**Basic UNIX Commands**

**#1) cal**: Displays the calendar.

* **Syntax**: cal [[month] year]
* **Example**: display the calendar for April 2018
  + $ cal 4 2018

**#2) date:** Displays the system date and time.

* **Syntax**: date [+format]
* **Example**: Display the date in dd/mm/yy format
  + $ date +%d/%m/%y

**#3) banner**: Prints a large banner on the standard output.

* **Syntax**: banner message
* **Example**: Print “Unix” as the banner
  + $ banner Unix

**#4) who**: Displays the list of users currently logged in

* **Syntax**: who [option] … [file][arg1]
* **Example**: List all currently logged in users
  + $ who

**#5) whoami**: Displays the user id of the currently logged-in user.

* **Syntax**: whoami [option]
* **Example**: List currently logged in user
  + $ whoami

**#1) touch**: Create a new file or update its timestamp.

* **Syntax**: touch [OPTION]…[FILE]
* **Example**: Create empty files called ‘file1’ and ‘file2’
  + $ touch file1 file2

**#2) cat**: Concatenate files and print to stdout.

* **Syntax**: cat [OPTION]…[FILE]
* **Example**: Create file1 with entered cotent
  + $ cat > file1
  + Hello
  + ^D

**#3) cp**: Copy files

* **Syntax**: cp [OPTION]source destination
* **Example**: Copies the contents from file1 to file2 and contents of file1 is retained
  + $ cp file1 file2

**#4) mv**: Move files or rename files

* **Syntax**: mv [OPTION]source destination
* **Example**: Create empty files called ‘file1’ and ‘file2’
  + $ mv file1 file2

**#5) rm**: Remove files and directories

* **Syntax**: rm [OPTION]…[FILE]
* **Example**: Delete file1
  + $ rm file1

**#6) mkdir**: Make directory

* **Syntax**: mkdir [OPTION] directory
* **Example**: Create directory called dir1
  + $ mkdir dir1

**#7) rmdir**: Remove a directory

* **Syntax**: rmdir [OPTION] directory
* **Example**: Create empty files called ‘file1’ and ‘file2’
  + $ rmdir dir1

**#8) cd**: Change directory

* **Syntax**: cd [OPTION] directory
* **Example**: Change working directory to dir1
  + $ cd dir1

**#9) pwd**: Print the present working directory

* **Syntax**: pwd [OPTION]
* **Example**: Print ‘dir1’ if a current working directory is dir1
  + $ pwd

**#1) ls**: List directory contents

* **Syntax**: ls [OPTION] [FILE]
* **Example**: list all (including hidden files) directory contents, in long format, sorted by time,
  + $ ls -alt

**#2) which:**Locate a command

* **Syntax**: which [-a] filename
* **Example**: List all paths from where ‘cat’ can run
  + $ which -a cat

**#3) man**: Interface for working with the online reference manuals.

* **Syntax**: man [-s section] item
* **Example**: Show manual page for the ‘cat’ command
  + $ man cat

**#4) su**: Change user-id or become super-user.

* **Syntax**: su [options] [username]
* **Example**: Change user-id to ‘user1’ (if it exists)
  + $ su user1

**#5) sudo**: Execute a command as some other user or super-user

* **Syntax**: sudo [options] [command]
* **Example**: Get a file listing of an unlisted directory
  + $ sudo ls /usr/local/protected

**#6) find**: Used to search for files and directories as mentioned in the ‘expression’

* **Syntax**: find [starting-point] [expression]
* **Example**: In ‘/usr’ folder, find character device files, of name ‘backup’
  + $ find /usr -type c -name backup

**#7) du**: Estimate disk usage is blocks

* **Syntax**: du [options] [file]
* **Example**: Show number of blocks occupied by files in the current directory
  + $ du

**#8) df**: Show number of free blocks for mounted file system

* **Syntax**: df [options] [file]
* **Example**: Show number of free blocks in local file systems
  + $ df -l

**1. chmod**: change file access permissions

* **description:**This command is used to change the file permissions. These permissions are read, write and execute permission for the owner, group, and others.
* **syntax (symbolic mode)**:

chmod [ugoa][[+-=][mode]] file

* The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)thers or (a)ll
* The second optional parameter indicates opcode – this can be for adding (+), removing (-) or assigning (=) permission.
* The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.

