377 octal, same as 255 decimal
$octal = 0377;

# FF hex, also 255 decimal
$hexa = 0xff;

print "integer = $integer\n";
print "negative = $negative\n";
print "floating = $floating\n";
print "bigfloat = $bigfloat\n";
print "octal = $octal\n";
print "hexa = $hexa\n";
```

String Scalars

```
#!/usr/bin/perl

$var = "This is string scalar!";
$quote = 'I m inside single quote - $var';
$double = "This is inside single quote - $var";

$escape = "This example of escape -\tHello, World!";
```

```
print "var = $var\n";
print "quote = $quote\n";
print "double = $double\n";
print "escape = $escape\n";
```

Scalar Operations

```
#!/usr/bin/perl

$str = "hello" . "world";       # Concatenates strings.
$num = 5 + 10;                  # adds two numbers.
$mul = 4 * 5;                   # multiplies two numbers.
$mix = $str . $num;             # concatenates string and number.

print "str = $str\n";
print "num = $num\n";
print "mul = $mul\n";
print "mix = $mix\n";
```

Multiline Strings

```
#!/usr/bin/perl

$string = 'This is
a multiline
string';

print "$string\n";

############################
print <<EOF;
This is
a multiline
string
EOF
```

V-Strings

```
#!/usr/bin/perl

$smile  = v9786;
$foo    = v102.111.111;
$martin = v77.97.114.116.105.110;
```

```
print "smile = $smile\n";
print "foo = $foo\n";
print "martin = $martin\n";
```

Special Literals

```
#!/usr/bin/perl

print "File name ". __FILE__ . "\n";
print "Line Number " . __LINE__ ."\n";
print "Package " . __PACKAGE__ ."\n";

# they can not be interpolated
print "__FILE__ __LINE__ __PACKAGE__\n";
```

Sequential Number Arrays

```
#!/usr/bin/perl

@var_10 = (1..10);
@var_20 = (10..20);
@var_abc = (a..z);

print "@var_10\n";   # Prints number from 1 to 10
print "@var_20\n";   # Prints number from 10 to 20
print "@var_abc\n";  # Prints number from a to z
```

Array Size

```
#!/usr/bin/perl

@array = (1,2,3);
$array[50] = 4;

$size = @array;
$max_index = $#array;

print "Size:  $size\n";
print "Max Index: $max_index\n";
```

Array oparationns

Adding and Removing Elements in Array

push @ARRAY, LIST	Pushes the values of the list onto the end of the array.
pop @ARRAY	Pops off and returns the last value of the array.
shift @ARRAY	Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down.
unshift @ARRAY, LIST	Prepends list to the front of the array, and returns the number of elements in the new array.

Array oparations

```
#!/usr/bin/perl

# create a simple array

@coins = ("Quarter","Dime","Nickel");
print "1. \@coins = @coins\n";

# add one element at the end of the array

push(@coins, "Penny");
print "2. \@coins = @coins\n";

# add one element at the beginning of the array

unshift(@coins, "Dollar");
print "3. \@coins = @coins\n";

# remove one element from the last of the array.

pop(@coins);
print "4. \@coins = @coins\n";

# remove one element from the beginning of the array.

shift(@coins);
print "5. \@coins = @coins\n";
```

Slicing Array Elements

```
#!/usr/bin/perl

@days = qw/Mon Tue Wed Thu Fri Sat Sun/;

@weekdays = @days[3,4,5];

print "@weekdays\n";
```

Replacing Array Elements

```
#!/usr/bin/perl

@nums = (1..20);
print "Before - @nums\n";

splice(@nums, 5, 5, 21..25);
print "After - @nums\n";
```

Transform Strings to Arrays

```
#!/usr/bin/perl

# define Strings

$var_string = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$var_names = "Larry,David,Roger,Ken,Michael,Tom";

# transform above strings into arrays.

@string = split('-', $var_string);
@names = split(',', $var_names);

print "$string[3]\n";  # This will print Roses
print "$names[4]\n"; # This will print Michael
```

Transform Arrays to Strings

```
#!/usr/bin/perl

# define Strings
$var_string = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$var_names = "Larry,David,Roger,Ken,Michael,Tom";

# transform above strings into arrays.
@string = split('-', $var_string);
```

```
@names  = split(',', $var_names);

$string1 = join( '-', @string );
$string2 = join( ',', @names );

print "$string1\n";
print "$string2\n";
```

Sorting Arrays

```
#!/usr/bin/perl

# define an array
@foods = qw(pizza steak chicken burgers);
print "Before: @foods\n";

# sort this array
@foods = sort(@foods);
print "After: @foods\n";
```

The $[Special Variable

```
#!/usr/bin/perl

# define an array
@foods = qw(pizza steak chicken burgers);
print "Foods: @foods\n";

# Let's reset first index of all the arrays.
$[ = 1;

print "Food at \@foods[1]: $foods[1]\n";
print "Food at \@foods[2]: $foods[2]\n";
```

Merging Arrays

```
#!/usr/bin/perl

@odd = (1,3,5);
@even = (2, 4, 6);

@numbers = (@odd, @even);

print "numbers = @numbers\n";
```

Selecting Elements from Lists

```
#!/usr/bin/perl

@list = (5,4,3,2,1)[1..3];

print "Value of list = @list\n";
```

Accessing Hash Elements

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

print "$data{'John Paul'}\n";
print "$data{'Lisa'}\n";
print "$data{'Kumar'}\n";
```

Extracting Slices

```
#!/uer/bin/perl


%data = (-JohnPaul => 45, -Lisa => 30, -Kumar => 40);

@array = @data{-JohnPaul, -Lisa};

print "Array : @array\n";
```

Extracting Keys and Values

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

@names = keys %data;

print "$names[0]\n";
print "$names[1]\n";
print "$names[2]\n";
```

Getting Hash Size

```
#!/usr/bin/perl
```

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

@keys = keys %data;
$size = @keys;
print "1 - Hash size:  is $size\n";

@values = values %data;
$size = @values;
print "2 - Hash size:  is $size\n";
```

Add and Remove Elements in Hashes

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
@keys = keys %data;
$size = @keys;
print "1 - Hash size:  is $size\n";

# adding an element to the hash;
$data{'Ali'} = 55;
@keys = keys %data;
$size = @keys;
print "2 - Hash size:  is $size\n";

# delete the same element from the hash;
delete $data{'Ali'};
@keys = keys %data;
$size = @keys;
print "3 - Hash size:  is $size\n";
```

Control Flow

if-else

```
#!/usr/bin/perl

# Perl program to illustrate
# Decision-Making statements

$a = 10;
```

```
$b = 15;

# if condition to check
# for even number
if($a % 2 == 0 )
{
    printf "Even Number";
}

# if-else condition to check
# for even number or odd number
if($b % 2 == 0 )
{
    printf "\nEven Number";
}
else
{
    printf "\nOdd Number";
}
```

The ? : Operator

```
#!/usr/local/bin/perl

$name = "Ali";
$age = 10;

$status = ($age > 60 )? "A senior citizen" : "Not a senior citizen";

print "$name is  - $status\n";
```

for loop

```
#!/usr/bin/perl

# Perl program to illustrate
# the use of for Loop

# for loop
print("For Loop:\n");
for ($count = 1 ; $count <= 3 ; $count++)
{
    print "GeeksForGeeks\n"
}
```

foreach loop

```
#!/usr/bin/perl

# Perl program to illustrate
# the use of foreach Loop

# Array
@data = ('GEEKS', 4, 'GEEKS');

# foreach loop
print("For-each Loop:\n");
foreach $word (@data)
{
    print ("$word ");
}
```

while and do-while

```
#!/usr/bin/perl

# Perl program to illustrate
# the use of foreach Loop

# while loop
$count = 3;

print("While Loop:\n");
while ($count >= 0)
{
    $count = $count - 1;
    print "GeeksForGeeks\n";
}

print("\ndo...while Loop:\n");
$a = 10;

# do..While loop
do {

    print "$a ";
    $a = $a - 1;
} while ($a > 0);
```

Object Oriented Programming

Class and object

```
#!/usr/bin/perl

# Perl Program for creation of a
# Class and its object
use strict;
use warnings;

package student;

# constructor
sub student_data
{

    # shift will take package name 'student'
    # and assign it to variable 'class'
    my $class_name = shift;
    my $self = {
                'StudentFirstName' => shift,
                'StudentLastName' => shift
            };
    # Using bless function
    bless $self, $class_name;

