Special Characters

&	Background job
#	Comment
~	Home Directory
!	Logical NOT
‘	Quote (Strong)
“	Quote (Weak)
<	Redirect input
>	Redirect output
>>	Redirect output + append to file
|	Redirect (pipe) output to next command
/	Separator for pathname directories
;	Separator for shell commands
[ ]	Start and end a character-set wildcard
{ }	Start and end a command block
( )	Start and end a subshell
(( ))	Perform arithmetic
*	Wildcard
?	Wildcard – single character
$	Variable expression
\	Escape a special character
n>&m	Descriptor n is a copy of ouput file descriptor m
n<&m	Descriptor n is a copy of input file descriptor m

String Operators

${varname:-word}	Returns word
${varname:=word}	Sets and returns word
${varname:?message}	Prints message and exits
${varname:offset:length}	Returns substring
${varname:+word}	If varname is defined, return word

Pattern-matching operators

${varname#pattern}	Match first from the start
${varname##pattern }	Match last from the start
${varname%pattern}	Match first from the end
${varname/pattern/replace}	Match longest and replace
${varname//pattern/replace}	Match all and replace

Variables

$0, $1, $2,	Positional parameters
$@	“$1” “$2” “$3” …
$*	A string of positional params > 0
$#	Number of positional params
$?	Exit status of last command run

Functions

define	function myfunction { … } *or* myfunction ( ) { … }
call	myfunction arg1 arg2 …
keywords	local – limit var scope

If / else conditions

x && y	If x runs, then run y
x || y	If x fails, then run y
x -a y	x AND y
x -o y	x OR y
-lt, -le, -eq, -gt, -ge, -ne	Integer comparisons
=, !=, <, >	String comparisons
-n str1	str1 has length > 0 (nonzero)
-z str1	str1 has length 0 (zero)
-d file	File exists and is a directory
-e file	File exists
-f file	File exists and is a regular file
-r file	User has read permission on file
-s file	File exists and is non empty
-w file	User has write permission on file
-x file	User has execute permission on file, or search if directory
-N file	File was modified since it was left read
-O file	User owns file
-G file	File’s group ID matches the user’s group ID
file1 -nt file2	file1 has newer modification time than file2

Flow control sentences

if	if *condition*; then *commands*; fi
for	for ((*init*; *condition*; *increment*)); do *commands*; done
for	for *var* in *array*; do *commands*; done
case	case *expression* in *pattern1*) *commands* ;; *pattern2*) *commands* ;; *) *commands* ;; esac
while	while *condition*; do *commands*; done
until	until *condition*; do *commands*; done

Arrays

Arr_name=(‘el1’ ‘ el2’ ‘el3’)	define
Arr_name[index]	Element #index
Arr_name[-1]	Last element
Arr_name[@]	All elements, space-separated
#Arr_name[@]	Array length
#Arr_name[index]	String length of the Nth element
Arr_name[@]:m:n	Range (from position m, length n
!Arr_name[@]	Keys of all elements
Arr_name=("${Arr_name[@]}” “newElement”)	Push
Arr_name+=('newElement’)	Also Push
unset Arr_name[n]	Remove one item

Dictionaries

declare –A dict	Define
dict[key]=”value”	Define value of a key
dict[key]	Value of a key
dict[@a]	All values
!dict[@]	All keys
#dict[@]	Number of elements
unset dict[key]	Delete the key

