Chuleta de Comandos Unix

FILE AND DIRECTORY	
pwd	Return path to current directory.
ls	List directories and *file*s here.
ls *dir*	List directories and *file*s in a directory.
ls -d */	List the name of all subdirectory.
ls -a	List all *file*s including hidden *file*s.
ls -lh	List including more data in readable format.
cd *dir*	Change directory.
cd	Go to home directory.
pushd *dir*	Save current directory (push onto stack) and go to *dir*.
popd *dir*	Return (pop from the stack) to last saved directory.
dirs	List all saved directories (the stack).
touch *file*	Create an empty *file*.
mkdir *dir*	Create an empty directory.
ln -s *file*	Create a soft link here to *file*/*dir*
mv *dir1 dir2*	Move or renames a *file*.
cp *file1 file2*	Copy *file1* to *file2*.
cp -r *dir1 dir2*	Copy *dir1* to *dir2* including subdirectories.
rmdir *dir*	Remove an empty directory.
rm *file*	Remove a *file*.
rm -r *dir*	Remove a directory and its subdirectories and *file*s.
rm -f *file*	Remove a *file*, suppress all warning.
find *dir* -name pattern	Search for *file* name matching pattern in *dir*.
md5sum *file*	Calculate MD5 checksum of *file*, there are other algorithms too, e.g. variaous SHA.

READ MANUAL	
help	Display bash help.
help *cmd*	Show usage of built-in commands.
man *cmd*	Show usage of most commands.
info *cmd*	Show more info about command.

FILE ATTRIBUTES	
chmod +x *file*	Set execute permission to *file*.
chmod -x *file*	Unset execute permission to *file*.
chmod IJK *file*	Set permission denoted by IJK to *file*. I, J, K = 0 to 7, to calculate, sum all permissions, read=4, write=2, execute=1. I is for user, J is for group, K is for everyone.
chmod -R IJK *dir*	Set permission denoted by IJK to *dir* and all subdirectories and *file*s.
chown -R user:group *dir*	Change the ownership of *dir* and all subdirectories and *file*s.

COMPRESSION	
tar -cf *file.tar dir*	Group *file*s.
tar -xf *file.tar*	Ungroup *file*s.
tar -zcf *file.tar.gz dir*	Group and compress *file*s.
tar -zxf *file.tar.gz*	Extract and ungroup *file*s.

TEXT VIEWING	
less *file*	View a *file.*
less -N *file*	View *file* with line numbers.
less -S *file*	View *file*, wrap long lines.
cat *file*	Print *file* to STDOUT.
tac *file*	Print *file* to STDOUT in reverse line order.
head *file*	Print first lines from a *file.*
head -n 5 *file*	Print first 5 lines from a *file.*
tail *file*	Print last lines from a *file.*
tail -n 5 *file*	Print last 5 lines from a *file.*
grep *str file*	Display lines containing *str* in *file*.
grep -c *'pattern' file*	Count lines matching a pattern.
sort *file*	Sort lines from a *file.*
sort -u *file*	Sort and return unique lines.
uniq -c *file*	Filter adjacent repeated lines.
wc *file*	Count *file* for line, word and characters.
wc -l *file*	Count number of line for *file*.
diff *file1 file2*	Show difference between *file1* and *file2*.
cut -f 1,3 *file*	Retrieve data from 1,3 columns in a tab-delimited *file.*

REMOTE ACCESS	
wget *url*	Download *url.*
ssh user@server	SSH to a server.
scp -r *local_dir user@server:remote_dir*	Copy *file* from local to remote computer.
scp -r *user@server:remote_dir local_dir*	Copy *file* from remote to local computer.

TEXT EDITING	
paste *file1 file2*	Join *file1* and *file2* line by line.
truncate -s *size file*	Remove contents in file.
nano -S *file*	Nano editor with smooth scrolling.

JOB CONTROL	
ps aux	Show running processes.
pkill -u user	Terminate all process for user.
pkill *cmd*	Terminate a process with SIGTERM.
pkill -9 *cmd*	Terminate *cmd* with SIGKILL.
top	View top CPU using processes.
nohup *cmd*	Run *cmd* disregarding the hangup signal.
cmd &	Run *cmd* in background.
jobs	Show running jobs.
fg N	Bring job N to foreground.
bg	Bring job N to background.

