**Name: Aditya Ramnath Nawale**

**Division: AI&DS-B**

**Roll No: 38**

**Subject: OS**

**Batch: B2**

**Experiment No:2**

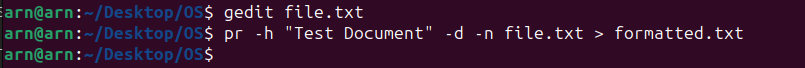
**Title: Advance Linux Command**

**Problem Statement: Execution of advance Linux commands.**

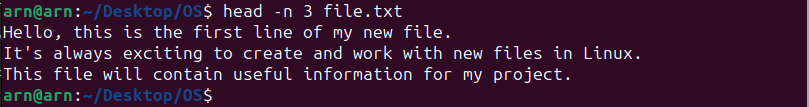
**Solution:**

* **Simple Filter:**

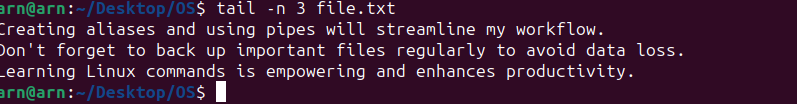
1. **pr <file>: Paginates the file for printing, preparing it for a printer. It formats the text into multiple columns and adds headers, line numbers, and page breaks. Ex: pr –h “test” –d –n fname**

****

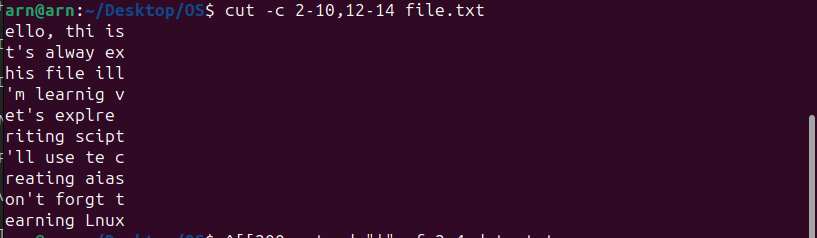
1. **head <file>: Display first 10 lines of file. Ex: head –n -3 fname**

****

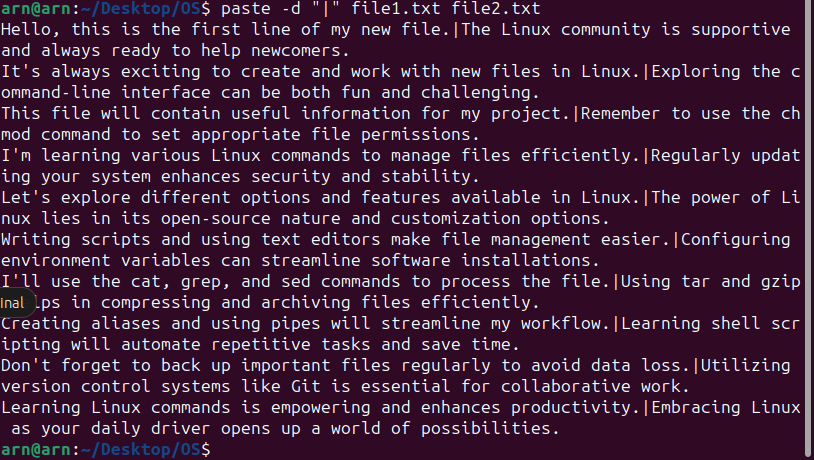
1. **tail <file>: To display last 10 lines of file. Ex: tail -3 fname ; tail –c 100 fname**

****

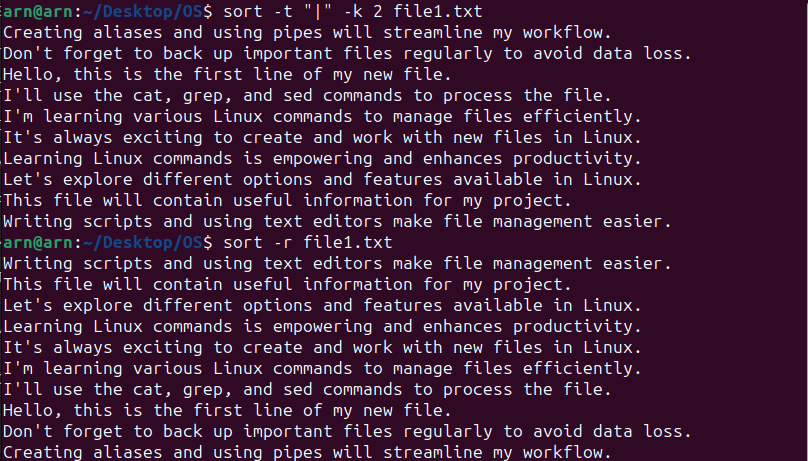
1. **cut <file>: Splits the lines of the file vertically by specific characters (delimiter) or fields. Example: To extract characters 2 to 10, and 12 to 14 from the file file.txt, you can use the -c option. Ex: cut –c 2-10,12-14 fname , cut –d “|” –f 2,4 fname**

