Ubuntu 16.04 - commands

command	description	rwx
	main directories	
/	root directory	
/bin	binary files forming the commands and shells used by the system administrator and users	
/boot	files used during the initial boot-up process including the kernel	
/dev	device files for connected hardware	
/etc	system configuration files	
/home	individual directories owned by each user	
/lib	shared libraries needed to boot the system and run the commands in the root filesystem (i.e. commands in /bin and /sbin)	
/lost+found	recovered files that were corrupted by power failures or system crashes	
/mnt	mount points for floppies, cds, or other file systems	
/opt	add-on software packages and/or commercial applications	
/proc	kernel level process information	
/root	home directory for the root user	
/sbin	system administration commands reserved for the superuser (root)	
/tmp	temporary files that are deleted when the system is rebooted or started	
/usr	program files and related files for use by all users	
/var	log files, print spool files, and mail queues	
	joker keys	
*	represents 0 or more characters	
?	represents exactly 1 character	
[__]	represents 1 character within the brackets e.g. [abcdefg]; [a-g]; file[1-4] (file1, file2, file3, file4); file[1\-4] (file1, file-, file4); NOT: file[4-1]	
[!__]	represents 1 character except those within the brackets e.g. [!qQ]	
{__,__,__} {__..__}	creates strings e.g. touch file{{A..C},{1..3},Franz,Sepp} ... fileA, fileB, fileC, file1, file2, file3, fileFranz, fileSepp	
\	compensates replacement function of the following character	
Tabulator	completes name shows list of possible names	
	general	
__ < __	input redirection e.g. tr -s " " < test	
__ > __	output redirection e.g. sort -n nominations.txt | cut -d";" -f1,4-5 > sortetNominations (creates file sortedN...)	
__ >> __	append __ to __	
clear	clears output screen (changes view, doesn't delete)	-

`exit`	exit console	-
`file`	dertermine file type	
`history`	history of entered commands	-
`man __`	show manual for command __ (x on directory and parents)	x
`sudo __`	executes command __ as superuser (password required)	-
`which __`	show where command __ resides on path	-
`__ | __`	pipeline ... uses output of __ as input for __	

	files and directories	
`cat __`	show content of __ (x on directory of target file + r on target file)	(x +) r
`cd`	change directory (changes directory to home directory without arguments) (x on target)	x
`.`	current directory	x
`..`	parent domain	x
`../..`	parent of parent	x
`/`	root directory	x
`~`	home directory	x
`cp __ __ __`	copy files or directories __ __ to __ (r on sources + wx on target directory) e.g. cp file1 file2 ... copies file1 to file2 (creates or overwrites file2) e.g. cp file1 file2 dir1 ... copy file1 and file2 into dir1 e.g. cp file[12] dir1/ ... copies file1 and file2 to dir1 (if they exist, otherwise only 1 file or error) e.g. cp file1 dir1/file2 ... copy file 1 to dir1 and rename it to file2	r + wx
`-i`	interactive ... ask before overwrite (overwrites -n)	r+wx
`-n`	no-clobber ... do not overwrite existing files	r+wx
`-R, -r`	rekursiv (for directories) e.g. cp dir1 dir2 ... copies dir1 into dir2 (dir1\dir2)	r+wx
`du`	estimate file size usage (in blocks of 1024 Bytes) (x on . + r on target)	(x +) r
`-h`	human-readable ... prints more readable with units: __K, __M, __G, ...	(x +) r
`--si`	like -h but with blocksize 1000 Bytes instead of 1024 Bytes	(x +) r
`less __`	show content of __ page per page (exit with q) (x on directory of target file + r on target file)	(x +) r
`ls`	list information about files (default current directory) (r on directory and x on directory if more informations (e.g. -l) are to be displayed)	r (+ x)
`-a`	list all ... also files starting with . (hidden files)	r (+ x)
`-d`	list directories themselves, not their content e.g. ls -d */	r (+ x)
`-l`	long listing format	r (+ x)
`-la`	list all in long listing format	r (+ x)
`-lR`	long listing format of directory and subdirectories	r (+ x)
`-r`	reverse order while sorting	r (+ x)
`-R`	recursive ... also list information about subdirectories	r (+ x)
`-s`	sort by file size (largest first)	r (+ x)
`-t`	sort by modification time (newest first)	r (+ x)