**Example**: Add write permission for user, group and others for file1

$ ls -l

-rw-r–r– 1 user staff 39 Jun 21 15:37 file1  
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2

$ chmod ugo+w file1

$ ls -l

-rw-rw-rw- 1 user staff 39 Jun 21 15:37 file1  
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2

$ chmod o-w file1

$ ls -l

-rw-rw-r– 1 user staff 39 Jun 21 15:37 file1  
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2

* **syntax (numeric mode)**:

chmod [mode] file

* The mode is a combination of three digits – the first digit indicates the permission for the user, the second digit for the group, and the third digit for others.
* Each digit is computed by adding the associated permissions. Read permission is ‘4’, write permission is ‘2’ and execute permission is ‘1’.
* **Example**: Give read/write/execute permission to the user, read/execute permission to the group, and execute permission to others.

$ ls -l

-rw-r–r– 1 user staff 39 Jun 21 15:37 file1  
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2

$ chmod 777 file1

$ ls -l

-rwxrwxrwx 1 user staff 39 Jun 21 15:37 file1  
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2

**2. chown**: change ownership of the file.

* **description**: Only the owner of the file has the rights to change the file ownership.
* **syntax**: chown [owner] [file]
* **Example**: Change the owner of file1 to user2 assuming that it is currently owned by the current user

$ chown user2 file1

**3. chgrp**: change the group ownership of the file.

* **description**: Only the owner of the file has the rights to change the file ownership.
* **syntax**: chgrp [group] [file]
* **Example**: Change group of file1 to group2 assuming it is currently owned by the current user.

$ chgrp group2 file1

While creating a new file, Unix sets the default file permissions. Unix uses the value stored in a variable called umask to decide the default permissions. The umask value tells Unix which of the three sets of permissions need to be disabled.

The flag consists of three octal digits, each representing the permissions masks for the user, the group, and others. The default permissions are determined by subtracting the umask value from ‘777’ for directories and ‘666’ for files. The default value of the umask is ‘022’.

**4. umask**: change default access permissions

* **description:**This command is used to set the default file permissions. These permissions are read, write and execute permission for owner, group, and others.
* **syntax**: umask [mode]
* The mode is a combination of three digits – the first digit indicates the permission for the user, the second digit for the group, and the third digit for others.
* Each digit is computed by adding the associated permissions. Read permission is ‘4’, write permission is ‘2’ and execute permission is ‘1’.

**Example**: Give read/write/execute permission to the user, and no permissions to group or others. i.e. the permission for files will be 600, and for directories will be 700.

$ umask 077

* **Example**: Give read/write/execute permission to the user, read/execute permissions to group or others for directories and read-only permission to group or others for other files. i.e. the permission for files will be 644, and for directories will be 755.

$ umask 022

### Find Command in Unix

**Syntax:**

find [options] [paths] [expression]

**List all files found in the current directory and its hierarchy**

$ find.

**List all files found in the current hierarchy, and all the hierarchy below /home/xyz**

$ find. /home/XYZ

**Search for a file by the name abc in the current directory and its hierarchy**

$ find ./ -name abc

**Search for a directory by the name xyz in the current directory and its hierarchy**

$ find ./ -type d -name xyz

**Search for a file by the name abc.txt below the current directory, and prompt the user to delete each match.**

Note that the “{}” string is substituted by the actual file name while running and that the “\;” string is used to terminate the command to be executed.

$ find ./ -name abc.txt -exec rm -i {} \;

**Search for files that were modified in the last 7 days below the current directory**

$ find ./ -mtime -7

**Search for files that have all permissions set in the current hierarchy**

$ find ./ -perm 777

Grep Command in Unix with Examples

**Syntax:**

grep [options] [pattern] [file]

The pattern is specified as a regular expression. A regular expression is a string of characters that is used to specify a pattern matching rule. Special characters are used to define the matching rules and positions.

**#1) Anchor Characters:** ‘^’ and ‘$’ at the beginning and end of the pattern are used to anchor the pattern to the start of the line, and to the end of the line respectively.

**Example:** “^Name” matches all lines that start with the string “Name”. The strings “\<” and “\>” are used to anchor the pattern to the start and end of a word respectively.

**#2) Wildcard Character:** ‘.’ Is used to match any character.

**Example:**“^.$” will match all lines with any single character.

**#3) Escaped Characters:** Any of the special characters can be matched as a regular character by escaping them with a ‘\’.