    # returning object from constructor
    return $self;
}

# Object creating and constructor calling
my $Data = student_data student("Geeks", "forGeeks");

# Printing the data
print "$Data->{'StudentFirstName'}\n";
print "$Data->{'StudentLastName'}\n";
```

Subroutines

```
#!/usr/bin/perl

# Perl Program to demonstrate the
# subroutine declaration and calling

# defining subroutine
sub ask_user
{
    print "Hello Geeks!\n";
}

# calling subroutine
# you can also use
# &ask_user();
ask_user();
```

Modules and Packages

```
#!/usr/bin/perl

# Using the Package 'Calculator'
use Calculator;

print "Enter two numbers to multiply";

# Defining values to the variables
$a = 5;
$b = 10;

# Subroutine call
Calculator::multiplication($a, $b);

print "\nEnter two numbers to divide";

# Defining values to the variables
$a = 45;
$b = 5;

# Subroutine call
Calculator::division($a, $b);
```

References

```
# Perl program to illustrate the
# Referencing and Dereferencing
```

```
# of an Array

# defining an array
@array = ('1', '2', '3');

# making an reference to an array variable
$reference_array = \@array;

# Dereferencing
# printing the value stored
# at $reference_array by prefixing
# @ as it is a array reference
print @$reference_array;
```

Regular Expression

```
# Perl program to demonstrate
# the m// and =~ operators

# Actual String
$a = "GEEKSFORGEEKS";

# Prints match found if
# its found in $a
if ($a =~ m[GEEKS])
{
    print "Match Found\n";
}

# Prints match not found
# if its not found in $a
else
{
    print "Match Not Found\n";
}
```

File Handling

```
# Opening the file
open(fh, "GFG2.txt") or die "File '$filename' can't be opened";

# Reading First line from the file
$firstline = <fh>;
print "$firstline\n";
```

File Test Operators

```
#!/usr/bin/perl

# Using predefined modules
use warnings;
use strict;

# Providing path of file to a variable
my $filename = 'C:\Users\GeeksForGeeks\GFG.txt';

# Checking for the file existence
if(-e $filename)
{

    # If File exists
    print("File $filename exists\n");
}

else
{

    # If File doesn't exists
    print("File $filename does not exists\n");
}
```

Working with Excel Files

```
#!/usr/bin/perl
use Excel::Writer::XLSX;

my $Excelbook = Excel::Writer::XLSX->new( 'GFG_Sample.xlsx' );
my $Excelsheet = $Excelbook->add_worksheet();

$Excelsheet->write( "A1", "Hello!" );
$Excelsheet->write( "A2", "GeeksForGeeks" );
$Excelsheet->write( "B1", "Next_Column" );

$Excelbook->close;
```

Reading from an Excel File:

```
use 5.016;
use Spreadsheet::Read qw(ReadData);
my $book_data = ReadData (‘new_excel.xlsx');
```

```
say 'A2: ' . $book_data->[1]{A2};
```

Error Handling

```
if(open(DATA, $file)) {
   ...
} else {
   die "Error: Couldn't open the file - $!"
}
#example
open(DATA, $file) || die "Error: Couldn't open the file $!";
## example
unless(chdir("/etc")) {
   die "Error: Can't change directory - $!";
}
##example
print(exists($hash{value}) ? 'There' : 'Missing',"\n");
```

The warn Function

```
chdir('/etc') or warn "Can't change directory";
```

The die function

```
chdir('/etc') or die "Can't change directory";
```

Errors within Modules

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp;

sub function {
   warn "Error in module!";
}
1;
#use T;
```

```
#function();
# all below code call the funtion
```

The carp Function

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp;

sub function {
   carp "Error in module!";
}
1;
```

The cluck Function

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp qw(cluck);

sub function {
   cluck "Error in module!";
}
1;
```

The croak Function

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp;

sub function {
   croak "Error in module!";
```

```
}
1;
```

The confess Function

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp;

sub function {
   confess "Error in module!";
}
1;
```

Date and Time

Current Date and Time

```
#!/usr/local/bin/perl

@months = qw( Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec );
@days = qw(Sun Mon Tue Wed Thu Fri Sat Sun);

($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime();
print "$mday $months[$mon] $days[$wday]\n";
#or
#!/usr/local/bin/perl

$datestring = localtime();
print "Local date and time $datestring\n";
```

GMT Time

```
#!/usr/local/bin/perl

$datestring = gmtime();
print "GMT date and time $datestring\n";
```

Format Date and Time

```
#!/usr/local/bin/perl

($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime();

printf("Time Format - HH:MM:SS\n");
printf("%02d:%02d:%02d", $hour, $min, $sec);
```

Epoch time

```
#!/usr/local/bin/perl

$epoc = time();

print "Number of seconds since Jan 1, 1970 - $epoc\n";
#or
#!/usr/local/bin/perl

$datestring = localtime();
print "Current date and time $datestring\n";

$epoc = time();
$epoc = $epoc - 24 * 60 * 60;   # one day before of current date.

$datestring = localtime($epoc);
print "Yesterday's date and time $datestring\n";
```

POSIX Function strftime()

%a	Abbreviated weekday name	Thu
%A	Full weekday name	Thursday
%b	Abbreviated month name	Aug
%B	Full month name	August
%c	Date and time representation	Thu Aug 23 14:55:02 2001
%C	Year divided by 100 and truncated to integer (00-99)	20
%d	Day of the month, zero-padded (01-31)	23
%D	Short MM/DD/YY date, equivalent to %m/%d/%y	08/23/01

%e	Day of the month, space-padded (1-31)	23
%F	Short YYYY-MM-DD date, equivalent to %Y-%m-%d	2001-08-23
%g	Week-based year, last two digits (00-99)	01
%G	Week-based year	2001
%h	Abbreviated month name (same as %b)	Aug
%H	Hour in 24h format (00-23)	14
%I	Hour in 12h format (01-12)	02
%j	Day of the year (001-366)	235
%m	Month as a decimal number (01-12)	08
%M	Minute (00-59)	55
%n	New-line character ('\n')	
%p	AM or PM designation	PM
%r	12-hour clock time	02:55:02 pm
%R	24-hour HH:MM time, equivalent to %H:%M	14:55
%S	Second (00-61)	02
%t	Horizontal-tab character ('\t')	
%T	ISO 8601 time format (HH:MM:SS), equivalent to %H:%M:%S	14:55
%u	ISO 8601 weekday as number with Monday as 1 (1-7)	4
%U	Week number with the first Sunday as the first day of week one (00-53)	33
%V	ISO 8601 week number (00-53)	34
%w	Weekday as a decimal number with Sunday as 0 (0-6)	4
%W	Week number with the first Monday as the first day of week one (00-53)	34
%x	Date representation	08/23/01
%X	Time representation	14:55:02

%y	Year, last two digits (00-99)	01
%Y	Year	2001
%z	ISO 8601 offset from UTC in timezone (1 minute = 1, 1 hour = 100)If timezone cannot be termined, no characters	+100
%Z	Timezone name or abbreviation If timezone cannot be termined, no characters	CDT
%%	A % sign	%

```
#!/usr/local/bin/perl
use POSIX qw(strftime);

$datestring = strftime "%a %b %e %H:%M:%S %Y", localtime;
printf("date and time - $datestring\n");

# or for GMT formatted appropriately for your locale:
$datestring = strftime "%a %b %e %H:%M:%S %Y", gmtime;
printf("date and time - $datestring\n");
```


Top Cheatsheet

Python Cheatsheet
Quick Reference

Vim Cheatsheet
Quick Reference

JavaScript Cheatsheet
Quick Reference

Bash Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

Find

This is a quick reference list of cheatsheet for linux find command, contains common options and examples.

Getting Started

Usage

```
$ find [path...] [options] [expression]
```

Wildcard

```
$ find . -name "*.txt"
$ find . -name "2020*.csv"
$ find . -name "json_*"
```

- Regex reference (cheatsheets.zip)
- Find cheatsheet (gist.github.com)

Option Examples

Option	Example	Description
`-type`	find . -type d	Find only directories
`-name`	find . -type f -name "*.txt"	Find file by name
`-iname`	find . -type f -iname "hello"	Find file by name (case-insensitive)
`-size`	find . -size +1G	Find files larger than 1G
`-user`	find . -type d -user jack	Find jack's file
`-regex`	find /var -regex '.*/tmp/.*[0-9]*.file'	Using Regex with find. See regex

Option	Example	Description
`-maxdepth`	find . -maxdepth 1 -name "a.txt"	In the current directory and subdirectories
`-mindepth`	find / -mindepth 3 -maxdepth 5 -name pass	Between sub-directory level 2 and 4

Type

`-type d`	Directory
`-type f`	File
`-type l`	Symbolic link
`-type b`	Buffered block
`-type c`	Unbuffered character
`-type p`	Named pipe
`-type s`	Socket

Size

`-size b`	512-byte blocks (default)
`-size c`	Bytes
`-size k`	Kilobytes
`-size M`	Megabytes
`-size G`	Gigabytes
`-size T`	Terabytes (only BSD)
`-size P`	Petabytes (only BSD)

Size +/-

Find all bigger than 10MB files

```
$ find / -size +10M
```

Find all smaller than 10MB files

```
$ find / -size -10M
```

Find all files that are exactly 10M

```
$ find / -size 10M
```

Find Size between 100MB and 1GB

```
$ find / -size +100M -size -1G
```

The + and - prefixes signify greater than and less than, as usual.