`mkdir`	make directory (wx on parent directory) e.g. mkdir {a,b,c,d}	wx
`-p` `--parents`	no error if exiting, make parent directories as needed e.g. mkdir -p dir1/dir2/dir3 e.g. mkdir -p {a,b,c,d}/{voz,kek,lel}	wx
`mv __ __`	rename or move files or directories (overwrites existing); source __ to target __ (wx on source directory + wx on target directory + - on file / w on directory) e.g. mv file1 file2 (rename file1 to file2); mv dir1 dir2 (rename dir1 to dir2) e.g. mv file1 file2 dir1 (move file1 and file2 to dir1) e.g. mv file1 dir1/filea (move file1 to dir1 and rename it there to filea)	wx + wx + -/w
`-i`	interactive ... prompt before overwrite	wx + wx + -/w
`-t __ __`	target-directory __ sources __	
`pwd`	show path of current directory	-
`touch`	create file (wx on parent directory) e.g. touch folder/file{,1,2,{3..7}}	wx
`rm`	remove file (wx on directory of removed item)	wx
`-r, -R`	rekursiv ... removes directory und subdirectories	wx
`rmdir`	remove directory (only empty directories) (wx on parent of target)	wx

	simple and useful	
`bc`	binary calculator (exit with quit)	-
`cal`	show calendar	-
`date`	show time and date	-
`echo __`	display __ (print string)	-

	system information	
`hostname`	show name of computer	-
`id`	show UID, GID and groups	-
`ps`	show process information	-
`uname`	show name of operating system	-
`-a`	all ... prints additional informations	-
`tty`	show terminal device being used for session	-

	user management	
`/etc/group`	textfile: groupname : password : groupID : group member list	r
`/etc/passwd`	textfile: contains 1 line per user username : password : userID : groupID : comments : home directory : login shell e.g. bin:x:2:2:bin:/bin:/usr/sbin/nologin ... system user (not allowed to log in) note: system user have userIDs 100 to 999; normal users have userIDs 1000 to 29999	r
`/etc/shadow`	textfile: contains passwords (encrypted)	sudo
`/etc/sudoers`	textfile: ONLY CHANGE WITH visudo (contains syntax check) OR usermod -aG sudo __ 1 wrong character might result in inaccesability of operating system	sudo
`addgroup`	adds a new group to the system	sudo

