

## Useful Unix Commands Cheat Sheet

The Chinese University of Hong Kong

SIGSC Training (Fall 2016)

F	ILE 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.	
<b>ls</b> -d */	List the name of all subdirectory.	
ls -a	List all files including hidden files.	
ls -lh	List including more data in readable	
15 -111	format.	
cd dir	Change directory.	
cd	Go to home directory.	
pushd dir	Save current directory (push onto	
	stack) and go to <i>dir</i> .	
nond dir	Return (pop from the stack) to last	
popd dir	saved directory.	
dirs	List all saved directories (the stack).	
touch file	Create an empty <i>file</i> .	
mkdir dir	Create an empty directory.	
<b>ln</b> -s file	Create a soft link here to <i>file/dir</i>	
mv dir1 dir2	Move or renames a <i>file</i> .	
<b>cp</b> file1 file2	Copy <i>file1</i> to <i>file2</i> .	
<b>cp</b> -r <i>dir1 dir2</i>	Copy dir1 to dir2 including	
<b>Cp</b> -1 <i>utr1 utr2</i>	subdirectories.	
<b>rmdir</b> dir	Remove an empty directory.	
<b>rm</b> file	Remove a <i>file</i> .	
<b>rm</b> -r <i>dir</i>	Remove a directory and its	
1 III -1 <i>atr</i>	subdirectories and <i>files</i> .	
<b>rm</b> -f <i>file</i>	Remove a file, suppress all warning.	
find dir -name	Search for <i>file</i> name matching pattern	
pattern	in <i>dir</i> .	
	Calculate MD5 checksum of <i>file</i> , there	
md5sum file	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		
<b>chmod</b> +x <i>file</i>	Set execute permission to <i>file</i> .	
<b>chmod</b> -x <i>file</i>	Unset execute permission to file.	
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	Set permission denoted by IJK to	
dir	dir and all subdirectories and files.	
chown -R	Change the ownership of <i>dir</i> and all	
user:group dir	subdirectories and <i>files</i> .	

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

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

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

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

JOB CONTROL		
<b>ps</b> aux	Show running processes.	
pkill -u user	Terminate all process for user.	
pkill cmd	Terminate a process with SIGTERM.	
pkill -9 cmd	Terminate <i>cmd</i> with SIGKILL.	
top	View top CPU using processes.	
nohup cmd	Run cmd disregarding the hangup	
	signal.	
cmd &	Run <i>cmd</i> 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	
<b>printf</b> format- str args	C like printf	
date	Display current date time information.	
time cmd	Time the execution of <i>cmd</i>	
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 <i>a</i> to <i>b</i> incremented by <i>incr</i> .	
yes	Keep saying yes	
yes str	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	
1	Root directory	
~/.profile	A shell-independent initialization file,	
~/.prome	not preferred.	
~/.bash_profile	Configure environment and	
~/.basii_prome	preferences for login shell	
~/.bashrc	Configure environment and	
~/.bashrc	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	
sumg	substitution and escaping	
\$var	Replace by value of <i>var</i>	
\${ <i>var</i> }	Sometimes you need this for replacing	
	by value of <i>var</i>	
\$( <i>expr</i> )	Evaluate <i>expr</i> and substitute the result	
`expr`	Evaluate <i>expr</i> and substitute the result	
<i>\$[arithmetic-</i>	Evaluate arithmetic- <i>expr</i> and substitute	
expr]	value	

