03 – File and Directory Commands

ls (list)

- Use **ls** without any arguments to display current directory contents.
- **ls** with the **-a** option. files all begin with a "**dot**", which indicates they are "**hidden**" files.
  - **ls -a**
- **ls** with **-F**, this command is useful for distinguishing between directories, ordinary files, and executable files.
  - **ls -F**
- **ls** with **-l** to obtain a "long" listing of files. An explanation of the information it provides appears below. **ls -l**

```
-rwxr-xr-x   1 jsmith   staff   43      Mar 23 18:14  prog1
-rw-r--r--   1 jsmith   staff   10030   Mar 22 20:41 sample.f
drwxr-sr-x   2 jsmith   staff   512     Mar 23 18:07 subdir1
drwxr-sr-x   2 jsmith   staff   512     Mar 23 18:06 subdir2
drwxr-sr-x   2 jsmith   staff   512     Mar 23 18:06 subdir3
      1      2   3      4       5       6              7
```

1 = access modes/permissions
2 = number of links
3 = owner
4 = group
5 = size (in bytes)
6 = date/time of last modification
7 = name of file

- ls with R, Recursive listings which will display files and folders inside the folders recursively.
  - **ls -R**
  - **ls -Rl**

=== * ===

pwd (Present Working Directory)

- Will give the present working directory path informatoin

=== * ===

mkdir (Make Directory)

- mkdir will create a new directory (folder)
- **Syntax**: **mkdir directoryname(folder name)**

Example 1: **mkdir polytechnic** // will create directory with name polytechnic

Example 2: Create some additional subdirectories within **polytechnic**. List new directories after the command completes.

mkdir polytechnic/mechanical polytechnic/computer polytechnic/civil

ls polytechnic

Example 3: Try to create a directory in a location where user doesn't have permission.

mkdir /etc/mydir

Output: mkdir: cannot create directory '/etc/mydir': Permission denied

=== * ===

cd (Change Directory)

- cd is used to Change Directory
- Change to home direcotory
  - **cd** // Change to default home directory
  - **cd ~** // Change to default home directory
- Change to a subdirectory within user home directory
  - **cd polytechnic/computer**
  - pwd
- Go up one level back to the current directory's parent directory
  - **cd ..**
  - pwd
- Change to the root (top-most) directory
  - **cd /**
  - pwd
- Change to another directory
  - **cd ~/polytechnic**
  - pwd
- Change to another one of subdirectories
  - **cd ~/polytechnic/computer**
  - pwd

=== * ===

rmdir (Remove Directory)

- Will remove (delete) the directory
- Make sure that folder is empty before issuing rmdir command.
- Make sure current directory is not part of the directory being deleted.
    - **rmdir civil** **(**assume current directory is /home/polytechnic**)**
    - rmdir /polytechnic/computer (assume /polytechnic/computer folder exists)

=== * ===

File Manipulation Commands:

Creating File:

- **touch**: will create empty files
    - Example: touch a.txt b.txt
- **Create file and add contents** at the end press enter key and **Ctrl+z**
    - Example:

cat > a.txt
Welcome to Operating System Lab
Enjoy Typing commands
Ctrl+z
cat a.txt

=== * ===

Display the Contents of Files:

- **Syntax**: **cat filename**
    - **cat /path/filename**
- **Example**:
    - **cat a.txt**
    - **cat polytechnic/computer/a.txt** (Assume a.txt is present in polytechnic/computer directory)

=== * ===

rm (Remove File)

- Will remove (delete) the files
- **Syntax: rm FileName**
  - **rm path/Filename**
- **Example**:
  - **rm a.txt** // Assume a.txt is present in current directory
  - **rm polytechnic/civil/a.txt** // Assume polytechnic/civil folder has file a.txt
  - **rm -i *** // will delete all the files but interactively, asking user about confirmation of deleting the files
  - **rm polytechnic/civil/*** // will delete all files in polytechnic/civil folder