Useful Commands

type <cmd>	Determine type of command: -a ; displays all the locations
builtin <cmd>	Run builtin commands explicitly
which <cmd>	Locate the executable of a command: -a ; show all locations
clear	Clear the terminal screen
echo “str1”	Print message to terminal screen: -e ; uses escape sequences like (\n = newline, \t = tab) -n ; supresses automatic newline after print
printf <format> <variables>	Print messages to terminal screen. Formatting be like: %s – String %-Xs – String wide X chars, left aligned %Xs – String wide X chars, right aligned %d – Integer (%-Xd, %Xd) %f – Float %.Xf - Round to X decimal spaces
date <options> <+format>	Will display date and time. Formats (“+%Y-%m-%d”): %Y – Year, %m – month, %d – day, %H – hours, %M – minutes, %S – seconds, (%A uppercase for full name) %a – DayOfTheWeek, (%B) %b - Month Options (-d “yesterday”): “yesterday”, “next Monday”,
read <options> <variable>	Read input from user or file and store into variable (read var1). Options: -p “Text” : print before input -a : store the input in array
history <options>	Display the command history for that session. Options: -c : clear the history -X : print the last X commands -a : appends history to bash history file -d X : deletes the command with index X from history
sleep <num_time>	Delay the execution of a script. Num_time: Xs : delay for X second(s) (default) Xm : delay for X minute(s) Xh : delay for X hour(s)
man <command>	Opens the manual pages for the <command>.
ls <options> <path>	List the files and directories in the current working directory or given path. Options: -l : list detailed view for files -a : show all files, even hidden -alp : ???
find <path> <options>	Look recursively for files. Options: -name “pattern”: look for file names -type X: f = regular file, d = directory
pwd	Display the current working directory.
cd <directory>	Change the current working directory. <directory>: ‘/path’ : changes directory to path ‘..’ : changes to parent directory of the current one ‘~username’ : changes to home directory for username ‘-’ : changes to previous working directory used
mkdir <directory>	Creates new directory. <directory> can be: ‘d1’ : creates new directory called d1 ‘d1’ ‘d2’ ‘d3’ : creates more directories in the current one -p ‘d1/d2’ : creates d1 and another directory d2 as d1’s child
rmdir <directory>	Works the same as mkdir, but it deletes the directory if it is empty.
cat <file>	Display the contents of the file on the terminal. <file>: ‘file.txt’ : displays file.txt ‘f1.txt’ ‘f2.txt’ : displays files consecutively -n ‘file.txt’ : displays file.txt with numbered lines
more, less, od, hexdump	More and less are both text viewers, od gives octal output and hexdump hexadecimal.
vi, vim, emacs, nano	File editors. Use ‘man file_editor’ to learn how to use them.
cp <source> <destination>	Copy files or directories from source to destination. cp file /path : copy file to path cp –r directory /path : copy directory with all its contents to path
mv <source> <destination>	Moves files or directories from source to destination. mv file /path : move file to path mv directory /path : move directory to path mv file.txt newfile.txt : renames file.txt to newfile.txt
rm <options> <file>	Remove or delete files from directories. Options: -r: recursive -f: force the removal
head <options> <file(s)>	Display the beginning of a text file. -n X: specify the number of lines -c X: displays X bytes and not lines
tail <options> <file(s)>	Display the last few lines of a text file. Counterpart to ‘head’. -n X: specify the number of lines -c X: displays X bytes, not lines
cut <options> <file>	-c X: specify positions to cut (1-5 file.txt wil extract first five from lines) -f X: specify the fields to extract -d X: specify the delimiter for cut
sort <options> <file>	Sort the lines of a text file al. -r: reverse the lines order (Z-A) -n: perform numerical sort instead -u: outputs only the unique lines -f: ignore cases
seq <min> <max>	Generate sequence of numbers.
shuf <options> <file>	Generate random permutations. -n X: Outputs at most X lines. -o FILE: Writes the output to file -r: allow repeated samples

$(($RANDOM % MAX + 1))	Returns a random number from 1 to MAX.
nl <options> <file>	Add line numbers to a file or input stream.
uniq <options> <file>	Removes all consecutive lines. Options: -c : also counts the amount of duplicates -i : ignores the case -d : outputs only duplicates -u : outputs only the unique
rev <file>	Reverse the characters in each line of the input stream or file
tr <options> <set1> <set2> <file>	Translate or delete characters. Set1 is translated to Set2. -d : removes the characters -c : complement the Set1
wc <options> <file>	Counts the number of lines, words, bytes. Options: -l : only counts the lines -w : only counts the words -c : only counts the bytes
grep <options> <pattern> <file>	Search for specific pattern or regular expression. Options: -i : ignore case -v : invert the match (print only the lines not matching the pattern) -w : match only whole words -n : print the line numbers for each match -r : search recursively through directories -e: advanced pattern matching ()-group {}-multiplier []-range +,*,? – wildcards \bXXX\b-word |- or \N-backreference
shift <X>	Shift the positional parameters to the left. X is number of positions to shift.
jobs <options>	Display a list of jobs that are currently running in the background or are suspended. -l : also displays PID of a job -p : displays only the PIDs -r : displays the running jobs -s : displays the stopped jobs
fg <JID>	Bring a job that is running in the background to the foreground.
bg <JID>	Start a suspended job in the background.
disown %<JID>	Remove jobs from shell’s job control. (disown %2 : removes job with JID 2)
ulimit <options>	Display the resource limits of the current shell and its children. -a : displas all current limits