declare -r var=valDeclare and initialize a read- only var.readonly var=valDeclare and initialize a read- only var.declare -x var=valDeclare and initialize an exported var.export var=valDeclare and initialize an exported var.declare -i varDeclare a numeric var.declare -p varPrint the declaration of var.declare -p varPrint all exported vars.declare -xpPrint all exported vars.declare -xrPrint all read only vars.declare -a arr=(vall)Declare and initialize an array val2)starr[idx]}List all elements.\${arr[idx]}List all elements.starr[idx]=valAdd or overwrite an element by idxunset arr[idx]Declare and initialize a hash map=([key]=val)kapsAccess an element by idx.starp[*]}Access an element by key.starp[*]}Delete an element by idx.unset arr[idx]Delete an element by idx.unset arr[key]Access an element by key.starp[*]Access an element by key.starp[*]Delete an element by key.starp[*]Delete an element by key.unset map[key]Delete an entry by key.unset map[key]Delete an entry by key.	DATA STRUCTURES	
only var.readonly var=valDeclare and initialize a read- only var.declare -x var=valDeclare and initialize an exported var.export var=valDeclare and initialize an exported var.declare -i varDeclare a numeric var.declare -p varPrint the declaration of var.declare -p varPrint all vars.declare -xpPrint all exported vars.declare -xpPrint all exported vars.declare -xpPrint all exported vars.declare -xpPrint all read only vars.declare -xpDeclare and initialize an array part all read only vars.declare -a arr=(val)Declare and initialize an array val2)start[idx]Access an element by idx.s{arr[idx]List all elements.s{arr[idx]Delete an element by idx.unset arr[idx]Delete the array.declare -ADeclare and initialize a hash map=([key]=val)s{map[key]=val}Access an element by idx.s[map[key]=val]Access an element by idx.sex and initialize a hash map=([key]=val)Access an element by idx.s[map[key]=val]Access an element by idx.s[map[key]=val]Access an element by key.s[map[key]=val]Access an element by key.s[map[key]=val]Access an element by key.s[map[key]=val]Access an element by key.s[map[key]=val]Access an element by key.	dooloro rugr-ugl	Declare and initialize a read-
readonly var=valonly var.declare -x var=valDeclare and initialize an exported var.export var=valDeclare and initialize an exported var.declare -i varDeclare a numeric var.declare -p varPrint the declaration of var.declare -p varPrint all vars.declare -xpPrint all exported vars.declare -xpPrint all exported vars.declare -xpPrint all exported vars.declare -xar=(val)Declare and initialize an array val2)karr[idx]}Access an element by idx.s{arr[idx]}List all elements.s{larr[idx]}Delete an element by idx.art[idx]=valDelete an element by idx.unset arr[idx]Delete an element by idx.imap=([key]=val)Access an element by idx.s{map[key]}=valAccess an element by idx.sex arr [idx]Delete an element by idx.imap[key]=valAccess an element by idx.imap[key]=valAccess an element by idx.sex arr[idx]Delete an element by idx.imap[key]=valAccess an element by idx.sex arr[idx]Delete an element by idx.imap[key]=valAccess an element by key.sex and initialize a hash by idximap[key]=valAccess an element by key.sex and [keys]Access an element by key.		only var.
only var.declare -x var=valDeclare and initialize an exported var.export var=valDeclare and initialize an exported var.declare -i varDeclare a numeric var.declare -p varPrint the declaration of var.declare -p varPrint all vars.declare -xpPrint all exported vars.declare -xpPrint all exported vars.declare -xpPrint all exported vars.declare -a arr=(val1)Declare and initialize an array val2)karr[idx]}Access an element by idx.k[arr[*]}List all elements.k[varr[*]}Delete an element by idx.unset arr[idx]Delete an are val.by idxDelete an are set.maned arr.Si ant[idx]k[sarr[*]]Delete an element by idx.k[sarr[*]]Delete an element by idx.imset arrDelete an areay.declare -ADeclare and initialize a hash map=([key]=val)k[map[key]]Access an element by key.k[map[key]]Access an element by key.s[map[key]=valAccess an element by key.s[map[key]=valAccess an element by key.s[map[key]=valAdd or overwrite an entry by key.map[key]=valAdd or overwrite an entry by key.map[key]=valDelete an entry by key.	readonly var-val	Declare and initialize a read-
declare -x var=valexported var.export var=valDeclare and initialize an exported var.declare -i varDeclare a numeric var.declare -p varPrint the declaration of var.declare -pPrint all vars.declare -xpPrint all exported vars.declare -xrPrint all read only vars.declare -a arr=(val)Declare and initialize an array val2)s{arr[idx]}Access an element by idx.\${arr[idx]=valList all indexes that are set.arr[idx]=valDeclare and initialize a hash map=([key]=val)kable named var.\${map[key]=valAccess an element by idx.\${map[key]=valAccess an element by key.anated var.anated var.anated var.beclare -Aand or overwrite an element by idxanated var.beclare -Aand or overwrite an element by idx.anated var.beclare -Abeclare and initialize a hash by idxanap=([key]=val)kable named var.anap[key]Access all values.anap[key]=valAdd or overwrite an entry by key.anap[key]=valAdd or overwrite an entry by key.		only var.
exported var.export var=valDeclare and initialize an exported var.declare -i varDeclare a numeric var.declare -p varPrint the declaration of var.declare -pPrint all vars.declare -xpPrint all exported vars.exportPrint all exported vars.declare -arr=(val)Declare and initialize an array val2)s{arr[idx]}Access an element by idx.\${arr[idx]}List all indexes that are set.arr[idx]=valDelete an element by idx.unset arr[idx]Delete an element by idx.faap=([key]=val)table named var.\${map[key]}Access all keys.saps (key]=valAccess all keys.anap[key]=valAccess all keys.anap[key]=valAccess all keys.anap[key]Delete an entry by key.	declare _v var_val	Declare and initialize an