Names

Find files using name in current directory

```
$ find . -name tecmint.txt
```

Find files under home directory

```
$ find /home -name tecmint.txt
```

Find files using name and ignoring case

```
$ find /home -iname tecmint.txt
```

Find directories using name

```
$ find / -type d -name tecmint
```

Find php files using name

```
$ find . -type f -name tecmint.php
```

Find all php files in directory

```
$ find . -type f -name "*.php"
```

Permissions

Find the files whose permissions are 777.

```
$ find . -type f -perm 0777 -print
```

Find the files without permission 777.

```
$ find / -type f ! -perm 777
```

Find SUID set files.

```
$ find / -perm /u=s
```

Find SGID set files.

```
$ find / -perm /g=s
```

Find Read Only files.

```
$ find / -perm /u=r
```

Find Executable files.

```
$ find / -perm /a=x
```

Owners and Groups

Find single file based on user

```
$ find / -user root -name tecmint.txt
```

Find all files based on user

```
$ find /home -user tecmint
```

Find all files based on group

```
$ find /home -group developer
```

Find particular files of user

```
$ find /home -user tecmint -iname "*.txt"
```

Multiple filenames

```
$ find . -type f \( -name "*.sh" -o -name "*.txt" \)
```

Find files with `.sh` and `.txt` extensions

Multiple dirs

```
$ find /opt /usr /var -name foo.scala -type f
```

Find files with multiple dirs

Empty

```
$ find . -type d -empty
```

Delete all empty files in a directory

```
$ find . -type f -empty -delete
```

Find Date and Time

Means	
`atime`	access time (last time file opened)
`mtime`	modified time (last time file contents was modified)
`ctime`	changed time (last time file inode was changed)
Example	
`-mtime +0`	Modified greater than 24 hours ago
`-mtime 0`	Modified between now and 1 day ago
`-mtime -1`	Modified less than 1 day ago (same as `-mtime 0`)
`-mtime 1`	Modified between 24 and 48 hours ago
`-mtime +1`	Modified more than 48 hours ago
`-mtime +1w`	Last modified more than 1 week ago

`-atime 0`	Last accessed between now and 24 hours ago
`-atime +0`	Accessed more than 24 hours ago
`-atime 1`	Accessed between 24 and 48 hours ago
`-atime +1`	Accessed more than 48 hours ago
`-atime -1`	Accessed less than 24 hours ago (same as `-atime 0`)
`-ctime -6h30m`	File status changed within the last 6 hours and 30 minutes

Examples

Find last 50 days modified files

```
$ find / -mtime 50
```

find last 50 days accessed files

```
$ find / -atime 50
```

find last 50-100 days modified files

```
$ find / -mtime +50 –mtime -100
```

find changed files in last 1 hour

```
$ find / -cmin -60
```

find modified files in last 1 hour

```
$ find / -mmin -60
```

find accessed files in last 1 hour

```
$ find / -amin -60
```

Find and

Find and delete

Find and remove multiple files

```
$ find . -type f -name "*.mp3" -exec rm -f {} \;
```

Find and remove single file

```
$ find . -type f -name "tecmint.txt" -exec rm -f {} \;
```

Find and delete 100mb files

```
$ find / -type f -size +100m -exec rm -f {} \;
```

Find specific files and delete

```
$ find / -type f -name *.mp3 -size +10m -exec rm {} \;
```

Find and replace

Find all files and modify the content `const` to `let`

```
$ find ./ -type f -exec sed -i 's/const/let/g' {} \;
```

Find readable and writable files and modify the content `old` to `new`

```
$ find ./ -type f -readable -writable -exec sed -i "s/old/new/g" {} \;
```

See also: sed cheatsheet

Find and rename

Find and suffix (added `.bak`)

```
$ find . -type f -name 'file*' -exec mv {} {}.bak\;
```

Find and rename extension (`.html` => `.gohtml`)

```
$ find ./ -depth -name "*.html" -exec sh -c 'mv "$1" "${1%.html}.gohtml"' _ {} \;
```

Find and move

```
$ find . -name '*.mp3' -exec mv {} /tmp/music \;
```

Find and move it to a specific directory (`/tmp/music`)

Find and copy

```
$ find . -name '*2020*.xml' -exec cp -r "{}" /tmp/backup \;
```

Find matching files and copy to a specific directory (`/tmp/backup`)

Find and concatenate

Merge all csv files in the download directory into `merged.csv`

```
$ find download -type f -iname '*.csv' | xargs cat > merged.csv
```

Merge all sorted csv files in the download directory into `merged.csv`

```
$ find download -type f -iname '*.csv' | sort | xargs cat > merged.csv
```

Find and sort

Find and sort in ascending

```
$ find . -type f | sort
```

find and sort descending

```
$ find . -type f | sort -r
```

Find and chmod

Find files and set permissions to 644.

```
$ find / -type f -perm 0777 -print -exec chmod 644 {} \;
```

Find directories and set permissions to 755.

```
$ find / -type d -perm 777 -print -exec chmod 755 {} \;
```

Find and compress

Find all `.java` files and compress it into `java.tar`

```
$ find . -type f -name "*.java" | xargs tar cvf java.tar
```

Find all `.csv` files and compress it into `cheatsheets.zip`

```
$ find . -type f -name "*.csv" | xargs zip cheatsheets.zip
```

Related Cheatsheet

Grep Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

Grep

This cheat sheet is intended to be a quick reminder for the main concepts involved in using the command line program grep and assumes you already understand its usage.

Getting Started

Usage

Search standard output (i.e. a stream of text)

```
$ grep [options] search_string
```

Search for an exact string in file:

```
$ grep [options] search_string path/to/file
```

Print lines in myfile.txt containing the string "mellon"

```
$ grep 'mellon' myfile.txt
```

Wildcards are accepted in filename.

Option examples

`-i`	grep -i ^DA demo.txt	Forgets about case sensitivity
`-w`	grep -w "of" demo.txt	Search only for the full word
`-A`	grep -A 3 'Exception' error.log	Display 3 lines after matching string
`-B`	grep -B 4 'Exception' error.log	Display 4 lines before matching string
`-C`	grep -C 5 'Exception' error.log	Display 5 lines around matching string
`-r`	grep -r 'cheatsheets.zip' /var/log/nginx/	Recursive search (within subdirs)

`-v`	grep -v 'warning' /var/log/syslog	Return all lines which don't match the pattern
`-e`	grep -e '^al' filename	Use regex (lines starting with 'al')
`-E`	grep -E 'ja(s|cks)on' filename	Extended regex (lines containing jason or jackson)
`-c`	grep -c 'error' /var/log/syslog	Count the number of matches
`-l`	grep -l 'robot' /var/log/*	Print the name of the file(s) of matches
`-o`	grep -o search_string filename	Only show the matching part of the string
`-n`	grep -n "go" demo.txt	Show the line numbers of the matches

Grep regular expressions

Refer

- Regex syntax (cheatsheets.zip)
- Regex examples (cheatsheets.zip)

Please refer to the full version of the regex cheat sheet for more complex requirements.

Wildcards

.	Any character.
?	Optional and can only occur once.
*	Optional and can occur more than once.
+	Required and can occur more than once.

Quantifiers

`{n}`	Previous item appears exactly n times.
`{n,}`	Previous item appears n times or more.
`{,m}`	Previous item appears n times maximum.
`{n,m}`	Previous item appears between n and m times.

POSIX

`[:alpha:]`	Any lower and upper case letter.
`[:digit:]`	Any number.
`[:alnum:]`	Any lower and upper case letter or digit.
`[:space:]`	Any whitespace.

Character

`[A-Za-z]`	Any lower and upper case letter.
`[0-9]`	Any number.
`[0-9A-Za-z]`	Any lower and upper case letter or digit.