`adduser`	adds a new (normal) user to the system	sudo
`--force-badname`	allows characters in username which contains characters unacaptable by the NAME_REGEX file	sudo
`--system`	adds a system user instead of an normal user	sudo
`chgrp __ __`	changes group to __ of file/directory list __ e.g. sudo chgrp lolgroup dir112 dir113	sudo
`-R`	recursive ... changes group for all files, directories and subdirectories of __	sudo
`chmod`	modifies permits of files/directories; format: chmod [ugoa][+-=]perms; perms = 0-6x [rwxXst] (x on parent directory) e.g. chmod u+rw,g=r,o-wx file1 e.g. chmod 752 dir1 (r=4, w=2, x=1 --> drwxr-x-w-) e.g. chmod 1666 file1 (-rw-rw-rwT ... Sticky Bit gesetzt --> only owner change/delete) e.g. chmod u=rws,g=rw,o=r file2 (-rwsrw-r-- ... SUID gesetzt --> all like owner) e.g. chmod u=rw,g=rwS,o= file3 (-rw-rwS--- ... SGUI gesetzt --> all like group members) e.g. chmod 2712 (4??? ... set SUID; 2??? ... set SGID; 1??? ... set Sticky Bit) note: only owner and superuser can change rights on file/directory	x
`+t __`	sets Sticky Bit of __ (usually only used on directories)	x
`-R __`	recursive ... sets permissions for all files, directories and subdirectories of __	x
`chown __` `chown __:__`	changes owner to __ of file/directory list __ changes owner:group to __:__ of file/directory list __ e.g. sudo chown loluser dir112 dir113 e.g. sudo chown loluser:lolgroup file1	sudo
`-R`	recursive ... changes owner for all files, directories and subdirectories of __	sudo
`delgroup`	deletes group	sudo
`deluser`	delete user from the system (home directory of user stays)	sudo
`--remove-home`	also deletes the users home directory	sudo
`__ sudo`	deletes user __ from sudo-group (/etc/sudoers)	sudo
`passwd`	change your own password	-
`root`	set root password	sudo
`-l root`	lock root account	sudo
`-u root`	unlock root account	sudo
`__`	change password of user __	sudo
`umask`	manages default permissions of new created file/directory (all rights - umask) 1...execute; 2...write; 3...wx; 4...read; 5...rx; 6...rw; 7...rwx e.g. umask; umask 700; umask u=rwx; umask o-x	-
`usermod`	modifies a user account	sudo
`-aG __ __`	group __ is extended by user __	sudo
`-aG __ sudo`	adds user __ to sudo-group (/etc/sudoers)	sudo
`-g __ __`	changes primary group of user __ to group __	sudo
`-l __ __`	changes username to __ from __ (user has to be logged out)	sudo
`who`	show all logged in users	-
`whoami`	show my user	-

	network	
`ping __`	for IPv4 addresses	-
`ping6 __`	for IPv6 addresses	-
`visudo`	for changes in /etc/sudoers (CRITICAL FILE)	sudo

	text processing	
`cut __`	cuts out coloumns/fields/... (r on file) e.g. wc -l euro16.txt | cut -d" " -f1	r
`-c__`	returns the characters __ of each line e.g. cut -c3-7 file.txt	r
`-d"__"`	use delimiter __ instead of TAB e.g. cut -d":" -f1,4 file	r
`-f__`	returns the fields __ (standard delimiter is TAB) e.g. cut -f2,4 file.txt	r
`dos2unix`	converts plain text files from dos to unix format (line seperators)	
`head __`	prints the first 10 lines of __	
`-__l __`	prints the first __ lines of __ e.g. head -3l ranking.txt	
`nano __`	opens file __ in text editor nano (r to open, w to save changes under same file name)	r (+ w)
`od`	dump files in octal or other formats	
`-c`	output format: printable characters or backslash escapes	
`sort __`	sorts __ by character value (goes to stdOut)	
`-k`	select field by which data is sorted e.g. ls -la | sort -k 5n ... sorts all files/directories by size (numerical) e.g. sort -k 3.4 ... sort by 4th character in 3rd field	
`-n`	sort by numerical string value	
`-r`	--reverse ... reverse the result	
`-t"__"`	sets field seperator to __	
`tail __`	prints the last 10 lines of __	
`-__l __`	prints the last __ lines of __ e.g. tail -5l ranking	
`tr`	translates or deletes characters (trimm)	
`-s`	reduces multiple ocurrences of certain characters (mostly used for whitespace) to one e.g. tr -s ´ ´ < file e.g. sort -k2 file | tr -s " "	
`uniq`	reduzes identical sequencial lines to one (usually for sorted files)	
`-c`	--count ... prefix lines by number of ocurrences	
`unix2dos`	converts plain text files from unix to dos format	
`wc __`	counts and prints lines / words / bytes of __	
`-c`	returns Bytes	
`-l`	returns lines	
`-m`	returns characters	
`-L`	returns length of longest line	
`-w`	returns words	

meta character	Regular Expressions
`\`	subsequent meta character looses it's meta function
`(__)`	Gruppierung von Unterausdrücken; e.g. Freitag(e|en)?
`|`	Alternation; e.g. (mon|satur)days?
Anker	
`^`	start of line
`$`	end of line