=== * ===

cp (Copy)

- Used to create a copy of a existing file.
- Copy an existing file in current directory to another file in the current directory. Make sure that the file to be copied must be exists in specified location.
- **Syntax: cp file1 file2**
- **cp path1/file1 path2/file2** // creates copy of file1 in current directory or in specified directory
- **cp a.txt b.txt** **(**Make sure file a.txt exists in current directory)
  - new file b.txt is created and contents of b.txt is same as a.txt
- **cp polytechnic/civil/a.txt polytechnic/computer/b.txt**
  - new file b.txt will be created in polytechnic/computer (Make sure both polytechnic/civil and polytechnic/computer folder exists)
- In order to avoid accidental over writting, **-i option** can be used. It will altert the user when file2 is already exists.
  - **cp -i file1 file2** (to test, make sure both files exists)
- Use the recursive option to copy an entire subdirectory to a new subdirectory and then list both directories to prove that it worked:
  - **cp -R subdir1 subdir4**
- Copying a file from another location to the current directory and want its name to remain the same, user can use the shorthand "." to indicate the current directory.
  - **cp polytechnic/civil/a.txt .**

=== * ===

mv (Move)

- Used to rename the file (in other words, used to move the files from one location to another location)
- It can be used to rename the Directory (Folder) also.
- **Example 1:**
  - **mv OldFileName NewFileName**
  - ls
- **Example 2:**
  - **mv OldFolderName NewFolderName**
  - ls

=== * ===

Pipes: The pipe command lets user sends the output of one command to another. **Piping**, as the term suggests, can redirect the standard output, input, or error of one process to another for further processing. A traditional pipe is "**unnamed**" and lasts only as long as the process.
A **named** pipe, however, can last as long as the system is up, beyond the life of the process. It can be deleted if no longer used.

Named Pipe:

- In computing, a named pipe (also known as a **FIFO**) is one of the methods for inter-process communication.
- It is an extension to the traditional pipe concept on Unix.
- Usually a named pipe appears as a file and generally processes attach to it for inter-process communication.
- A FIFO file is a special kind of file on the local storage which allows two or more processes to communicate with each other by reading/writing to/from this file.
- A FIFO special file is entered into the file system by calling **mkfifo**() in C.
- Once a FIFO special file is created in this way, any process can open it for reading or writing, in the same way as an ordinary file. However, it has to be **open at both ends simultaneously** before it can proceed to do any input or output operations on it.

- **Creating a FIFO file:** In order to create a FIFO file, a function calls i.e. **mkfifo** is used.
- **Example:** a.c and b.c communication

// a.c

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   int fd;

   // FIFO file path
   char * myfifo = "/tmp/myfifo";

   // Creating the named file(FIFO)
   // mkfifo(<pathname>, <permission>)
   mkfifo(myfifo, 0666);

   char arr1[80], arr2[80];
   while (1)
   {
      // Open FIFO for write only
      fd = open(myfifo, O_WRONLY);

      // Take an input arr2ing from user.
      // 80 is maximum length
      fgets(arr2, 80, stdin);

      // Write the input arr2ing on FIFO
      // and close it
      write(fd, arr2, strlen(arr2)+1);
      close(fd);

      // Open FIFO for Read only
      fd = open(myfifo, O_RDONLY);

      // Read from FIFO
      read(fd, arr1, sizeof(arr1));

      // Print the read message
      printf("User2: %s\n", arr1);
      close(fd);
   }
   return 0;
}
```

// b.c

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   int fd1;

   // FIFO file path
   char * myfifo = "/tmp/myfifo";

   // Creating the named file(FIFO)
   // mkfifo(<pathname>,<permission>)
   mkfifo(myfifo, 0666);

   char str1[80], str2[80];
   while (1)
   {
      // First open in read only and read
      fd1 = open(myfifo,O_RDONLY);
      read(fd1, str1, 80);

      // Print the read string and close
      printf("User1: %s\n", str1);
      close(fd1);

      // Now open in write mode and write
      // string taken from user.
      fd1 = open(myfifo,O_WRONLY);
      fgets(str2, 80, stdin);
      write(fd1, str2, strlen(str2)+1);
      close(fd1);
   }
   return 0;
}
```