PROGRAMMING IN C

syscall(x, …)	Make system calls in a program. x: System call number …: Arguments required for system call x
perror(char* str)	Print a descriptive error message to stderr
open(path, flags, mode)	Open or create new files. Flags: O_RDONLY: read only O_WRONLY: write only O_APPEND: append O_RDWR: reading and writing O_CREAT: create file if not exist O_TRUNC: truncate file to 0 len
close(fd)	Close the file descriptor fd
read(fd, *buffer, x)	Read data from a file or file descriptor fd. Stores read data to buffer and read x bytes.
write(fd, *buffer, x)	Write x bytes from buffer to file descriptor fd.
printf(…)	Format and print data to stdout
dup(oldfd)	Duplicate an existing file descriptor oldfd to a new one
dup2(oldfd, newfd)	Duplicate an existing file descriptor oldfd to a specified file descriptor number newfd
rename(oldname, newname)	Change the name of an existing file or directory.
link(oldpath, newpath)	Create a new hard link to an existing file.
unlink(pathname)	Remove a specific file from the file system.
chmod(pathname, mode)	Change the permissions of a file or dir in the file system.
chown(pathname, owner, group)	Change ownership of a file or dir in the file system.
readdir(DIR *dirp)	Used to read contents of a directory.
opendir(char* dirname)	Open a directory. Returns DIR* to directory system.
closedir(*dirp)	Close a directory system.
chdir(path)	Change the current working directory of the process.
mkdir(path, mode)	Create new directory.
rmdir(path)	Remove or delete an empty dir
symlink(target, linkpath)	Create soft link / symbolic link. Linkpath references to target
readlink(path, buffer, buf_size)	Read value of a symbolic link.
getuid(), setuid(), getgid(), setgid(), geteuid(),getegid()	Get parameters: UID – user ID, GID – group ID, EUID – effective user ID
fork()	Create a new process by duplicating the existing process. Returns pid_t -> 0 = child
exec()	Replace the current process with a new process. execl(), execle(), execlp(): take program name and a list of arguments execv(), execvp(): take program name and an array of arguments execve(): similar to execvp() but you can specify environment vars
wait(int* status)	Make the parent process wait until one of its child processes terminates.
waitpid(pid, status)	Wait for specific process with pid to terminate.
exit(x)	Terminate the current process and return exit status x.
getpid(), getppid()	Retrieve process ID, retrieve parent process ID
sleep(x)	Suspend the execution of a program for x seconds.
pipe(int pipefd[2])	Create an interprocess communication pipe. pipefd[0]: file descriptor for read pipefd[1]: file descriptor for write
kill(pid, sig)	Kill a signal to a specified process(es).
signal()	Specify the action to be taken when a particular signal is received by a process.

USERS AND DOCUMENTS (Bash)

whoami	Display the username of the current user
id	Display the user and group
groups <user>	Display the groups to which current user or <user> belongs
passwd <options> <username>	Change or update the password of a user account
$UID	Variable, holds user id
$HOME	Variable, holds the absolute path to current user’s home dir
sudo <options> <command>	Execute commands with elevated privileges
su <options> <username>	Switch to user <username>
useradd, userdel, usermod	Create new user, delete an user, modify user account
groupadd, groupdel, groupmod	Create new group, delete a group, modify group
ln -s <target> <link_name>	Create soft link with link_name that refers to target file or dir
ln <target> <link_name>	Create hard link or directory links.
readlink <link_name>	Display the target of a symbolic link
chown <user><:group> <file(s)>	Change the ownership of files or dirs. <user> and <:group> represents new owners.
chgrp <group> <file(s)>	Change the group ownership of files or directories

PROCESSES, SIGNALS, PIPES (Bash)

ps	Display information about active processes running on system
pidof <program_name>	Find the process ID (PID) of a running program based on name
pgrep <pattern>	Find PIDs by pattern
pstree	Display a tree-like representation of running processes
top	Monitor and manage system resources in real-time
kill <options> <pid(s)>	Send a signal to terminate to processes. Options: -s sig: sig(SIGTERM, SIGKILL, SIGINT) -a: send signal to all processes
trap <action> <signal(s)>	Define actions to be taken when specific signals are received. action -> command to be executed when signal(s) received
<cmd> | <cmd>	Pipe (no explanation needed)

THREADS (C)

pthread_t tx	Object that stores thread id
pthread_createt(thread, attr, start_routine, arg)	Create a new thread within multi-threaded program. thread:pointer to pthread_t attr:attributes for a thread start_routine: pointer to the function that will be executed by the new thread arg:optional arguments
pthread_join(thread, **value_ptr)	Wait for a specific thread to terminate. Value_ptr is optional for saving exit stat
pthread_yield()	Voluntarily yield the processor by suspending the execution of the calling thread
pthread_cancel(thread)	Request the cancellation of a specified thread.