Quantifikatoren	
?	zero or one occurences
+	one or more occurences
*	zero or more occurences
{__}	exactly __ occurences
{__,}	minimum of __ occurences
{__,__}	at least __ but not more than __ occurences; e.g. [A-Z][a-z]{0,5}
Zeichenklassen	
.	represents exactly one character
[__]	represents 1 character from within the brackets; e.g. b[aei]ll; e.g. [A-ZÄÖÜa-zäöü] e.g. [-a-d] ... represents "-" (hyphen) or the characters "a" to "d"
[^__]	represents 1 character except those within the brackets; e.g. b[^i]ll

Klasse	Regular Expressions
[:alpha:]	characters [:lower:] and [:upper:]; not only latin characters;
[:alnum:]	numbers and characters
[:blank:]	blanks and tabulators
[:cntrl:]	control characters (non printable characters)
[:digit:]	numbers; equal to [0-9]
[:graph:]	printable characters without [:blank:]
[:lower:]	lowercase characters
[:upper:]	uppercase characters
[:print:]	printable characters; equal to [:alnum:], [:punct:] and [:blank:]
[:punct:]	punctuation characters and others: , - . ! " # $ % & ' ( ) { } [ ] * + / : ; < = > ? @ \ ^ _ ` | ~
[:space:]	blanks, tabulators, return, form feed, carriage return, etc.
[:xdigit:]	hexadecimal numbers; equal to [0-9A-fa-f]

	vordefinierte Zeichenklassen - https://regexr.com/
/__/g	global
/__/i	case insensitive
Perl	
\d	any digit
\D	anything but digits
\s	space " "
\S	anything but space
\w	any alphanumeric (letter or digit) character or underscore
\W	anything but alphanumeric characters or underscore

	grep
grep	outputs all lines of textfile that match regexp
	uses BRE (basic regular expressions); ? + { } | ( ) have to be used with \ (e.g. \? \+ ...) syntax: grep [options] regex [files]
	e.g. grep [[:digit]]* anydata e.g. grep 'Wasserf[aä]lle\?' anydata
-E	extended grep: enables the use of ? + { } | ( ) without \ e.g. grep -E 'fridays?' anydata e.g. grep -ioE 'Wasserf[aä]lle?' anydata
-F	fixed string: simplified version which only allows strings (no regex)
-P	perl: interprets regex as Perl-compatible regex e.g. grep -P '\d' anydata
-V	version: outputs the version number of grep and exits
-c	count: only returns the number of lines containing match
-h	hide: suppresses the output of file names
-i	ignore: no case distinction
-l	list: only returns the names of files with lines containing match
-n	number: prepends the line number within input file of each matching line
-o	only: prints only the matched parts of matching line, each in matched part in a new line
-s	supress: suppresses error messages for missing files
-v	vice versa: outputs all lines that don't match the regexp
	sed
sed	non-interactive editor for text files; main feature: searching and replacing regexp doesn't save changes to file but displays changes on standard output
	syntax: sed [options] 'ADDRESSs/regexp/replacement/FLAGs' filename syntax: sed [options] 'PATTERNs/regexp/replacement/FLAGs' filename
	- Trennzeichen / kann bei Bedarf durch anderes ersetzt werden - Ausgabe auf Bildschirm (Ausgabeumleitung möglich)
	greedy: file with lines: < ..... > sed –r 's/<.*>//g' myhtml // deletes content of all lines sed –r 's/<[^>]*>//g' myhtml // only deletes content of 1 line
	e.g. sed -r 's/Linux/Linux-Unix/' mydata
[options]	
-r	regexp-extended: use extended regular expressions
ADDRESS	e.g. 2 -- sed -r '2 s/Linux/Linux-Unix/g' mydata -- suche in Zeile 2 e.g. 1,4 -- sed -r '1,4 s/Linux/Linux-Unix/g' mydata -- suche in Zeilen 1 bis 4
PATTERN	as regexp: only searches lines containing regex; regexp within "/" e.g. sed -r '/ich/s/€1500/€5000/g' Gehalt.dat
s	
s/regexp/replacement	substitute: replaces first occurance of rexexp by replacement
	seperator can be changed e.g. sed 's$06/Nov/1997$6.11.1997$g' file > newfile e.g. sed 'ss06/Nov/1997s6.11.1997sg' file > newfile e.g. sed 's;06/Nov/1997;6.11.1997;g' file > newfile