=== * ===

Unnammed Pipe:

- pipe( ) — Create an unnamed pipe
- **Format:** Fraction of C Code for the usage of pipe( )

```
#define _POSIX_SOURCE
#include <unistd.h>
…
int pipe(int fdinfo[2]);
…
pipe()
```

This example creates an I/O channel. The output shows the data written into one end and read from the other.

```
#define _POSIX_SOURCE
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

void reverse(char *s)
{
//  char *first, *last, temp;
  char temp;
  int i, n;
  n = strlen(s)-1;
  for(i=0;i<n/2; i++)
  {
    temp = s[i];
    s[i] = s[n-i];
    s[n-i]=temp;
  }
}

void main()
{
  char original[]="This is the original string";
  char buf[80];
  int  p1[2], p2[2];
    if (pipe(p1) != 0 || pipe(p2) != 0)
      perror("Any one PIPE is failed");

    if (fork() == 0)
    {
      close(p1[1]);
      close(p2[0]);

      read(p1[0], buf, sizeof(buf));
      reverse(buf);
      write(p2[1], buf, strlen(buf)+1);
      exit(0);
    }
    else
    {
      close(p1[0]);
      close(p2[1]);
      printf("parent is writing '%s' to pipe 1\n", original);

      write(p1[1], original, strlen(original)+1);
      read(p2[0], buf, sizeof(buf));
      printf("parent read '%s' from pipe 2\n", buf);
  }
}
```

Creates a pipe, an I/O channel that a process can use to communicate with another process (in the same process or another process), or in some cases with itself. Data is written into one end of the pipe and read from the other. **fdinfo**[2] points to a memory area where pipe() can store two file

descriptors. pipe() stores a file descriptor for the input end of the pipe in **fdinfo**[1], and stores a file descriptor for the output end of the pipe in **fdinfo**[0]. Thus, processes can read from **fdinfo**[0] and write to fdinfo[1]. Data written to **fdinfo**[1] is read from **fdinfo**[0] on a first-in-first-out (FIFO) basis.

=== * ===

more

- more is a filter for paging through text one screenful at a time
- Use the more command to read a file:
  - **more filename**
- Press Space bar / Return Key to read the remaining parts of the file.
- Press q to quit viewing contents

=== * ===

less

- **less** is the opposite of more command
- Use the more command to read a file:
  - less filename
- Searching can be done by typing /**searchkey**, will highlights the searched words
- Example: **less test.c**
  - /**printf**

=== * ===

cmp (Compare)

- compare two files byte by byte
- Syntax: cmp -b file1 file2

=== * ===

head and tail

- Output the first/last part of files (by Default first/last 10 lines)
  - **head filename**
  - **tail filename**

- **Example1**:
  - **head -5 a.txt** **will display first 5 lines of a.txt**
  - **tail -5 a.txt** **will display last 5 lines of a.txt**

=== * ===

File Permissions

- There are **three** users in linux namely **owner(u), group(g), others(o)**.
  - Note: For all users can be identified with letter **a**
- All three uses will have three possible permissions namely **read(r), write(w), execute(e)**
- Numbers assigned for permissions are **read(4), write(2), execute(1)**
- Command used to change permission of file is **chmod**
- **Syntax**: **chmod** **filePermissionPattern** **filename**
- **Permission Table:**
  - **Octal Table:**

binary	octal	permissions
000	0	- - -
001	1	- - x
010	2	- w -
011	3	- w x

binary	octal	permissions
100	4	r - -
101	5	r - x
110	6	r w -
111	7	r w x

- **Example**: **command:** **ls -lh**

Output:

-rw-rw-r-- 1 admincs admincs 805 Mar 28 13:42 b.c

drwx------ 2 admincs admincs 4.0K Mar 28 09:12 OS

-rwxrw-r-- 1 admincs admincs 1.9M Mar 28 08:45 output.pdf

drwxrwxr-x 2 admincs admincs 4.0K Jan 23 2021 Scratch Projects

Explanation

First letter (-) indicates file (in this example b.c, output.pdf are files)

First letter (d) indicates item is directory (in this example OS, Scrach Projects are Directories)

Next 9 characters indicates File Permissions for different uses

In this example, file **output.pdf** has permission **r w x r w – r – –** which indicates

- Owner has r w x Permissions, means file owner can read, write and execute the file

- Group users has r w – Permissions, means Group users can only read and write the File contents
- Other users has r – – Permissions, means Other users can only read the file contents

Examples

- add execution permission to group and other users
  - chmod go+x output.pdf
  - ls output.pdf -l
  - -rwxrwxr-x 1 admincs admincs 1.9M Mar 28 08:45 output.pdf
- remove write permissions to group users
  - chmod g-w output.pdf
  - ls output.pdf -l
  - -rwxr-xr-x 1 admincs admincs 1.9M Mar 28 08:45 output.pdf
- Keep only read permission to all users
  - chmod a=r output.pdf
  - ls output.pdf -l
  - -r--r--r-- 1 admincs admincs 1.9M Mar 28 08:45 output.pdf
- Adding execution to all users
  - chmod a+x output.pdf
  - ls output.pdf -l
  - -r-xr-xr-x 1 admincs admincs 1.9M Mar 28 08:45 output.pdf
- giving only writing permisin to other users (all settings may be not valid)
  - chmod o=w output.pdf
  - ls output.pdf -l
  - -r-xr-x-w- 1 admincs admincs 1.9M Mar 28 08:45 output.pdf
- give user all permissin, group users read and write, and other user only read permissions
  - chmod u=rwx,g=rw,o=r output.pdf
  - ls output.pdf -l
  - -rwxrw-r-- 1 admincs admincs 1.9M Mar 28 08:45 output.pdf

Working with numbers in chmod

Syntax: chmod chmodnumber filename

Example 1: if user wants rwxrw-r-- format set number to 764

- 777 = rwxrwxrwx
- 765 = rwxrw-r-x
- 654 = rw-r-xr--

Example 2:

- chmod 777 output.pdf will set all permissions to all users
- chmod 765 output.pdf will set rwxrw-r-x to users
- chmod 654 output.pdf will set rw-r-xr-- to users

=== * ===

umask

- While creating a file or directory, by default a set of permissions are applied. These default permissions are viewed by **umask** command.
- For safety reasons all Unix systems doesn't provide execution permission to newly created files.
- Adding execution permission is upto user.
  - **umask**

    Output: **0002**

Example 1:

- $ touch a.txt
- $ ls -l a.txt
- **-rw-rw-r--** 1 admincs admincs 28 Mar 28 14:32 a.txt
  - the default permission for file is (u,g,o) = (6-0, 6-0, 6-2) = (6,6,4). hence it is **r w – r w – r – –**

Example 2:

mkdir -m

- The 'mkdir -m' command can be used to set the mode.
- **Syntax:** mkdir -m <mode> <fileName>
- **Example:**
  - mkdir -m 777 new1
  - mkdir -m 000 new2
    - **ls -ld new1 new2**
    - d**rwxrwxrwx** 2 admincs admincs 4096 Mar 28 14:37 new1
    - d**- - - - - - - - -** 2 admincs admincs 4096 Mar 28 14:37 new2

=== * ===

File Compression and Decompression:

Compression reduces the size of an application or document for storage or transmission. Compressed files are smaller, download faster, and easier to transport. **Decompression** or expansion restores the document or application to its original size.