Position

`^`	Beginning of line.
`$`	End of line.
`^$`	Empty line.
`\<`	Start of word.
`\>`	End of word.

Related Cheatsheet

Awk Cheatsheet
Quick Reference

Emacs Cheatsheet
Quick Reference

Markdown Cheatsheet
Quick Reference

Sed Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

Sed

Sed is a stream editor, this sed cheat sheet contains sed commands and some common sed tricks.

Getting Started

Sed Usage

Syntax

```
$ sed [options] command [input-file]
```

With pipeline

```
$ cat report.txt | sed 's/Nick/John/g'
```

```
$ echo '123abc' | sed 's/[0-9]+//g'
```

Option Examples

Option	Example	Description
-i	sed -ibak 's/On/Off/' php.ini	Backup and modify input file directly
-E	sed -E 's/[0-9]+//g' input-file	Use extended regular expressions
-n	sed -n '3 p' config.conf	Suppress default pattern space printing
-f	sed -f script.sed config.conf	Execute sed script file
-e	sed -e 'command1' -e 'command2' input-file	Execute multiple sed commands

Multiple commands

```
$ echo "hello world" | sed -e 's/h/H/g' -e 's/w/W/g'
```

```
Hello World
```

Use `-e` to execute multiple sed commands

Sed script

```
$ echo 's/h/H/g' >> hello.sed
$ echo 's/w/W/g' >> hello.sed
$ echo "hello world" | sed -f hello.sed
Hello World
```

Use `-f` to execute sed script file

Examples

```
$ sed 's/old/new/g' file.txt
$ sed 's/old/new/g' file.txt > new.txt

$ sed 's/old/new/g' -i file.txt
$ sed 's/old/new/g' -i.backup file.txt
```

See: Sed examples

Sed commands

Commands

Command	Example	Description
`p`	sed -n '1,4 p' input.txt	Print lines 1-4
`p`	sed -n -e '1,4 p' -e '6,7 p' input.txt	Print lines 1-4 and 6-7
`d`	sed '1,4 d' input.txt	Print lines except 1-4
`w`	sed -n '1,4 w output.txt' input.txt	Write pattern space to file
`a`	sed '2 a new-line' input.txt	Append line after
`i`	sed '2 i new-line' input.txt	Insert line before

Space commands

`n`	Print pattern space, empty pattern space, and read next line
`x`	Swap pattern space with hold space
`h`	Copy pattern space to hold space
`H`	Append pattern space to hold space
`g`	Copy hold space to pattern space
`G`	Append hold space to pattern space

See also: File spacing

Flags

```
$ sed 's/old/new/[flags]' [input-file]
```

`g`	Global substitution
`1,2...`	Substitute the nth occurrence
`p`	Print only the substituted line
`w`	Write only the substituted line to a file
`I`	Ignore case while searching
`e`	Substitute and execute in the command line

Loops commands

`b label`	Branch to a label (for looping)
`t label`	Branch to a label only on successful substitution (for looping)
`:label`	Label for the b and t commands (for looping)
`N`	Append next line to pattern space
`P`	Print 1st line in multi-line
`D`	Delete 1st line in multi-line

Misc Flags

`/ | ^ @ ! #`	Substitution delimiter can be any character

`&`	Gets the matched pattern
`( ) \1 \2 \3`	Group using `(` and `)`. Use `\1`, `\2` in replacement to refer the group

Sed examples

Replacing text

Replace all occurrences of a string

```
$ sed 's/old/new/g' file.txt
```

Replace only the nth occurrence of a string

```
$ sed 's/old/new/2' file.txt
```

Replace replace a string only on the 5th line

```
$ sed '5 s/old/new/' file.txt
```

Replace "world" with "universe" but only if the line begins with "hello"

```
$ sed '/hello/s/world/universe/' file.txt
```

Remove "" from the end of each line

```
$ sed 's/\\$//' file.txt
```

Remove all whitespace from beginning of each line

```
$ sed 's/^\s*//' file.txt
```

Remove comments. Even those that are at the end of a line

```
$ sed 's/#.*$//' file.txt
```

Search for text

Search for a string and only print the lines that were matched

```
$ sed -n '/hello/p' file.txt
```

Case insensitive search

```
$ sed -n '/hello/Ip' file.txt
```

Search for a string but only output lines that do not match

```
$ sed -n '/hello/!p' file.txt
```

Appending lines

Append line after line 2

```
$ sed '2a Text after line 2' file.txt
```

Append line at the end of the file

```
$ sed '$a THE END!' file.txt
```

Append line after every 3rd line starting from line 3

```
$ sed '3~3a Some text' file.txt
```

Numbering

Number line of a file (simple left alignment)

```
$ sed = file.txt | sed 'N;s/\n/\t/'
```

Number line of a file (number on left, right-aligned)

```
$ sed = file.txt | sed 'N; s/^/   /; s/ *\(.\{6,\}\)\n/\1  /'
```

Number line of file, but only print numbers if line is not blank

```
$ sed '/./=' file.txt | sed '/./N; s/\n/ /'
```

Count lines (emulates "wc -l")

```
$ sed -n '$='
```

Prepending lines

Insert text before line 5

```
$ sed '5i line number five' file.txt
```

Insert "Example: " before each line that contains "hello"

```
$ sed '/hello/i Example: ' file.txt
```

Deleting lines

Delete line 5-7 in file

```
$ sed '5,7d' file.txt
```

Delete every 2nd line starting with line 3

```
$ sed '3~2d' file.txt
```

Delete the last line in file

```
$ sed '$d' file.txt
```

Delete lines starting with "Hello"

```
$ sed '/^Hello/d' file.txt
```

Delete all empty lines

```
$ sed '/^$/d' file.txt
```

Delete lines starting with "#"

```
$ sed '/^#/d' file.txt
```

File spacing

Double space

```
$ sed G
```

Delete all blank lines and double space

```
$ sed '/^$/d;G'
```

Triple space a file

```
$ sed 'G;G'
```

Undo double-spacing

```
$ sed 'n;d'
```

Insert a blank line above line which matches "regex"

```
$ sed '/regex/{x;p;x;}'
```

Insert a blank line below line which matches "regex"

```
$ sed '/regex/G'
```

Insert a blank line around line which matches "regex"

```
$ sed '/regex/{x;p;x;G;}'
```

Also see

sed cheatsheet (gist.github.com)

Related Cheatsheet

Emacs Cheatsheet

Vim Cheatsheet

Quick Reference

Quick Reference

VSCode Cheatsheet
Quick Reference

Awk Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

RegEX

A quick reference for regular expressions (regex), including symbols, ranges, grouping, assertions and some sample patterns to get you started.

Getting Started

Introduction

This is a quick cheat sheet to getting started with regular expressions.

- Regex in Python (cheatsheets.zip)
- Regex in JavaScript (cheatsheets.zip)
- Regex in PHP (cheatsheets.zip)
- Regex in Java (cheatsheets.zip)
- Regex in MySQL (cheatsheets.zip)
- Regex in Vim (cheatsheets.zip)
- Regex in Emacs (cheatsheets.zip)
- Online regex tester (regex101.com)

Character Classes

`[abc]`

A single character of: a, b or c

`[^abc]`

A character except: a, b or c

`[a-z]`

A character in the range: a-z

`[^a-z]`

A character not in the range: a-z

`[0-9]`

A digit in the range: 0-9

`[a-zA-Z]`

A character in the range: a-z or A-Z

`[a-zA-Z0-9]`

A character in the range: a-z, A-Z or 0-9

Quantifiers

`a?`	Zero or one of a
`a*`	Zero or more of a
`a+`	One or more of a
`[0-9]+`	One or more of 0-9
`a{3}`	Exactly 3 of a
`a{3,}`	3 or more of a
`a{3,6}`	Between 3 and 6 of a
`a*`	Greedy quantifier
`a*?`	Lazy quantifier
`a*+`	Possessive quantifier

Common Metacharacters

^	{	+
<	[	*
)	>	.
(	|	$
\	?	