MISC	
echo *string*	Print the string to STDOUT
printf *format-str args*	C like printf
date	Display current date time information.
time *cmd*	Time the execution of *cmd*
sleep N	Wait for N secs.
watch *cmd*	Repeatedly execute *cmd* every 2s and display result.
which *cmd*	Display the resolved command directory.
seq *a b incr*	Generate a list of number starting from *a* to *b* incremented by *incr*.
yes	Keep saying yes
yes *str*	Keep saying *str*, usually used to say "no".

USEFUL FILES	
Descriptor 0	STDIN
Descriptor 1	STDOUT
Descriptor 2	STDERR
/dev/null	A file that discard information
/dev/	A file that provides 0x00
/dev/urandom	A file that provide random bytes
~	Home directory
~+	Directory pointed by PWD
~-	Directory pointed by OLDPWD
.	Current directory
..	Parent directory
/	Root directory
~/.profile	A shell-independent initialization file, not preferred.
~/.bash_profile	Configure environment and preferences for login shell
~/.bashrc	Configure environment and preferences for interactive shell
~/.bash_login	Bash script to execute on login
~/.bash_logout	Bash script to execute before logout
~/.bash_history	Bash command history

QUOTING	
'string'	Represents a string exactly as is.
"string"	Represents a string exactly, except substitution and escaping
$*var*	Replace by value of *var*
${*var*}	Sometimes you need this for replacing by value of *var*
$(*expr*)	Evaluate *expr* and substitute the result
`` `expr` ``	Evaluate *expr* and substitute the result
$[*arithmetic-expr*]	Evaluate arithmetic-*expr* and substitute value

DATA STRUCTURES	
declare -r *var=val*	Declare and initialize a read-only *var*.
readonly *var=val*	Declare and initialize a read-only *var*.
declare -x *var=val*	Declare and initialize an exported *var*.
export *var=val*	Declare and initialize an exported *var*.
declare -i *var*	Declare a numeric *var*.
declare -p *var*	Print the declaration of *var*.
declare -p	Print all vars.
declare -xp	Print all exported vars.
export	Print all exported vars.
declare -xr	Print all read only vars.
readonly	Print all read only vars.
declare -a *arr*=(*val1 val2*)	Declare and initialize an array named *arr*.
${*arr*[*idx*]}	Access an element by *idx*.
${*arr*[*]}	List all elements.
${!*arr*[*]}	List all indexes that are set.
arr[*idx*]=*val*	Add or overwrite an element by *idx*
unset *arr*[*idx*]	Delete an element by *idx*.
unset *arr*	Delete the array.
declare -A *map*=([*key*]=*val*)	Declare and initialize a hash table named *var*.
${*map*[*key*]}	Access an element by *key*.
${*map*[*]}	Access all values.
${!*map*[*]}	Access all keys.
map[*key*]=*val*	Add or overwrite an entry by *key*.
unset *map*[*key*]	Delete an entry by *key*.
unset *map*	Delete the *map*.

IO REDIRECTION	
cmd > *file*	Write stdout to *file.*
cmd >> *file*	Append stdout to *file.*
cmd | tee *file*	Duplicate and write stdout to *file.*
cmd | tee -a *file*	Duplicate and append stdout to *file.*
cmd 2>&1	Redirect stderr to stdout.
cmd1 | *cmd2*	Pipe output of *cmd1* to *cmd2*.
cmd < *file*	Read *file* as stdin.
cmd << *eof-str* text *eof-str*	Use multiline text as stdin, terminated by the specific sequence *eof-str*.
cmd <<< *str*	Use string as stdin.

CONDITIONS	
[! *expr*]	True if *expr* is false.
[(*expr*)]	Overriding precedence with bracket.
[*expr1* -a *expr2*]	True if both *expr1* and *expr2* are true.
[*expr1* -o *expr2*]	True if either *expr1* and *expr2* are true.

CONDITIONS (LEXICOGRAPHIC)	
[-z *str*]	*str* is zero length.
test -z *str*	*str* is zero length, test command works for all other *cond*itions too.
[-n *str*]	*str* is non-zero length.
[*str1* = *str2*]	*Str1* is the same as *str2*.
[*str1* \> *str2*]	*Str1* sorts lexicographically after *str2*.
[*str1* \< *str2*]	*Str1* sorts lexicographically before *str2*.