Compressing files:

Syntax	Description	Example(s)
gzip {filename}	• Gzip compress the size of the given files using Lempel-Ziv coding (LZ77). • Whenever possible, each file is replaced by one with the extension .gz.	gzip mydata.doc gzip *.jpg ls -l
bzip2 {filename}	• bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. • Compression is generally considerably better than that achieved by bzip command (LZ77/LZ78-based compressors). • Whenever possible, each file is replaced by one with the extension .bz2.	bzip2 mydata.doc bzip2 *.jpg ls -l
zip {.zip-filename} {filename-to-compress}	• zip is a compression and file packaging utility for Unix/Linux. • Each file is stored in single .zip {.zip-filename} file with the extension .zip.	zip mydata.zip mydata.doc zip data.zip *.doc ls -l
tar -zcvf {.tgz-file} {files} **tar -jcvf {.tbz2-file} {files}**	• The GNU tar is archiving utility but it can be use to compressing large file(s). • GNU tar supports both archive compressing through gzip and bzip2. • If user have more than 2 files then it is recommended to use tar instead of gzip or bzip2. **-z**: use gzip compress **-j**: use bzip2 compress	tar -zcvf data.tgz *.doc tar -zcvf pics.tar.gz *.jpg *.png tar -jcvf data.tbz2 *.doc ls -l

Decompressing files

Syntax	Description	Example(s)
gzip -d {.gz file} **gunzip {.gz file}**	• Decompressed a file that is created using gzip command. • File is restored to their original form using this command.	gzip -d mydata.doc.gz gunzip mydata.doc.gz
bzip2 -d {.bz2-file} **bunzip2 {.bz2-file}**	• Decompressed a file that is created using bzip2 command. • File is restored to their original form using this command.	bzip2 -d mydata.doc.bz2 gunzip mydata.doc.bz2
unzip {.zip file}	Extract compressed files in a ZIP archive.	unzip file.zip unzip data.zip resume.doc
tar -zxvf {.tgz-file} **tar -jxvf {.tbz2-file}**	Untar or decompressed a file(s) that is created using tar compressing through gzip and bzip2 filter	tar -zxvf data.tgz tar -zxvf pics.tar.gz *.jpg tar -jxvf data.tbz2

List the contents of an archive/compressed file

Some time user just wanted to look at files inside an archive or compressed file. Then all of the above command supports file list option.

Syntax	Description	Example(s)
gzip -l {.gz file}	List files from a GZIP archive	gzip -l mydata.doc.gz
unzip -l {.zip file}	List files from a ZIP archive	unzip -l mydata.zip
tar -ztvf {.tar.gz} **tar -jtvf {.tbz2}**	List files from a TAR archive	tar -ztvf pics.tar.gz tar -jtvf data.tbz2

Important options in **tar** manual

- **-c : create archive**
- -x : extract
- -j : use bzip2 compress
- **-z** : use gzip compress
- -v : verbose mode
- -f : use archive file or device ARCHIVE

=== * ===

Text Processing Commands

Commands affecting text and text files

sort

- **sort** command is used to sort a file, arranging the records in a particular order.
- By default, the sort command sorts file assuming the contents are ASCII.
- Using options in the sort command can also be used to sort numerically.
- The sort command can also sort by items not at the beginning of the line, ignore case sensitivity, and return whether a file is sorted or not.
- Sorting is done based on one or more sort keys extracted from each line of input.
- By default, the entire input is taken as the sort key.
- Blank space is the default field separator.

Syntax:

sort [OPTIONS] filename

Options used:

- -o : used to save sorted content in specified file
  - sort -o output.txt inputfile.txt
- r : sort the file contents in descending order
  - sort -r inputfile.txt
- -n : if the file contains numbers, based on number values, file contents can be sorted.
  - sort -n inputfile.txt
- -k : sort the file contents based on kth field values
  - sort -k 2 inputfile.txt // sorts file contents based on values in 2nd field.
- -u : will remove duplicate entries in file while sorting.