Escape these special characters with \

Meta Sequences

.	Any single character
\s	Any whitespace character

\S	Any non-whitespace character
\d	Any digit, Same as [0-9]
\D	Any non-digit, Same as [^0-9]
\w	Any word character
\W	Any non-word character
\X	Any Unicode sequences, linebreaks included
\C	Match one data unit
\R	Unicode newlines
\v	Vertical whitespace character
\V	Negation of \v - anything except newlines and vertical tabs
\h	Horizontal whitespace character
\H	Negation of \h
\K	Reset match
\n	Match nth subpattern
\pX	Unicode property X
\p{...}	Unicode property or script category
\PX	Negation of \pX
\P{...}	Negation of \p
\Q...\E	Quote; treat as literals
\k<name>	Match subpattern name
\k'name'	Match subpattern name
\k{name}	Match subpattern name
\gn	Match nth subpattern
\g{n}	Match nth subpattern
\g<n>	Recurse nth capture group

`\g'n'`	Recurses nth capture group.
`\g{-n}`	Match nth relative previous subpattern
`\g<+n>`	Recurse nth relative upcoming subpattern
`\g'+n'`	Match nth relative upcoming subpattern
`\g'letter'`	Recurse named capture group `letter`
`\g{letter}`	Match previously-named capture group `letter`
`\g<letter>`	Recurses named capture group `letter`
`\xYY`	Hex character YY
`\x{YYYY}`	Hex character YYYY
`\ddd`	Octal character ddd
`\cY`	Control character Y
`[\b]`	Backspace character
`\`	Makes any character literal

Anchors

`\G`	Start of match
`^`	Start of string
`$`	End of string
`\A`	Start of string
`\Z`	End of string
`\z`	Absolute end of string
`\b`	A word boundary
`\B`	Non-word boundary

Substitution

`\0`	Complete match contents
`\1`	Contents in capture group 1

`$1`	Contents in capture group 1
`${foo}`	Contents in capture group `foo`
`\x20`	Hexadecimal replacement values
`\x{06fa}`	Hexadecimal replacement values
`\t`	Tab
`\r`	Carriage return
`\n`	Newline
`\f`	Form-feed
`\U`	Uppercase Transformation
`\L`	Lowercase Transformation
`\E`	Terminate any Transformation

Group Constructs

`(...)`	Capture everything enclosed
`(a|b)`	Match either a or b
`(?:...)`	Match everything enclosed
`(?>...)`	Atomic group (non-capturing)
`(?|...)`	Duplicate subpattern group number
`(?#...)`	Comment
`(?'name'...)`	Named Capturing Group
`(?<name>...)`	Named Capturing Group
`(?P<name>...)`	Named Capturing Group
`(?imsxXU)`	Inline modifiers
`(?(DEFINE)...)`	Pre-define patterns before using them

Assertions

`(?(1)yes|no)`	Conditional statement

`(?(R)yes|no)`	Conditional statement
`(?(R#)yes|no)`	Recursive Conditional statement
`(?(R&name\yes|no)`	Conditional statement
`(?(?=...)yes|no)`	Lookahead conditional
`(?(?<=...)yes|no)`	Lookbehind conditional

Lookarounds

`(?=...)`	Positive Lookahead
`(?!...)`	Negative Lookahead
`(?<=...)`	Positive Lookbehind
`(?<!...)`	Negative Lookbehind

Lookaround lets you match a group before (lookbehind) or after (lookahead) your main pattern without including it in the result.

Flags/Modifiers

`g`	Global
`m`	Multiline
`i`	Case insensitive
`x`	Ignore whitespace
`s`	Single line
`u`	Unicode
`X`	eXtended
`U`	Ungreedy
`A`	Anchor
`J`	Duplicate group names

Recurse

`(?R)`	Recurse entire pattern

`(?1)`	Recurse first subpattern
`(?+1)`	Recurse first relative subpattern
`(?&name)`	Recurse subpattern `name`
`(?P=name)`	Match subpattern `name`
`(?P>name)`	Recurse subpattern `name`

POSIX Character Classes

Character Class	Same as	Meaning
`[[:alnum:]]`	`[0-9A-Za-z]`	Letters and digits
`[[:alpha:]]`	`[A-Za-z]`	Letters
`[[:ascii:]]`	`[\x00-\x7F]`	ASCII codes 0-127
`[[:blank:]]`	`[\t ]`	Space or tab only
`[[:cntrl:]]`	`[\x00-\x1F\x7F]`	Control characters
`[[:digit:]]`	`[0-9]`	Decimal digits
`[[:graph:]]`	`[[:alnum:][:punct:]]`	Visible characters (not space)
`[[:lower:]]`	`[a-z]`	Lowercase letters
`[[:print:]]`	`[ -~] == [ [:graph:]]`	Visible characters
`[[:punct:]]`	`[!"#$%&’()*+,-./:;<=>?@[]^_`{|}~]`	Visible punctuation characters
`[[:space:]]`	`[\t\n\v\f\r ]`	Whitespace
`[[:upper:]]`	`[A-Z]`	Uppercase letters
`[[:word:]]`	`[0-9A-Za-z_]`	Word characters
`[[:xdigit:]]`	`[0-9A-Fa-f]`	Hexadecimal digits
`[[:<:]]`	`[\b(?=\w)]`	Start of word
`[[:>:]]`	`[\b(?<=\w)]`	End of word

Control verb

`(*ACCEPT)`	Control verb

(*FAIL)	Control verb
(*MARK:NAME)	Control verb
(*COMMIT)	Control verb
(*PRUNE)	Control verb
(*SKIP)	Control verb
(*THEN)	Control verb
(*UTF)	Pattern modifier
(*UTF8)	Pattern modifier
(*UTF16)	Pattern modifier
(*UTF32)	Pattern modifier
(*UCP)	Pattern modifier
(*CR)	Line break modifier
(*LF)	Line break modifier
(*CRLF)	Line break modifier
(*ANYCRLF)	Line break modifier
(*ANY)	Line break modifier
\R	Line break modifier
(*BSR_ANYCRLF)	Line break modifier
(*BSR_UNICODE)	Line break modifier
(*LIMIT_MATCH=x)	Regex engine modifier
(*LIMIT_RECURSION=d)	Regex engine modifier
(*NO_AUTO_POSSESS)	Regex engine modifier
(*NO_START_OPT)	Regex engine modifier

Regex examples

Characters

`ring`	Match ring springboard etc.
`.`	Match a, 9, + etc.
`h.o`	Match hoo, h2o, h/o etc.
`ring\?`	Match ring?
`\(quiet\)`	Match (quiet)
`c:\\windows`	Match c:\windows

Use `\` to search for these special characters:
`[ \ ^ $ . | ? * + ( ) { }`

Alternatives

`cat|dog`	Match cat or dog
`id|identity`	Match id or identity
`identity|id`	Match id or identity

Order longer to shorter when alternatives overlap

Character classes

`[aeiou]`	Match any vowel
`[^aeiou]`	Match a NON vowel
`r[iau]ng`	Match ring, wrangle, sprung, etc.
`gr[ae]y`	Match gray or grey
`[a-zA-Z0-9]`	Match any letter or digit
`[\u3a00-\ufa99]`	Match any Unicode Hàn (中文)

In `[ ]` always escape `. \ ]` and sometimes `^ - .`

Shorthand classes

`\w`	"Word" character (letter, digit, or underscore)

`\d`	Digit
`\s`	Whitespace (space, tab, vtab, newline)
`\W, \D, or \S`	Not word, digit, or whitespace
`[\D\S]`	Means not digit or whitespace, both match
`[^\d\s]`	Disallow digit and whitespace

Occurrences

`colou?r`	Match color or colour
`[BW]ill[ieamy's]*`	Match Bill, Willy, William's etc.
`[a-zA-Z]+`	Match 1 or more letters
`\d{3}-\d{2}-\d{4}`	Match a SSN
`[a-z]\w{1,7}`	Match a UW NetID

Greedy versus lazy

`* + {n,}` greedy	Match as much as possible
`<.+>`	Finds 1 big match in bold
`*? +? {n,}?` lazy	Match as little as possible
`<.+?>`	Finds 2 matches in bold

Scope

`\b`	"Word" edge (next to non "word" character)
`\bring`	Word starts with "ring", ex ringtone
`ring\b`	Word ends with "ring", ex spring
`\b9\b`	Match single digit 9, not 19, 91, 99, etc..
`\b[a-zA-Z]{6}\b`	Match 6-letter words
`\B`	Not word edge

`\Bring\B`	Match springs and wringer
`^\d*$`	Entire string must be digits
`^[a-zA-Z]{4,20}$`	String must have 4-20 letters
`^[A-Z]`	String must begin with capital letter
`[\.!?"')]$`	String must end with terminal puncutation

Modifiers

`(?i)`[a-z]*`(?-i)`	Ignore case ON / OFF
`(?s).*(?-s)`	Match multiple lines (causes . to match newline)
`(?m)^.*;$(?-m)`	^ & $ match lines not whole string
`(?x)`	#free-spacing mode, this EOL comment ignored
`(?-x)`	free-spacing mode OFF
/regex/`ismx`	Modify mode for entire string

Groups

`(in|out)put`	Match input or output
`\d{5}(-\d{4})?`	US zip code ("+ 4" optional)

Parser tries EACH alternative if match fails after group.
Can lead to catastrophic backtracking.