**Example:** “\$\\*” will match the lines that contain the string “$\*”

**#4) Character Range:** A set of characters enclosed in a ‘[‘ and ‘]’ pair specify a range of characters to be matched.

**Example:** “[aeiou]” will match all lines that contain a vowel. A hyphen can be used while specifying a range to shorten a set of consecutive characters. **E.g.** “[0-9]” will match all lines that contain a digit. A carat can be used at the beginning of the range to specify a negative range. **E.g.** “[^xyz]” will match all lines that do not contain x, y or z.

**#5) Repetition Modifier:** A ‘\*’ after a character or group of characters is used to allow matching zero or more instances of the preceding pattern.

**The grep command supports a number of options for additional controls on the matching:**

* -i: performs a case-insensitive search.
* -n: displays the lines containing the pattern along with the line numbers.
* -v: displays the lines not containing the specified pattern.
* -c: displays the count of the matching patterns.

**Examples:**

* Match all lines that start with ‘hello’. **E.g:** “hello there”

$ grep “^hello” file1

* Match all lines that end with ‘done’. **E.g:** “well done”

$ grep “done$” file1

* Match all lines that contain any of the letters ‘a’, ‘b’, ‘c’, ‘d’ or ‘e’.

$ grep “[a-e]” file1

* Match all lines that do not contain a vowel

$ grep “[^aeiou]” file1

* Match all lines that start with a digit following zero or more spaces. **E.g:** “ 1.” or “2.”

$ grep “ \*[0-9]” file1

* Match all lines that contain the word hello in upper-case or lower-case

$ grep -i “hello”

### Cut Command in Unix with Examples

The cut command extracts a given number of characters or columns from a file. For cutting a certain number of columns it is important to specify the delimiter. A delimiter specifies how the columns are separated in a text file

**Example:** Number of spaces, tabs or other special characters.

**Syntax:**

cut [options] [file]

The cut command supports a number of options for processing different record formats. For fixed width fields, the -c option is used.

$ cut -c 5-10 file1

This command will extract characters 5 to 10 from each line.

For delimiter separated fields, the -d option is used. The default delimiter is the tab character.

$ cut -d “,” -f 2,6 file1

This command will extract the second and sixth field from each line, using the ‘,’ character as the delimiter.

**Example:**

**Assume the contents of the data.txt file is:**

Employee\_id;Employee\_name;Department\_name;Salary  
10001;Employee1;Electrical;20000  
10002; Employee2; Mechanical;30000  
10003;Employee3;Electrical;25000  
10004; Employee4; Civil;40000

**And the following command is run on this file:**

$ cut -c 5 data.txt

**The output will be:**

o

1

2

3

4

**If the following command is run on the original file:**

$ cut -c 7-15 data.txt

**The output will be:**

ee\_id; Emp

Employee1

Employee2

Employee3

Employee4

**If the following command is run on the original file:**

$ cut -d “,” -f 1-3 data.txt

**The output will be:**

Employee\_id;Employee\_name;Department\_name

10001;Employee1;Electrical

10002; Employee2; Mechanical

10003;Employee3;Electrical

10004; Employee4; Civil

**ls Syntax:**

ls [options] [paths]

**The ls command supports the following options:**

* ls -a: list all files including hidden files. These are files that start with “.”.
* ls -A: list all files including hidden files except for “.” and “..” – these refer to the entries for the current directory, and for the parent directory.
* ls -R: list all files recursively, descending down the directory tree from the given path.
* ls -l: list the files in long format i.e. with an index number, owner name, group name, size, and permissions.
* ls – o: list the files in long format but without the group name.
* ls -g: list the files in long format but without the owner name.
* ls -i: list the files along with their index number.
* ls -s: list the files along with their size.
* ls -t: sort the list by time of modification, with the newest at the top.
* ls -S: sort the list by size, with the largest at the top.
* ls -r: reverse the sorting order.

**Examples:**

**List all non-hidden files in the current directory**

$ ls

**E.g:**

dir1 dir2 file1 file2

**List all the files including hidden files in the current directory**

$ ls -a

**E.g:**

..   ... .... .hfile dir1 dir2 file1 file2

**List all the files including hidden files in the current directory**

$ ls -al

**E.g:**

total 24

drwxr-xr-x 7 user staff 224 Jun 21 15:04 .

drwxrwxrwx 18 user staff 576 Jun 21 15: 02.