****

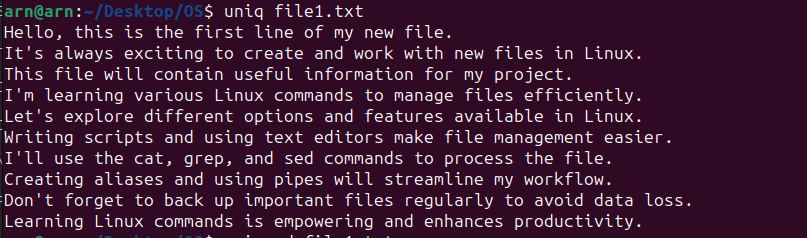
1. **paste <file1> <file2>: Combines two files vertically (side by side) with optional delimiter. Ex: paste –d “|” fname1 fname2**

****

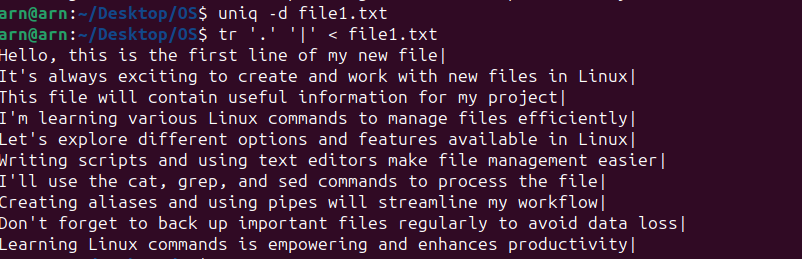
1. **sort <file>: Sorts the lines of the file in ascending order. Ex: sort –t”|” –k 2 fname sort –r fname**

****

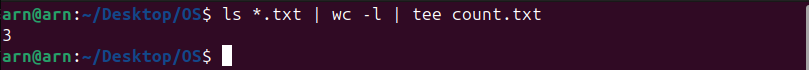
1. **uniq <file>: Locate repeated & nonrepeated lines. Ex: uniq fname; uniq –d fname**

****

1. **tr ch1 ch2 < <file1>: To translate occurrence of ch1 by ch2. Ex: tr ‘|’ ‘+’ < fname1**

****

1. **tee: read from standard input and write to standard output and files. Ex: ls \*.txt | wc -l | tee count.txt**

****

* **File permission:**

1. **Changing permission relative manner:**

**In Linux, you can change file permissions using the chmod command. File permissions define who can read, write, and execute a file. The syntax of the chmod command is as follows:**

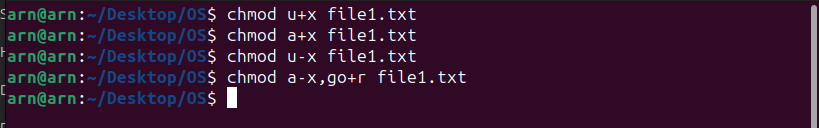
**chmod [category][operation][permission] <file>**

* **[category]: Represents the permission category you want to modify. It can be u for the user/owner, g for the group, o for others, or a for all (user, group, and others).**
* **[operation]: Defines the action you want to perform on the permissions. It can be + to add a permission, - to remove a permission, or = to set the permission explicitly.**
* **[permission]: Represents the specific permission you want to add, remove, or set.**

**The common permissions are r for read, w for write, and x for execute.**

**Now, let's explain the examples:**

1. **chmod u+x fname: This command adds execute permission to the user/owner (u) of the file fname. After running this command, the owner will be able to execute the file.**
2. **chmod a+x fname: This command adds execute permission to all (a) users of the file fname. After running this command, the owner, group members, and others will be able to execute the file.**
3. **chmod u-x fname: This command removes the execute permission from the user/owner (u) of the file fname. After running this command, the owner will no longer be able to execute the file.**
4. **chmod a-x,go+r fname: This command removes the execute permission from all (a), and then adds read permission (+r) to the group (g) and others (o) for the file fname. After running this command, everyone will be able to read the file, but no one will be able to execute it.**