Back references

`(to) (be) or not \1 \2`	Match to be or not to be
`([^\s])\1{2}`	Match non-space, then same twice more aaa, ...
`\b(\w+)\s+\1\b`	Match doubled words

Non-capturing group

`on(?:click|load)`	Faster than: `on(click|load)`

Use non-capturing or atomic groups when possible

Atomic groups

`(?>red|green|blue)`	Faster than non-capturing
`(?>id|identity)\b`	Match id, but not identity

"id" matches, but `\b` fails after atomic group, parser doesn't backtrack into group to retry 'identity'

If alternatives overlap, order longer to shorter.

Lookaround

`(?= )`	Lookahead, if you can find ahead
`(?! )`	Lookahead,if you can not find ahead
`(?<= )`	Lookbehind, if you can find behind
`(?<! )`	Lookbehind, if you can NOT find behind
`\b\w+?(?=ing\b)`	Match warbling, string, fishing, ...
`\b(?!\w+ing\b)\w+\b`	Words NOT ending in ing
`(?<=\bpre).*?\b`	Match pretend, present, prefix, ...
`\b\w{3}(?<!pre)\w*?\b`	Words NOT starting with pre
`\b\w+(?<!ing)\b`	Match words NOT ending in ing

If-then-else

Match "Mr." or "Ms." if word "her" is later in string

```
M(?(?=.*?\bher\b)s|r)\.
```

requires lookaround for IF condition

RegEx in Python

Getting started

Import the regular expressions module

```
import re
```

Examples

re.search()

```
>>> sentence = 'This is a sample string'
>>> bool(re.search(r'this', sentence, flags=re.I))
True
>>> bool(re.search(r'xyz', sentence))
False
```

re.findall()

```
>>> re.findall(r'\bs?pare?\b', 'par spar apparent spare part pare')
['par', 'spar', 'spare', 'pare']
>>> re.findall(r'\b0*[1-9]\d{2,}\b', '0501 035 154 12 26 98234')
['0501', '154', '98234']
```

re.finditer()

```
>>> m_iter = re.finditer(r'[0-9]+', '45 349 651 593 4 204')
>>> [m[0] for m in m_iter if int(m[0]) < 350]
['45', '349', '4', '204']
```

re.split()

```
>>> re.split(r'\d+', 'Sample123string42with777numbers')
['Sample', 'string', 'with', 'numbers']
```

re.sub()

```
>>> ip_lines = "catapults\nconcatenate\ncat"
>>> print(re.sub(r'^', r'* ', ip_lines, flags=re.M))
* catapults
* concatenate
* cat
```

re.compile()

```
>>> pet = re.compile(r'dog')
>>> type(pet)
<class '_sre.SRE_Pattern'>
>>> bool(pet.search('They bought a dog'))
True
>>> bool(pet.search('A cat crossed their path'))
False
```

Functions

`re.findall`	Returns a list containing all matches
`re.finditer`	Return an iterable of match objects (one for each match)
`re.search`	Returns a Match object if there is a match anywhere in the string
`re.split`	Returns a list where the string has been split at each match
`re.sub`	Replaces one or many matches with a string
`re.compile`	Compile a regular expression pattern for later use
`re.escape`	Return string with all non-alphanumerics backslashed

Flags

`re.I`	`re.IGNORECASE`	Ignore case
`re.M`	`re.MULTILINE`	Multiline
`re.L`	`re.LOCALE`	Make `\w,\b,\s` locale dependent
`re.S`	`re.DOTALL`	Dot matches all (including newline)
`re.U`	`re.UNICODE`	Make `\w,\b,\d,\s` unicode dependent
`re.X`	`re.VERBOSE`	Readable style

Regex in JavaScript

test()

```
let textA = "I like APPles very much";
let textB = "I like APPles";
let regex = /apples$/i;

// Output: false
console.log(regex.test(textA));

// Output: true
console.log(regex.test(textB));
```

search()

```
let text = "I like APPles very much";
let regexA = /apples/;
let regexB = /apples/i;

// Output: -1
console.log(text.search(regexA));

// Output: 7
console.log(text.search(regexB));
```

exec()

```
let text = "Do you like apples?";
let regex = /apples/;

// Output: apples
console.log(regex.exec(text)[0]);

// Output: Do you like apples?
console.log(regex.exec(text).input);
```

match()

```
let text = "Here are apples and apPleS";
let regex = /apples/gi;

// Output: [ "apples", "apPleS" ]
console.log(text.match(regex));
```

split()

```
let text = "This 593 string will be brok294en at places where d1gits are.";
let regex = /\d+/g;

// Output: [ "This ", " string will be brok", "en at places where d", "gits are."
console.log(text.split(regex));
```

matchAll()

```
let regex = /t(e)(st(\d?))/g;
let text = "test1test2";
```

```
let array = [...text.matchAll(regex)];

// Output: ["test1", "e", "st1", "1"]
console.log(array[0]);

// Output: ["test2", "e", "st2", "2"]
console.log(array[1]);
```

replace()

```
let text = "Do you like aPPles?";
let regex = /apples/i;

// Output: Do you like mangoes?
let result = text.replace(regex, "mangoes");
console.log(result);
```

replaceAll()

```
let regex = /apples/gi;
let text = "Here are apples and apPleS";

// Output: Here are mangoes and mangoes
let result = text.replaceAll(regex, "mangoes");
console.log(result);
```

Regex in PHP

Functions

`preg_match()`	Performs a regex match
`preg_match_all()`	Perform a global regular expression match
`preg_replace_callback()`	Perform a regular expression search and replace using a callback
`preg_replace()`	Perform a regular expression search and replace
`preg_split()`	Splits a string by regex pattern
`preg_grep()`	Returns array entries that match a pattern

preg_replace

```
$str = "Visit Microsoft!";
$regex = "/microsoft/i";

// Output: Visit CheatSheets!
echo preg_replace($regex, "CheatSheets", $str);
```

preg_match

```
$str = "Visit CheatSheets";
$regex = "#cheatsheets#i";

// Output: 1
echo preg_match($regex, $str);
```

preg_matchall

```
$regex = "/[a-zA-Z]+ (\d+)/";
$input_str = "June 24, August 13, and December 30";
if (preg_match_all($regex, $input_str, $matches_out)) {

    // Output: 2
    echo count($matches_out);

    // Output: 3
    echo count($matches_out[0]);

    // Output: Array("June 24", "August 13", "December 30")
    print_r($matches_out[0]);

    // Output: Array("24", "13", "30")
    print_r($matches_out[1]);
}
```

preg_grep

```
$arr = ["Jane", "jane", "Joan", "JANE"];
$regex = "/Jane/";

// Output: Jane
echo preg_grep($regex, $arr);
```

preg_split

```
$str = "Jane\tKate\nLucy Marion";
$regex = "@\s@";

// Output: Array("Jane", "Kate", "Lucy", "Marion")
print_r(preg_split($regex, $str));
```

Regex in Java

Styles

First way

```
Pattern p = Pattern.compile(".s", Pattern.CASE_INSENSITIVE);
Matcher m = p.matcher("aS");
boolean s1 = m.matches();
System.out.println(s1);   // Outputs: true
```

Second way

```
boolean s2 = Pattern.compile("[0-9]+").matcher("123").matches();
System.out.println(s2);   // Outputs: true
```

Third way

```
boolean s3 = Pattern.matches(".s", "XXXX");
System.out.println(s3);   // Outputs: false
```

Pattern Fields

CANON_EQ	Canonical equivalence
CASE_INSENSITIVE	Case-insensitive matching
COMMENTS	Permits whitespace and comments
DOTALL	Dotall mode
MULTILINE	Multiline mode
UNICODE_CASE	Unicode-aware case folding
UNIX_LINES	Unix lines mode

Methods

Pattern

- Pattern compile(String regex [, int flags])
- boolean matches([String regex,] CharSequence input)
- String[] split(String regex [, int limit])
- String quote(String s)

Matcher

- int start([int group | String name])
- int end([int group | String name])
- boolean find([int start])
- String group([int group | String name])
- Matcher reset()

String

- boolean matches(String regex)
- String replaceAll(String regex, String replacement)
- String[] split(String regex[, int limit])

There are more methods ...