export var=valexported var.declare -i varDeclare a numeric var.declare -p varPrint the declaration of var.declare -pPrint all vars.declare -xpPrint all exported vars.exportPrint all exported vars.declare -xrPrint all read only vars.readonlyPrint all read only vars.declare - a arr=(val1)Declare and initialize an array val2)s{arr[idx]}Access an element by idx.\${arr[idx]}List all elements.start[idx]Delete an element by idx.unset arr[idx]Delete the array.declare -ADeclare and initialize a hash map=([key]=val)stale named var.\${map[key]}Access all values.stanp[key]=valAdd or overwrite an element by key.unset map[key]Delete an element by key.stanp[key]Access all values.stanp[key]Delete an element by key.		exported var.
exported var.declare -i varDeclare a numeric var.declare -p varPrint the declaration of var.declare -pPrint all vars.declare -xpPrint all exported vars.exportPrint all exported vars.declare -xrPrint all read only vars.declare -a arr=(val1Declare and initialize an arrayval2)named arr.\${arr[idx]}Access an element by idx.\${arr[idx]}Delete an element by idx.starr[idx]=valDelete the array.declare -ADelete the array.declare -ADeclare and initialize a hashmap=([key]=val)table named var.\${map[key]=val}Access all keys.map[key]=valAdd or overwrite an element by key.	export var-val	Declare and initialize an
declare -p varPrint the declaration of var.declare -pPrint all vars.declare -xpPrint all exported vars.exportPrint all exported vars.declare -xrPrint all read only vars.readonlyPrint all read only vars.declare -a arr=(vallDeclare and initialize an array val2)s{arr[idx]}Access an element by idx.\${arr[idx]}List all elements.s{arr[idx]=valAdd or overwrite an element by idxunset arr[idx]Delete the array.declare -ADeclare and initialize a hash map=([key]=val)stalle named var.stanp[key]}Access an element by key.stanp[key]=valAdd or overwrite an element by key.stanp[key]=valDeclare and initialize a hash cars an element by key.stanp[key]=valAccess an element by key.stanp[key]=valAccess an element by key.stanp[key]=valDelete an and place.stanp[key]=valDelete an and place.stanp[key]=valAccess an element by key.stanp[key]=valDelete an and place.		exported var.
declare -pPrint all vars.declare -xpPrint all exported vars.exportPrint all exported vars.declare -xrPrint all read only vars.declare -a arr=(val1Declare and initialize an array val2)s{arr[idx]}Access an element by idx.\${arr[idx]}List all elements.\${arr[idx]}Delete an element by idx.starr[idx]=valDelete an element by idx.unset arr[idx]Delete an element by idx.unset arr[idx]Access an element by idx.fmap=([key]=val)Access an element by idx.stanap[key]Access an element by idx.unset map[key]=valAccess an element by key.wap[key]=valAccess an element by key.by add or overwrite an element by idx.by add or overwrite an element by idx.unset arrDelete the array.declare -ADeclare and initialize a hash map=([key]=val)kable named var.stable named var.stanp[key]Access all values.stanp[key]Access all values.stanp[key]=valDelete an entry by key.	declare -i var	Declare a numeric var.
declare -xpPrint all exported vars.exportPrint all exported vars.declare -xrPrint all read only vars.readonlyPrint all read only vars.declare -a arr=(val1)Declare and initialize an array named arr.val2)named arr.\${arr[idx]}Access an element by idx.\${arr[idx]}List all elements.\${arr[idx]=valAdd or overwrite an element by idxunset arr[idx]Delete an element by idx.inset arr[idx]Delete the array.declare -ADeclare and initialize a hash map=([key]=val)\${map[key]}Access an element by key.\${map[key]=val}Add or overwrite an element by key.unset map[key]Delete an and initialize a hash map[key]=valbeclare and initialize a hash map[key]=valAccess an element by key.\${map[key]Access an element by key.unset map[key]Delete an and initialize a hash by key.	declare -p var	Print the declaration of var.
exportPrint all exported vars.declare -xrPrint all read only vars.readonlyPrint all read only vars.declare -a arr=(val1Declare and initialize an array named arr. $val2$ )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 idxunset arr[idx]Delete an element by idx.unset arr[idx]Delete the array.declare -ADeclare and initialize a hash map=([key]=val) $\{map[key]\}$ Access an element by key. $\{map[*]\}$ Access all values. $map[key]=val$ Add or overwrite an entry by key.unset map[key]Delete an entry by key.	declare -p	Print all vars.
declare -xrPrint all read only vars.readonlyPrint all read only vars.declare -a $arr=(vall)$ Declare and initialize an array named $arr.$ $val2$ )named $arr.$ $s{arr[idx]}$ Access an element by $idx.$ $s{arr[idx]}$ List all elements. $s{arr[*]}$ 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 $arr[idx]$ Delete the array.declare -ADeclare and initialize a hash $map=([key]=val)$ $s\{map[key]\}$ Access an element by $key.$ $s\{map[*]\}$ Access all values. $smap[key]=val$ Add or overwrite an entry by $key.$ unset map[key]Delete an entry by $key.$	declare -xp	Print all exported vars.