****

1. **Changing permission absolute manner:**

**In Linux, you can change file permissions using the chmod command in an absolute manner. This means you specify the exact numeric value representing the desired permissions for the file. Each permission is assigned a numeric value:**

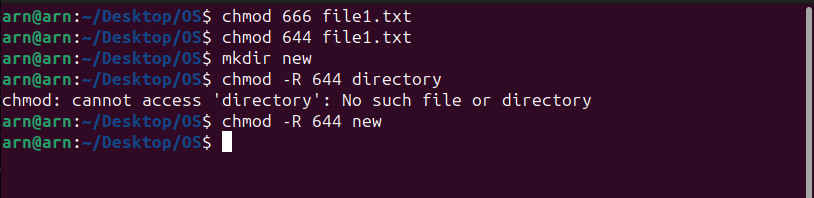
* **Read (r) = 4**
* **Write (w) = 2**
* **Execute (x) = 1**

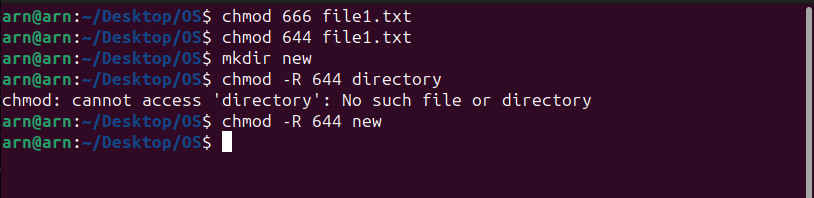
**To change permissions in absolute manner, you sum up the numeric values corresponding to the permissions you want to set. Here are some examples:**

1. **chmod 666 fname: This command sets read and write permissions for the user, group, and others on the file fname. All users will be able to read and write to the file, but no one will have execute permission.**
2. **chmod 644 fname: This command sets read-only permission for the user and read-only permission for the group and others on the file fname. Users can read the file, but they cannot modify it or execute it.**
3. **chmod -R 644 directory: The -R option means "recursively." This command changes the permissions of the entire directory and its contents to read-only for the user and read-only for the group and others. It's important to note that changing the permissions recursively can affect all files and subdirectories within the specified directory.**

**For example, if you have a directory named "employees" containing multiple files and subdirectories, the -R option will change the permissions of all files and subdirectories within the "employees" directory to read-only for the user, group, and others.**

**Remember that file permissions are represented using three digits in octal notation. The first digit represents the permissions for the user/owner, the second digit for the group, and the third digit for others. Each digit is the sum of the numeric values of the permissions you want to set (4 for read, 2 for write, and 1 for execute).**

****

****

* **Change owner & group**

**In Linux, the chown command is used to change the owner and/or group of one or more files or directories, while the chgrp command is used to change the group ownership of files and directories. Here are the explanations and examples for both commands:**

**chown command:**

**Syntax: chown [options] owner files**

**owner: Represents the new owner that you want to set for the specified files or directories.**

**files: Specifies the file or directory for which you want to change the ownership.**

**Example: To change the owner of the file file.txt to user xyz, you would use the following command:**

**chown xyz file.txt**

**This command changes the ownership of file.txt to the user with the username xyz. Note that you usually need to have administrative privileges (root or sudo) to change the ownership of files that you do not own.**

1. **chgrp command:**

* **Syntax: chgrp [options] group files**
* **group: Represents the new group that you want to set for the specified files or directories.**
* **files: Specifies the file or directory for which you want to change the group ownership.**

**Example: To change the group of the file file.txt to group xyz, you would use the following command:**

**chgrp xyz file.txt**

**This command changes the group ownership of file.txt to the group with the name xyz. Like with chown, you usually need administrative privileges to change the group ownership of files that you do not own.**

**Both chown and chgrp commands can also accept multiple filenames as arguments to change the ownership or group ownership of multiple files at once.**

**Please note that changing ownership or group ownership of system files or critical directories should be done with caution and with proper authorization, as it can impact the functioning of the system or cause permission-related issues.**

* **Redirection:**

**Redirection in Linux provides a powerful way to control input and output streams of commands. It allows you to manipulate where the standard input, output, and error of a command come from and go to. Redirection operators are used to perform these actions. Here are some common examples of how redirection can be used to control command-line input and output:**