=== * ===

uniq

- uniq filter removes duplicate lines from a sorted file. It is often seen in a pipe coupled with sort.
- **uniq -c filename**
- **removes the duplicates lines as well as displys frequency of duplicate lines.**

== * ===

cut

- A tool for extracting fields from files.
- Important options are -d, which specifies delimiter to separate fields, -f which indicates field numbers
- **Example 1: cut -d" " -f1-4,6,8 *FileName***
  - here contains information separated by space. This command displays contents in fields 1,2,3,4 and 6 and 8
- **Example 2: cut -d: -f1-4,6,8 *FileName***
  - Same as above but the fields in file are seperated by : (colon)

== * ===

join

- Join allows merging two files in a meaningful fashion, which essentially creates a simple version of a relational database.
- The **join** command operates on exactly two files, but pastes together only those lines with a common tagged field (usually a numerical label), and writes the result to stdout.
- The files to be joined should be sorted according to the tagged field for the matchups to work properly.

== * ===

head and tail

- **head/tail**: output the first/last part of files (by Default 10 lines)
  - **head *FileName***
  - **tail *FileName***
- **Example 1**: **head -5 a.txt** **will display first 5 lines of a.txt**
  tail -5 a.txt **will display last 5 lines of a.txt**

=== * ===

grep

- A multi-purpose file search tool that uses Regular Expressions. (Global Regular Expression Pring).
  - **grep pattern [*FileName...*]**
- options used with grep are:
  - The -i option causes a **case-insensitive** search.
  - The -w option matches only **whole words**.

- The -r (recursive) option searches files in the current working directory and all subdirectories below it.
- The -n option lists the matching lines, together with **line numbers**.
- The -c (--count) option gives a numerical count of matches.

- **Example 1:**
  - To see every process on the system using BSD syntax:
  - ps ax
  - But search only word **swap** associated with the processes.
- **Output:**
  - **ps ax | grep swap**
  - 55 ? S 0:00 [k**swap**d0]
  - 4206 pts/0 S+ 0:00 grep --color=auto **swap**

=== * ===

wc

- wc gives a "word count" on a file or I/O stream:
- Important Options Used:
  - **wc -l** gives only the line count.
  - **wc -w** gives only the word count.
  - **wc -c** gives only the byte count.
- **Syntax:** **wc -lwc *FileName***
- **Example:**
  - **wc -lwc temp2.txt**
- **Output:**
  - 3 6 18 temp2.txt
  - Says there 3 Lines, 6 Wrods and 18 Characters altogher in a file temp2.txt

=== * ===

tr

- character translation filter.
- **Example 1: tr "A-Z" "*" <filename**
  - will translate all the UPPER CASE characters to * in a file
- **Example 2: tr "0-9" "A" <filename**
  - will translate all the DIGITS to ‘A’ in a file

- The -c "complement" option inverts the character set to match.
- **Example 3: echo "acfdeb123" | tr -c b-d +**
  - +c+d+b++++
- **Example 4: echo "abcd2ef1" | tr '[:alpha:]' -**
  - ----2--1
- same as **echo "abcd2ef1" | tr 'a-zA-Z' -**
  - ----2--1

=== * ===

fold

- A filter that wraps lines of input to a specified width. This is especially useful with the -s option, which breaks lines at word spaces.
- **Example 1: fold -w 3 *FileName***
  - Will wrap each line for every three characters
- **Example 2: fold - -s -w 3**
  - Will wait for the user to enter text, after return key, the text will be wraped to every three characters.

=== * ===

nl

- Line numbering filter: **nl filename** lists filename to stdout, but inserts consecutive numbers at the beginning of each non-blank line.
- If filename omitted, operates on stdin.

=== * ===

Work on Practice Session

Work on Practice Session