FLAGs	
d	delete pattern space; start next cycle e.g. sed '/^$/d' anyfile // deletes all empty lines of anyfile e.g. sed '1d' anyfile // deletes the first line of anyfile e.g. sed '2,7d' anyfile // deletes lines 2 to 7 of anyfile e.g. sed '$d' anyfile // deletes last line of anyfile
g	replaces all matches with regexp with replacement (otherwise only first) e.g. ANEW=`echo $A | sed 's/ö/oe/g`
i	ignore: no case distinction
p	print: prints changed lines
'number'	replaces the number'th occurance of regexp

	Windows | Unix
file	determine file type e.g. file txtfile // output: "txtfile: ISO-8859 text, with CRLF line terminators"
-i	mime: mimes type string rather than converting them to more human readable ones e.g. file -i txtfile // output "txtfile: text/plain; charset=iso-8859-1"
iconv	converts text from one character encoding to another e.g. iconv -f iso-8859-1 -t utf-8 filename > filename_con
-f	from-encoding
-t	to-encoding
od	dump file in octal and other formats
-c	select printable characters or backslash escapes

	Wildcards
?	stands for 1 character
*	stands for 0, 1 or more characters; also: Wildcard for all files that don't start with a . e.g. xyz* // sting starting with xyz e.g. .* // all hidden files (starting with a dot)
[ ]	stands for 1 character within the brackets e.g. b[ei]ll // = bell or bill
[!]	stands for 1 character except those within brackets e.g. [!x] // any character but x e.g. [!abc] // any character but a, b or c
[^]	stands for 1 character except those within brackets e.g. [^x] // any character but x

	Shell Script
#!/bin/sh	Shebang Line: instructs the OS to interpret the script with /bin/sh
#	starts a comment till the end of the line
in|out|err|..	
0	stdin: standard input
1	stdout: standard output
2	stderr: standard error output
<	input redirection (numerical value: 0)
>	output redirection (overwrites file) e.g. 2> /dev/null

>>	output redirection (appends to file)
/dev/null	void; outputs to /dev/null are never seen again
Globbing	for f in *.txt do mv ./"$f" "${f%txt}bat" done
global var	
HOME	home directory: e.g. echo $HOME /home/user
HOSTNAME	name of machine
HOSTTYPE	type of machine (processor)
IFS	internal field separator
PATH	list of paths which are checked for commands (seperated by ':') e.g. PATH=$PATH:/home/user/progdir e.g. PATH=$PATH:. // adds current folder to local PATH variable (only current shell) export PATH // exports local variable PATH, making it global (all shells)
PWD	current path: e.g. echo $PWD /home/user/userdir/anotherdir/currentdir
USER	username (loginname)
apostrophes	
"__"	no replacement of Wildcards (*, $, [ ]); shell variables ($VAR) get replaced; e.g. echo "Really? I thought it was $NAME."
´__´	suppresses all replacement
`__`	command __ is executed an the result returned e.g. SIZE=`ls -l file | cut -d" " -f5`
basics	
=	variable assignment e.g. name=wert e.g. msg="Hello world."
` `	statements within ` are executed and the output can be assigned to variables e.g. ANZ=`echo $X | grep -ioE "Wasserfall" | wc -l`
$__	accesses __ (returns emtpy string '' if __ is not set)
export	exports variable from current shell and makes it accessable for other shells e.g. export name
set	lists all set variables
special param	
$*	expands to the arguments "$*" == "$1 $2 $3 ..." // separated by first character of $IFS
$@	expands to the arguments e.g. echo "My parameters are: $@ "
$#	expands to the number of arguments e.g. echo "I was called with $# parameters."
$?	expands to the return value of the last executed visible command command successful: $? = 0 command not successful: $? > 0 e.g. echo "exit code of last command: $?"
$$	expands the prozess ID of the shell e.g. echo "My internal process ID is $$"
$0	expands the name of the Shell of Shell Skript e.g. echo "My name is $0 ."
$1 $2 ...	first parameter second parameter ... e.g. echo "My first parameter is $1 ." e.g. echo "My second parameter is $2 ."