Examples

Replace sentence:

```
String regex = "[A-Z\n]{5}$";
String str = "I like APP\nLE";

Pattern p = Pattern.compile(regex, Pattern.MULTILINE);
Matcher m = p.matcher(str);

// Outputs: I like Apple!
System.out.println(m.replaceAll("pple!"));
```

Array of all matches:

```
String str = "She sells seashells by the Seashore";
String regex = "\\w*se\\w*";

Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
Matcher m = p.matcher(str);

List<String> matches = new ArrayList<>();
while (m.find()) {
    matches.add(m.group());
}

// Outputs: [sells, seashells, Seashore]
System.out.println(matches);
```

Regex in MySQL

Functions

`REGEXP`	Whether string matches regex
`REGEXP_INSTR()`	Starting index of substring matching regex (NOTE: Only MySQL 8.0+)
`REGEXP_LIKE()`	Whether string matches regex (NOTE: Only MySQL 8.0+)
`REGEXP_REPLACE()`	Replace substrings matching regex (NOTE: Only MySQL 8.0+)
`REGEXP_SUBSTR()`	Return substring matching regex (NOTE: Only MySQL 8.0+)

REGEXP

```
expr REGEXP pat
```

Examples

```
mysql> SELECT 'abc' REGEXP '^[a-d]';
1
mysql> SELECT name FROM cities WHERE name REGEXP '^A';
```

```
mysql> SELECT name FROM cities WHERE name NOT REGEXP '^A';
mysql> SELECT name FROM cities WHERE name REGEXP 'A|B|R';
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
1   0
```

REGEXP_REPLACE

```
REGEXP_REPLACE(expr, pat, repl[, pos[, occurrence[, match_type]]])
```

Examples

```
mysql> SELECT REGEXP_REPLACE('a b c', 'b', 'X');
a X c
mysql> SELECT REGEXP_REPLACE('abc ghi', '[a-z]+', 'X', 1, 2);
abc X
```

REGEXP_SUBSTR

```
REGEXP_SUBSTR(expr, pat[, pos[, occurrence[, match_type]]])
```

Examples

```
mysql> SELECT REGEXP_SUBSTR('abc def ghi', '[a-z]+');
abc
mysql> SELECT REGEXP_SUBSTR('abc def ghi', '[a-z]+', 1, 3);
ghi
```

REGEXP_LIKE

```
REGEXP_LIKE(expr, pat[, match_type])
```

Examples

```
mysql> SELECT regexp_like('aba', 'b+')
1
mysql> SELECT regexp_like('aba', 'b{2}')
0
mysql> # i: case-insensitive
mysql> SELECT regexp_like('Abba', 'ABBA', 'i');
1
mysql> # m: multi-line
mysql> SELECT regexp_like('a\nb\nc', '^b$', 'm');
1
```

REGEXP_INSTR

```
REGEXP_INSTR(expr, pat[, pos[, occurrence[, return_option[, match_type]]]])
```

Examples

```
mysql> SELECT regexp_instr('aa aaa aaaa', 'a{3}');
2
mysql> SELECT regexp_instr('abba', 'b{2}', 2);
2
mysql> SELECT regexp_instr('abbabba', 'b{2}', 1, 2);
5
mysql> SELECT regexp_instr('abbabba', 'b{2}', 1, 3, 1);
7
```

Related Cheatsheet

Grep Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

Cron

Cron is most suitable for scheduling repetitive tasks. Scheduling one-time tasks can be accomplished using the associated at utility.

Crontab Format

Format

```
Min  Hour Day  Mon  Weekday
```

```
*    *    *    *    *  command to be executed


T    T    T    T    T
|    |    |    |    └─  Day of Week   (0=Sun .. 6=Sat)
|    |    |    └──────  Month         (1..12)
|    |    └───────────  Day of Month  (1..31)
|    └────────────────  Hour          (0..23)
└─────────────────────  Minute        (0..59)
```

Field	Range	Special characters
Minute	0 - 59	, - * /
Hour	0 - 23	, - * /
Day of Month	1 - 31	, - * ? / L W
Month	1 - 12	, - * /
Day of Week	0 - 6	, - * ? / L #

Examples

`*/15 * * * *`	Every 15 mins

`0 * * * *`	Every hour
`0 */2 * * *`	Every 2 hours
`15 2 * * *`	At 2:15AM of every day
`15 2 * * ?`	At 2:15AM of every day
`10 9 * * 5`	At 9:10AM of every Friday
`0 0 * * 0`	At midnight of every Sunday
`15 2 * * 1L`	At 2:15am on the last monday of every month
`15 0 * * 4#2`	At 00:15am on the second thursday of every month
`0 0 1 * *`	Every 1st of month (monthly)
`0 0 1 1 *`	Every 1st of january (yearly)
`@reboot`	Every reboot (non-standard)

Special strings

@reboot	Run once, at system startup (non-standard)
@yearly	Run once every year, "0 0 1 1 *" (non-standard)
@annually	(same as @yearly) (non-standard)
@monthly	Run once every month, "0 0 1 * *" (non-standard)
@weekly	Run once every week, "0 0 * * 0" (non-standard)
@daily	Run once each day, "0 0 * * *" (non-standard)
@midnight	(same as @daily) (non-standard)
@hourly	Run once an hour, "0 * * * *" (non-standard)

Crontab command

`crontab -e`	Edit or create a crontab file if doesn't already exist.
`crontab -l`	Display the crontab file.
`crontab -r`	Remove the crontab file.

`crontab -v`	Display the last time you edited your crontab file. (non-standard)

Special characters	
`Asterik(*)`	Matches all values in the field or any possible value.
`Hyphen(-)`	Used to define a range.Ex: 1-5 in 5th field(Day Of Week) Every Weekday i.e., Monday to Friday
`Slash (/)`	1st field(Minute) /15 meaning every fifteen minute or increment of range.
`Comma (,)`	Used to separate items.Ex: 2,6,8 in 2nd fields(Hour) executes at 2am,6am and 8am
`L`	It is allowed only for Day of Month or Day Of Week field, 2L in Day of week indicates Last tuesday of every month
`Hash (#)`	It is allowed only for Day Of Week field, which must be followed within range of 1 to 5. For example, 4#1 means "The first Thursday" of given month.
`Question mark (?)`	Can be instead of '*' and allowed for Day of Month and Day Of Week. Usage is restricted to either Day of Month or Day Of Week in a cron expression.

Also see

Devhints (devhints.io)
Crontab Generator (crontab-generator.org)
Crontab guru (crontab.guru)

Top Cheatsheet

Python Cheatsheet
Quick Reference

Vim Cheatsheet
Quick Reference

JavaScript Cheatsheet
Quick Reference

Bash Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

SSH

This quick reference cheat sheet provides various for using SSH.

Getting Started

Connecting

Connect to a server (default port 22)

```
$ ssh [email protected]
```

Connect on a specific port

```
$ ssh [email protected] -p 6222
```

Connect via pem file (0400 permissions)

```
$ ssh -i /path/file.pem [email protected]
```

See: SSH Permissions

Executing

Executes remote command

```
$ ssh [email protected] 'ls -l'
```

Invoke a local script

```
$ ssh [email protected] bash < script.sh
```

Compresses and downloads from a server

```
$ ssh [email protected] "tar cvzf - ~/source" > output.tgz
```

SCP

Copies from remote to local

```
$ scp user@server:/dir/file.ext dest/
```

Copies between two servers

```
$ scp user@server:/file user@server:/dir
```

Copies from local to remote

```
$ scp dest/file.ext user@server:/dir
```

Copies a whole folder

```
$ scp -r user@server:/dir dest/
```

Copies all files from a folder

```
$ scp user@server:/dir/* dest/
```

Copies from a server folder to the current folder

```
$ scp user@server:/dir/* .
```

Config location

`/etc/ssh/ssh_config`	System-wide config
`~/.ssh/config`	User-specific config
`~/.ssh/id_{type}`	Private key
`~/.ssh/id_{type}.pub`	Public key
`~/.ssh/known_hosts`	Known Servers
`~/.ssh/authorized_keys`	Authorized login key

SCP Options

scp -r	Recursively copy entire directories
scp -C	Compresses data
scp -v	Prints verbose info
scp -P 8080	Uses a specific Port
scp -B	Batch mode (Prevents password)
scp -p	Preserves times and modes

Config sample

```
Host server1
    HostName 192.168.1.5
    User root
    Port 22
    IdentityFile ~/.ssh/server1.key
```

Launch by alias

```
$ ssh server1
```

See: Full Config Options

ProxyJump

```
$ ssh -J proxy_host1 remote_host2
```

```
$ ssh -J user@proxy_host1 user@remote_host2
```

