



Useful Unix Commands Cheat Sheet

The Chinese University of Hong Kong

SIGSC Training (Fall 2016)

FILE AND DIRECTORY	
pwd	Return path to current directory.
ls	List directories and <i>files</i> here.
ls dir	List directories and <i>files</i> in a directory.
ls -d */	List the name of all subdirectory.
ls -a	List all <i>files</i> including hidden <i>files</i> .
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 <i>dir</i> .
popd dir	Return (pop from the stack) to last saved directory.
dirs	List all saved directories (the stack).
touch file	Create an empty <i>file</i> .
mkdir dir	Create an empty directory.
ln -s file	Create a soft link here to <i>file/dir</i>
mv dir1 dir2	Move or renames a <i>file</i> .
cp file1 file2	Copy <i>file1</i> to <i>file2</i> .
cp -r dir1 dir2	Copy <i>dir1</i> to <i>dir2</i> including subdirectories.
rmdir dir	Remove an empty directory.
rm file	Remove a <i>file</i> .
rm -r dir	Remove a directory and its subdirectories and <i>files</i> .
rm -f file	Remove a <i>file</i> , suppress all warning.
find dir -name pattern	Search for <i>file</i> name matching pattern in <i>dir</i> .
md5sum file	Calculate MD5 checksum of <i>file</i> , there are other algorithms too, e.g. various 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 <i>file</i> .
chmod -x file	Unset execute permission to <i>file</i> .
chmod IJK file	Set permission denoted by IJK to <i>file</i> . 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 <i>dir</i> and all subdirectories and <i>files</i> .
chown -R user:group dir	Change the ownership of <i>dir</i> and all subdirectories and <i>files</i> .

COMPRESSION	
tar -cf file.tar dir	Group <i>files</i> .
tar -xf file.tar	Ungroup <i>files</i> .
tar -zcf file.tar.gz dir	Group and compress <i>files</i> .
tar -zxf file.tar.gz	Extract and ungroup <i>files</i> .

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

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

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

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

MISC	
echo <i>string</i>	Print the string to STDOUT
printf <i>format-str args</i>	C like printf
date	Display current date time information.
time <i>cmd</i>	Time the execution of <i>cmd</i>
sleep <i>N</i>	Wait for <i>N</i> secs.
watch <i>cmd</i>	Repeatedly execute <i>cmd</i> every 2s and display result.
which <i>cmd</i>	Display the resolved command directory.
seq <i>a b incr</i>	Generate a list of number starting from <i>a</i> to <i>b</i> incremented by <i>incr</i> .
yes	Keep saying yes
yes <i>str</i>	Keep saying <i>str</i> , 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	
<code>cmd > file</code>	Write stdout to <i>file</i> .
<code>cmd >> file</code>	Append stdout to <i>file</i> .
<code>cmd tee file</code>	Duplicate and write stdout to <i>file</i> .
<code>cmd tee -a file</code>	Duplicate and append stdout to <i>file</i> .
<code>cmd 2>&1</code>	Redirect stderr to stdout.
<code>cmd1 cmd2</code>	Pipe output of <i>cmd1</i> to <i>cmd2</i> .
<code>cmd < file</code>	Read <i>file</i> as stdin.
<code>cmd << eof-str</code> text <code>eof-str</code>	Use multiline text as stdin, terminated by the specific sequence <i>eof-str</i> .
<code>cmd <<< str</code>	Use string as stdin.

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

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

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

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

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

CONTROL FLOW

if <i>cond</i> ; then <i>cmds</i> ; fi	If-then statement, fi means "end if".
if <i>cond</i> then <i>cmds</i> ; fi	If-then statement, new-line instead of ';' is also a valid syntax.
if <i>cond</i> ; then <i>cmds1</i> ; else <i>cmds2</i> ; fi	If-then-else statement, fi means "end if".
if <i>cond1</i> ; then <i>cmds1</i> ; elif <i>cond2</i> ; then <i>cmds2</i> ; else <i>cmds3</i> ; fi	If-then-elseif-else statement, elif means "else if", more than 1 elif is also valid.
case <i>\$var</i> in 1) <i>cmds1</i> ::; 2) <i>cmds2</i> ::; esac	switch-case statement based on value contained in <i>var</i> .
case <i>`expr`</i> in 1) <i>cmds1</i> ::; 2) <i>cmds2</i> ::; esac	switch-case statement with value evaluated from <i>expr</i> .
case <i>\$var</i> in 1) <i>cmds1</i> ::; *) <i>default-cmd</i> ::; esac	switch-case statement with default case handled by *).
for <i>var</i> in list; do <i>cmds</i> ; done	For every element in list, execute <i>cmds</i> with <i>var</i> set to the element.
for <i>var</i> in <i>`expr`</i> ; do <i>cmds</i> ; done	For loop with list value evaluated from <i>expr</i> .
for <i>var</i> in list; do <i>cmds</i> ; break ; done	For loop with break.
for <i>var</i> in list; do <i>cmds</i> ; continue ; done	For loop with continue.
while <i>cond</i> ; do <i>cmds</i> ; done	Execute <i>cmds</i> while <i>cond</i> is true.
while <i>cond</i> ; do <i>cmds</i> ; continue ; done	While loop with continue.
while <i>cond</i> ; do <i>cmds</i> ; break ; done	While loop with break.
until <i>cond</i> ; do <i>cmds</i> ; done	Execute <i>cmds</i> while <i>cond</i> is false. Break and continue also applies.
while <i>cond1</i> ; do while <i>cond2</i> ; do <i>cmds</i> ; continue 2; done ; done	Continue on outer nested loop
while <i>cond1</i> ; do while <i>cond2</i> ; do <i>cmds</i> ; 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*; }

func() { *cmds*; }

local *var=val* Declare and initialize function-scoped *var*.

return *val* Specify a return code for a function, default return 0 if omitted.