-rw-r--r-- 1 user staff 6 Jun 21 15:04 .hfile

drwxr-xr-x 3 user staff 96 Jun 21 15:08 dir1

drwxr-xr-x 2 user staff 64 Jun 21 15:04 dir2

-rw-r--r-- 1 user staff 6 Jun 21 15:04 file1

-rw-r--r-- 1 user staff 4 Jun 21 15:08 file2

**List all the files in the current directory in long format, sorted by modification time, oldest first**

$ ls -lrt

**E.g:**

total 16

-rw-r--r-- 1 user staff 6 Jun 21 15:04 file1

drwxr-xr-x 2 user staff 64 Jun 21 15:04 dir2

-rw-r--r-- 1 user staff 4 Jun 21 15:08 file2

drwxr-xr-x 3 user staff 96 Jun 21 15:08 dir1

**List all the files in the current directory in long format, sorted by size, smallest first**

$ ls -lrS

**E.g:**

total 16

-rw-r--r-- 1 user staff 4 Jun 21 15:08 file2

-rw-r--r-- 1 user staff 6 Jun 21 15:04 file1

drwxr-xr-x 2 user staff 64 Jun 21 15:04 dir2

drwxr-xr-x 3 user staff 96 Jun 21 15:08 dir1

**List all the files recursively from the current directory**

$ ls -R

**E.g:**

dir1 dir2 file1 file2

./dir1:

file3

./dir2:

**Tar Syntax:**

tar [function] [options] [paths]

**Tar options:**

**The tar command supports the following functions:**

* tar -c: Create a new archive.
* tar -A: Append a tar file to another archive.
* tar -r: Append a file to an archive.
* tar -u: Update files in an archive if the one in the filesystem is newer.
* tar -d: Find the diff between an archive and the filesystem.
* tar -t: List the contents of an archive.
* tar -x: Extract the contents of an archive.

While specifying the function, the ‘-‘ prefix is not required, and the function can be followed by other single letter options.

**Some of the supported options include:**

* -j: Read or write archives using the bzip2 compression algorithm.
* -J: Read or write archives using the xz compression algorithm.
* -z: Read or write archives using the gzip compression algorithm.
* -a: Read or write archives using the compression algorithm determined by the archive file name.
* -v: Perform the operations verbosely.
* -f: Specify the file name for the archive.

**Examples:**

**Create an archive file containing file1 and file2**

$ tar cvf archive.tar file1 file2

**Create an archive file containing the directory tree below dir**

$ tar cvf archive.tar dir

**List the contents of archive.tar**

$ tar tvf archive.tar

**Extract the contents of archive.tar to the current directory**

$ tar xvf archive.tar

**Create an archive file containing the directory tree below dir and compress it using gzip**

$ tar czvf archive.tar.gz dir

**Extract the contents of the gzipped archive file**

$ tar xzvf archive.tar.gz

**Extract only the given folder from the archive file**

$ tar xvf archive.tar docs/work

**Extract all “.doc” files from the archive**

$ tar xvf archive.tar –-wildcards ‘\*.doc’

| **Command** | **ps - displays a snapshot of all current processes** |
| --- | --- |
| Common Syntax | $ ps [options] |
| Example | $ ps -ef |
|  | Show every process running, formatted as a table |
| **Command** | **top - displays a live status of current processes** |
| Common Syntax | $ top [options] |
| Example | $ top |
|  | Show a live view of all current processes |

| **Command** | **bg - resume a background suspended a job** |
| --- | --- |
| Common Syntax | $ bg [job\_spec …] |
| Example | $ xterm Ctrl-Z $ bg |
|  | Continue running a job that was previously suspended (using Ctrl-Z) in the background |
| **Command** | **fg - bring a background job to the foreground** |
| Common Syntax | $ fg [job\_spec] |
| Example | $ xterm Ctrl-Z $ bg $ fg |
|  | Bring a previous background job to the foreground |

| **Command** | **clear – clear a terminal screen** |
| --- | --- |
| Common Syntax | $ clear |
| Example | $ clear |
|  | Clear all prior text from the terminal screen |
| **Command** | **history – print history of commands in the current session** |
| Common Syntax | $ history [options] |
| Example | $ history |