input	
read	reads value from stdin and assigns it to variable // separator for input is the IFS syntax: read variable {variable} more variables than values -> remaining variables get empty string assigned more values than variables -> last variable gets rest of values
comparisons	
[__]	comparison __ returns 0 or 1 0 ... true 1 ... false
test __	comparison __ returns 0 or 1 0 ... true 1 ... false
logic gates	
! __	NOT __
__ -a __	__ AND __
__ -o __	__ OR __
\(__\)	brackets for grouping ligic expression __ e.g. if [\(*condition1* -a *condition2* \) -o *condition3*]; then ...
c: numbers	
__ -eg __	equals e.g. if [$# -eg 0] then echo "No arguments defined." e.g. if test $# -eg 0 then echo "No arguments defined."
__ -le __	less or equal than
__ -lt __	less than
__ -ne __	not equals
__ -ge __	greater or equal than
__ -gt __	greater than
c: files	
-d __	__ exists & directory?
-e __	__ exists?
-f __	__ exists & regular file?
-r __	__ exists & read permission granted?
-s __	__ exists & size > 0?
-w __	__ exists & write permission granted?
-x __	__ exists & execute permission granted?
c: files	
__ -ef __	__ and __ have same inode number (hardlinks)
__ -nt __	__ newer than __ ?
__ -ot __	__ older than __ ?
c: strings	
-n __	string is not empty? (string length is 0)
-z __	string is empty? (string length is not 0)
__ = __	strings __ and __ are equal?
__ != __	strings __ and __ are not equal?
IF	
	if __ then fi
	if __ then else fi
	if __ then elif __ then else fi

FOR	
	for *selector* in *list* do *commands* done
	e.g. // change file ending from .txt to .bat for f in *.txt do mv ./"$f" "${f%txt}bat" done e.g. // write each word of anyfile in a seperate line for WORD in `cat *anyfile*` do echo $WORD done e.g. // counting from 1 to 10 for i in {1..} do echo "i is now $i" done
	for *selector* do *commands* done
	// if no list is defined
WHILE	
	while *condition* do *commands* done
UNTIL	
	until *condition* do *commands* done
execute if	
__ && __	only when __ is executed successful ($?=0) __ is executed
__ || __	only when __ is executed unsuccessful ($?>0) __ is executed
	e.g. test -d VAR && echo "$VAR is dir" \|\| echo "$VAR no dir"
calculate INT	
$((__))	e.g. echo $((9/3*10)) // 30 e.g. X=$((3+5+8/4)) // 10
calculat REAL	
"__" | bc	basic calculator e.g. echo "(3+5+8)/4" // 4 e.g. echo "3+5+8/4" // 10
"scale=__;__" | bc	returns result with __ decimals for calculation __ (default: 0 decimals) e.g. echo "scale=5; 10000/3" | bc // 3333.33333 e.g. echo "scale=-2; 10000/3" | bc // 3333 e.g. RESULT=`echo "70/3" | bc // $RESULT=23