readonlyPrint all read only vars.declare -a arr=(vallDeclare and initialize an array named arr. $\langle arr[idx] \rangle$ Access an element by idx. $\langle arr[idx] \rangle$ Access an element by idx. $\langle arr[*] \rangle$ List all elements. $\langle arr[*] \rangle$ List all indexes that are set. $arr[idx]=val$ Add or overwrite an element by idxunset arr[idx]Delete an element by idx.unset arr[idx]Delete the array.declare -ADeclare and initialize a hash map=([key]=val) $\langle map[key] \rangle$ Access an element by key. $\langle map[key] = val$ Access all values. $\langle hdd or overwrite an entry bykey.Add or overwrite an entry bykey.unset map[key]Delete an entry by key.$	export	Print all exported vars.
declare -a $arr=(val1)$ Declare and initialize an array named $arr$ . $val2$ )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 $arr[idx]$ Delete the array.declare -ADeclare and initialize a hash $map=([key]=val)$ $\{map[key]\}$ Access an element by $key$ . $\{map[*]\}$ Access all values. $\{map[key]=val$ Add or overwrite an entry by $key$ .unset $map[key]$ Delete an entry by $key$ .	declare -xr	Print all read only vars.
val2)named arr. $\{arr[idx]\}$ Access an element by idx. $\{arr[idx]\}$ List all elements. $\{arr[*]\}$ List all indexes that are set. $\{rr[idx]=val$ Add or overwrite an element by idxunset arr[idx]Delete an element by idx.unset arrDelete the array.declare -ADeclare and initialize a hash map=([key]=val) $\{map[key]\}$ Access an element by key. $\{map[*]\}$ Access all values. $\{map[key]=val$ Add or overwrite an entry by key.unset map[key]Delete an element by key.	readonly	Print all read only vars.
$arr[idx]$ Access an element by idx. $arr[idx]$ List all elements. $\{!arr[*]\}$ List all indexes that are set. $arr[idx]=val$ Add or overwrite an element by idxunset arr[idx]Delete an element by idx.unset arrDelete the array.declare -ADeclare and initialize a hash map=([key]=val) $\{map[key]\}$ Access an element by key. $\{map[*]\}$ Access all values. $\{map[key]=val$ Add or overwrite an entry by key.unset map[key]Delete an entry by key.	<b>declare</b> -a <i>arr</i> =( <i>val1</i>	Declare and initialize an array
$\{arr[*]\}$ List all elements. $\{\{arr[*]\}\}$ List all indexes that are set. $arr[idx]=val$ Add or overwrite an element by idxunset arr[idx]Delete an element by idx.unset arrDelete the array.declare -ADeclare and initialize a hash map=([key]=val) $\{map[key]\}$ Access an element by key. $\{map[*]\}$ Access all values. $\{map[key]=val$ Add or overwrite an entry by key.unset map[key]Delete an entry by key.	val2)	named arr.
$\{!arr[*]\}$ 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 $arr[idx]$ Delete the array.declare -ADeclare and initialize a hash $map=([key]=val)$ $\{map[key]\}$ Access an element by $key$ . $\{map[*]\}$ Access all values. $\{map[*]\}$ Add or overwrite an entry by $key$ .unset $map[key]$ Delete an entry by $key$ .	${arr[idx]}$	Access an element by <i>idx</i> .
arr[idx]=valAdd or overwrite an element by idxunset arr[idx]Delete an element by idx.unset arrDelete the array.declare -ADeclare and initialize a hash map=([key]=val)kable named var.\${map[key]}Access an element by key.\${map[*]}Access all values.\${map[key]=val}Add or overwrite an entry by key.unset map[key]Delete an entry by key.	\${ <i>arr</i> [*]}	List all elements.
arr[idx]=valby idxunset arr[idx]Delete an element by idx.unset arrDelete the array.declare -ADeclare and initialize a hash $map=([key]=val)$ table named $var$ . $\{map[key]\}$ Access an element by $key$ . $\{map[*]\}$ Access all values. $\{map[*]\}$ Add or overwrite an entry by $key$ .unset map[key]Delete an entry by key.	\${! <i>arr</i> [*]}	List all indexes that are set.
by idxunset arr[idx]Delete an element by idx.unset arrDelete the array.declare -ADeclare and initialize a hash $map=([key]=val)$ table named $var$ . $\{map[key]\}$ Access an element by $key$ . $\{map[*]\}$ Access all values. $\{map[*]\}$ Add or overwrite an entry by $key$ .unset map[key]Delete an entry by key.	arr[idx]_val	Add or overwrite an element
unset arrDelete the array.declare -ADeclare and initialize a hash $map=([key]=val)$ table named $var$ . $\{map[key]\}$ Access an element by $key$ . $\{map[*]\}$ Access all values. $\{map[*]\}$ Access all values. $\{map[*]\}$ Add or overwrite an entry by $key$ .unset map[key]Delete an entry by key.		by <i>idx</i>
declare -ADeclare and initialize a hash $map=([key]=val)$ table named var. $\{map[key]\}$ Access an element by key. $\{map[*]\}$ Access all values. $\{map[*]\}$ 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.	<b>unset</b> <i>arr</i> [ <i>idx</i> ]	Delete an element by <i>idx</i> .
$map=([key]=val)$ 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 arr	Delete the array.
\${map[key]}Access an element by key.\${map[*]}Access all values.\${!map[*]}Access all values.\${!map[*]}Access all keys.map[key]=valAdd or overwrite an entry by key.unset map[key]Delete an entry by key.	declare -A	Declare and initialize a hash
${map[*]}$ 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.	map=([key]=val)	table named var.
$\{map[*]\}$ Access all keys. $map[key]=val$ Add or overwrite an entry by key.unset map[key]Delete an entry by key.	${map[key]}$	Access an element by key.
map[key]=valAdd or overwrite an entry by key.unset $map[key]$ Delete an entry by key.	\${ <i>map</i> [*]}	Access all values.
map[key]=valkey.unset map[key]Delete an entry by key.	\${!map[*]}	Access all keys.
key.unset map[key]Delete an entry by key.	man[kev]-val	Add or overwrite an entry by
	map[ney]=vai	key.
<b>unset</b> <i>map</i> Delete the <i>map</i> .	<b>unset</b> <i>map</i> [ <i>key</i> ]	Delete an entry by key.
	unset map	Delete the <i>map</i> .