CONDITIONS (ARITHMETIC)	
[*arg1* -eq *arg2*]	*Arg1* is equal to *arg2*.
[*arg1* -ne *arg2*]	*Arg1* is not equal to *arg2*.
[*arg1* -lt *arg2*]	*Arg1* is less than to *arg2*.
[*arg1* -le *arg2*]	*Arg1* is less than or equal to *arg2*.
[*arg1* -gt *arg2*]	*Arg1* is greater than to *arg2*.
[*arg1* -ge *arg2*]	*Arg1* is greater than or equal to *arg2*.

CONDITIONS (*FILE* ATTRIBUTES)	
[-a *file*]	*File* exists.
[-e *file*]	*File* exists.
[-d *file*]	*File* is directory.
[-f *file*]	*File* is regular *file.*
[-h *file*]	*File* is symbolic link.
[-s *file*]	*File* size greater than 0.
[-r *file*]	*File* can be read.
[-w *file*]	*File* can be written.
[-x *file*]	*File* can be executed.
[-O *file*]	*File* is owned by effective user.
[-G *file*]	*File* is owned by effective group.
[*file1* -nt *file2*]	*File1* is newer than *file2.*
[*file1* -ot *file2*]	*File1* is older than *file2.*

USEFUL, POWERFUL (NOT NOW)
patch
vi
sed
awk
expect

CONTROL FLOW	
if *cond*; **then** *cmds*; **fi**	If-then statement, fi means "end if".
if *cond* **then** *cmds*; **fi**	If-then statement, new-line instead of ';' is also a valid syntax.
if *cond*; **then** *cmds1*; **else** *cmds2*; **fi**	If-then-else statement, fi means "end if".
if *cond*1; **then** *cmds1*; **elif** *cond*2; **then** *cmds2*; **else** *cmds3*; **fi**	If-then-elseif-else statement, elif means "else if", more than 1 elif is also valid.
case $*var* **in 1)** *cmds1*;; **2)** *cmds2*;; **esac**	switch-case statement based on value contained in *var*.
case `*expr*` **in 1)** *cmds1*;; **2)** *cmds2*;; **esac**	switch-case statement with value evaluated from *expr*.
case $*var* **in 1)** *cmds1*;; ***)** *default-cmd*;; **esac**	switch-case statement with default case handled by *).
for *var* **in** list; **do** *cmds*; **done**	For every element in list, execute *cmds* with *var* set to the element.
for *var* **in** `*expr*`; **do** *cmds*; **done**	For loop with list value evaluated from *expr*.
for *var* **in** list; **do** *cmds*; **break**; **done**	For loop with break.
for *var* **in** list; **do** *cmds*; **continue**; **done**	For loop with continue.
while *cond*; **do** *cmds*; **done**	Execute *cmds* while *cond* is true.
while *cond*; **do** *cmds*; **continue**; **done**	While loop with continue.
while *cond*; **do** *cmds*; **break**; **done**	While loop with break.
until *cond*; **do** *cmds*; **done**	Execute *cmds* while *cond* is false. Break and continue also applies.
while *cond*1; **do while** *cond*2; **do** *cmds*; **continue** 2; **done**; **done**	Continue on outer nested loop
while *cond*1; **do while** *cond*2; **do** *cmds*; **break** 2; **done**; **done**	Break outer nested loop

BASH SCRIPTING	

#!/bin/bash
#This is comment, below is your script
foo(){
  local x=1;
  # This function is not implemented
  return $x;
}
exit `foo`
	Sample bash script

exit *val*	Specify a return code for a bash script, default return 0 if omitted.
$?	Return code from last command
$0	Script name
$N	The N-th argument, only work for N=1-9
${N}	The N-th argument, this form must be used for N>9
$#	Number of argument
shift N	Discard the first N arguments and shift the remaining argument, $0 is not affected.
function *func* { *cmds*; }	Declare a function named func containing *cmds*.
*func***()** { *cmds*; }	Function declaration without using keyword function.
local *var=val*	Declare and initialize function-scoped *var*.
return *val*	Specify a return code for a function, default return 0 if omitted.