Multiple jumps

```
$ ssh -J user@proxy_host1:port1,user@proxy_host2:port2 user@remote_host3
```

ssh-copy-id

```
$ ssh-copy-id user@server
```

Copy to alias server

```
$ ssh-copy-id server1
```

Copy specific key

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub user@server
```

SSH keygen

ssh-keygen

```
$ ssh-keygen -t rsa -b 4096 -C "[email protected]"
```

`-t`	Type of key
`-b`	The number of bits in the key
`-C`	Provides a new comment

Generate an RSA 4096 bit key with email as a comment

Generate

Generate a key interactively

```
$ ssh-keygen
```

Specify filename

```
$ ssh-keygen -f ~/.ssh/filename
```

Generate public key from private key

```
$ ssh-keygen -y -f private.key > public.pub
```

Change comment

```
$ ssh-keygen -c -f ~/.ssh/id_rsa
```

Change private key passphrase

```
$ ssh-keygen -p -f ~/.ssh/id_rsa
```

Key type

- rsa
- ed25519
- dsa
- ecdsa

known_hosts

Search from known_hosts

```
$ ssh-keygen -F <ip/hostname>
```

Remove from known_hosts

```
$ ssh-keygen -R <ip/hostname>
```

Key format

- PEM
- PKCS8

Also see

OpenSSH Config File Examples (cyberciti.biz)
ssh_config (linux.die.net)

Related Cheatsheet

Mitmproxy Cheatsheet
Quick Reference

Netcat Cheatsheet
Quick Reference

Netstat Cheatsheet
Quick Reference

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference

OpenSSL

This is a reference of commands to use to interact with electronic certificates

Private Key

Print out the private key details

```
openssl rsa -check -text -in privateKey.key
```

Print out the hashes of the private key

```
openssl rsa -noout -modulus -in privateKey.key | openssl md5
openssl rsa -noout -modulus -in privateKey.key | openssl sha1
openssl rsa -noout -modulus -in privateKey.key | openssl sha256
openssl rsa -noout -modulus -in privateKey.key | openssl sha512
```

Change password

```
openssl rsa -aes256 -in privateKey.key -out newPrivateKey.key
```

List available elliptic curves

```
openssl ecparam -list_curves
```

Create elliptic curve private key with a specific curve

```
openssl ecparam -name secp521r1 -genkey -noout -out privateKey.key
```

Certificate

Print out the hashes of the certificate

```
openssl x509 -noout -modulus -in certificate.crt | openssl md5
openssl x509 -noout -modulus -in certificate.crt | openssl sha1
openssl x509 -noout -modulus -in certificate.crt | openssl sha256
openssl x509 -noout -modulus -in certificate.crt | openssl sha512
```

Or, alternatively:

```
openssl x509 -noout -fingerprint -in certificate.crt
openssl x509 -noout -fingerprint -sha256 -in certificate.crt
```

Print out the content of the certificates

```
openssl x509 -in certificate.crt -noout -text|more
```

Print out specific fields of the certificates

```
openssl x509 -noout -subject certificate.crt
openssl x509 -noout -issuer certificate.crt
openssl x509 -noout -dates certificate.crt
```

Inspect server certificates

```
echo | openssl s_client -servername www.openssl.org -connect \
www.openssl.org:443 2>/dev/null | openssl x509 -noout -text|more
echo | openssl s_client -servername imap.arcor.de -connect \
imap.arcor.de:993 2>/dev/null | openssl x509 -noout -text|more
```

Verify certificates

OK

```
openssl verify -verbose -x509_strict -CAfile \
issuer.crt Test\ Haeschen\ 1.crt
```

Result:

```
Test Haeschen 1.crt: OK
```

Corrupted (for example)

```
openssl verify -verbose -x509_strict -CAfile \
```

```
issuer.crt Test\ Haeschen\ 1_corrupted.crt
```

Result:

```
C = DE, ST = Thueringen, L = Rudolstadt, O = Damaschkestr. 11, OU = Arbeitszimmer,
error 7 at 0 depth lookup: certificate signature failure
error Test Haeschen 1_corrupted.crt: verification failed
40270500477F0000:error:0200008A:rsa routines:RSA_padding_check_PKCS1_type_1:invali
40270500477F0000:error:02000072:rsa routines:rsa_ossl_public_decrypt:padding check
40270500477F0000:error:1C880004:Provider routines:rsa_verify:RSA lib:../providers/
40270500477F0000:error:06880006:asn1 encoding routines:ASN1_item_verify_ctx:EVP li
```

S/Mime

create signature

```
openssl smime -sign -in msg.txt -text -out msg.p7s \
-signer certificate.crt -inkey privateKey.key
```

Verify signature

```
openssl smime -verify -in msg.p7s -CAfile chain.pem
```

CRL

Print out the contents of the CRL

```
openssl crl -inform DER -noout -text  -in crl/cacrl.der
openssl crl -inform PEM -noout -text  -in crl/cacrl.pem
```

PKCS#12

Display contents

```
openssl pkcs12 -info -in  digitalIdentity.p12
```

Create from certificate and private key

```
openssl pkcs12 -export -in certificate.cert \
-inkey privateKey.key -out digitalIdentity.p12
```

Extract private key

```
openssl pkcs12 -in digitalIdentity.p12 -out privateKey.key
```

Convert to PEM

```
openssl pkcs12 -in digitalIdentity.p12 -out digitalIdentity.pem
```

TSA

Display query

```
openssl ts -query -in query.tsq -text
```

Display reply

```
openssl ts -reply -in reply.tsr -text
```

Verify reply

```
openssl ts -verify -in reply.tsr -data data.dat -CAfile chain.pem
```

Extract token from reply

```
openssl ts -reply -in reply.tsr -token_out -out token.tk
```

Extract certificates from token

```
openssl pkcs7 -inform DER -in token.tk -print_certs -noout -text
```

CSR

Create from existing key

```
openssl req -new -key privateKey.key -out my.csr
```

This can of course be a RSA key or one based on an elliptic curve. Available curves can be listed using

```
openssl ecparam -list_curves
```

Afterwards you chose one of the curves and create a private key like so:

```
openssl ecparam -name secp521r1 -genkey -noout \
-out privateKey.key
```

Display

```
openssl req -in my.csr -noout -text
```

HTTPS

Dump Certificates PEM encoded

```
openssl s_client -showcerts -connect www.example.com:443
```

STARTTLS

Dump Certificates PEM encoded

```
openssl s_client -showcerts -starttls imap \
-connect mail.domain.com:139
```

S/MIME verification

Possible outcomes

Message was tampered with (return code 4):

```
Verification failure
140485684135232:error:2E09A09E:CMS routines:CMS_SignerInfo_verify_content:verifica
140485684135232:error:2E09D06D:CMS routines:CMS_verify:content verify error:../cry
```

Message signature not trusted (return code 4):

```
Verification failure
140146111432000:error:2E099064:CMS routines:cms_signerinfo_verify_cert:certificate
```

Message not signed (return code 2):

```
Error reading S/MIME message
140701208487232:error:0D0D40CD:asn1 encoding routines:SMIME_read_ASN1:invalid mime
```

Validation successful (return code 0):

```
Verification successful
```

Verify the validity of an email message

```
openssl cms -verify -in some_email_message.eml
```

Verify the validity of an email message explicitly specifying trust

```
openssl cms -verify -in some_email_message \
-CAfile trust_anchor-crt
```

Signed and encrypted messages need to be decrypted first:

Note: the P12 file holding the digital identity must be pem-encoded! (see above)

```
openssl cms -decrypt -out decrypted_email_message \
```

```
-inkey p12.pem -in some_encrypted_email_message
```

Raw

See the raw structure of an ASN.1 file (only for DER encoded files)

```
openssl asn1parse -in mysterious_file.pem
```

With a bit more detail

```
openssl asn1parse -dump -strictpem -in mysterious_file.pem
```

Some resources with useful OpenSSL commands

OpenSSL command cheatsheet
21 OpenSSL Examples to Help You in Real-World
The Most Common OpenSSL Commands
OpenSSL Quick Reference Guide
openssl_commands.md
OpenSSL Essentials: Working with SSL Certificates, Private Keys and CSRs
OpenSSL tips and tricks
Checking A Remote Certificate Chain With OpenSSL
OpenSSL: how to extract certificates and token status from RFC3161 timestamping reply?
Steps to generate CSR for SAN certificate with openssl
Howto add a Subject Alternative Name extension into a Certificate Signing Request

Recent Cheatsheet

Gnome Desktop Cheatsheet
Quick Reference

Arduino Programming Cheatsheet
Quick Reference

HTML Canvas Cheatsheet
Quick Reference

TypeScript Cheatsheet
Quick Reference