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

CONDITIONS

bracket.

[! *expr*]

[( *expr*)]

[expr1 -a expr2]

[ *str1* \< *str2* ]

str2.

True if *expr* is false.

Overriding precedence with

True if both *expr1* and *expr2* are

<b>CONDITIONS (ARITHMETIC)</b>		
eq arg21	Argl is equal to arg2.	

1

[ arg1 -eq arg2 ]	Arg1 is equal to arg2.
[ <i>arg1</i> -ne <i>arg2</i> ]	Arg1 is not equal to arg2.
[ <i>arg1</i> -lt <i>arg2</i> ]	Arg1 is less than to arg2.
[ <i>arg1</i> -le <i>arg2</i> ]	<i>Arg1</i> is less than or equal to <i>arg2</i> .
[ <i>arg1</i> -gt <i>arg2</i> ]	Arg1 is greater than to arg2.
[ arg1 -ge arg2 ]	Arg1 is greater than or equal to
L . 0 0 0 - 1	arg2.

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

	true.
[expr1 -0 expr2]	True if either <i>expr1</i> and <i>expr2</i> are
	true.
CONDITIONS (LEXICOGRAPHIC)	
[-z <i>str</i> ]	<i>str</i> is zero length.
test -z str	str is zero length, test command works
	for all other <i>cond</i> itions too.
[ -n <i>str</i> ]	str is non-zero length.
[ <i>str1</i> = <i>str2</i> ]	Str1 is the same as str2.
[ str1 > str2 ]	<i>Str1</i> sorts lexicographically after <i>str2</i> .

Str1 sorts lexicographically before

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

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

DASH SUNI TING		
#!/bin/bash		
#This is comment, below is your script	comment, below is your script	
foo(){		
local x=1;	Sample bash script	
# This function is not implemented	Sample bash script	
return \$x;		
}		
exit `foo`		
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	
Shift IV	affected.	
<pre>function func { cmds; }</pre>	Declare a function named func containing <i>cmds</i> .	
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.	