1. **Output Redirection (">"): The ">" operator is used to redirect the output of a command to a file. It creates or overwrites the specified file with the output of the command.**

**Example: Suppose you want to store the output of the "ls" command (list files in the current directory) in a file named "filelist.txt."**

**ls > filelist.txt**

**The file "filelist.txt" will be created (or overwritten if it already exists), and the output of the "ls" command will be written to it.**

1. **Append Output (">>"): The ">>" operator is used to append the output of a command to an existing file.**

**Example: To continuously add the output of the "date" command to a log file named "logfile.txt," you can use the following:**

**date >> logfile.txt**

**The output of the "date" command will be appended to "logfile.txt" each time you run the command.**

1. **Input Redirection ("<"): The "<" operator is used to redirect input to a command from a file instead of typing it manually.**

**Example: If you have a file named "input.txt" containing some data, you can pass the content of that file as input to a command like "grep" to search for a specific pattern:**

**grep "pattern" < input.txt**

1. **Redirecting Standard Error (2>): By default, only standard output is redirected using ">", but you can also redirect standard error (stderr) using the "2>" operator.**

**Example: If you want to redirect the error output of a command to an error log file named "error.log," you can use the following:**

**command\_not\_found 2> error.log**

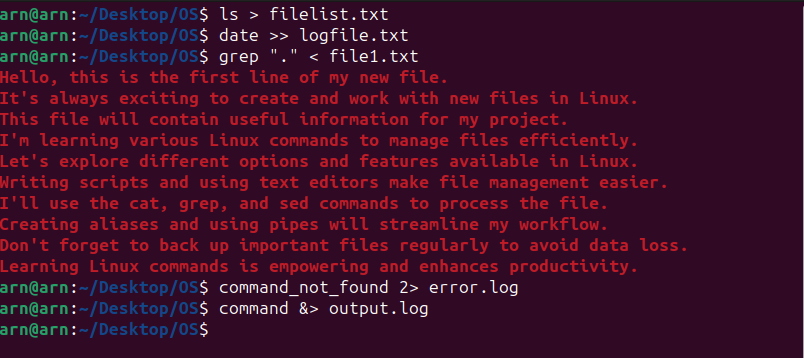
**Any error message generated by the "command\_not\_found" will be written to "error.log."**

1. **Redirecting Both Output and Error (2>&1): To redirect both standard output and standard error to the same file, you can use "2>&1."**

**Example: To redirect both output and error of a command to a log file named "output.log":**

**command &> output.log**

**These are some of the examples of how redirection in Linux provides powerful control over the input and output of commands, making the command-line experience more versatile and efficient.**

****

* **Pipe:**

**Pipe (represented by the vertical bar |) in Linux connects multiple commands in a way that the output of the first command becomes the input for the second command. It allows you to combine various commands and perform complex operations efficiently. Here are some examples to illustrate how the pipe works:**

**Example: ls -l | more**

**This command lists the files and directories in the current directory using the ls -l command, but instead of displaying all the results on the screen, it pipes the output to the more command. The more command allows you to scroll through the output one screen at a time, making it easier to read long directory listings.**

**Example: cat file1 file2 | sort > file3**

**In this example, the cat command is used to concatenate the contents of file1 and file2, and the output is piped to the sort command. The sort command sorts the lines alphabetically, and the sorted result is then redirected to a new file called file3.**

**Suppose file1 contains the following content:**

**apple**

**banana**

**And file2 contains the following content:**

**grape**

**orange**

**After running the command, file3 will contain the following content:**

**apple**

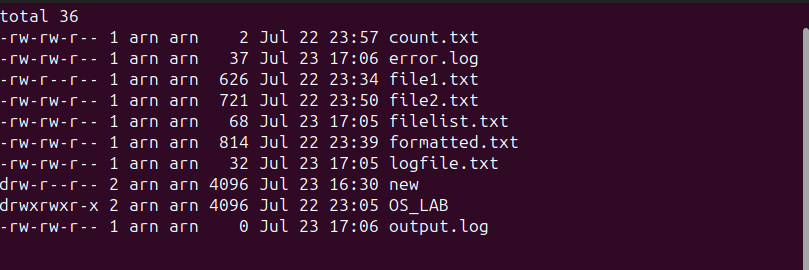
**banana**

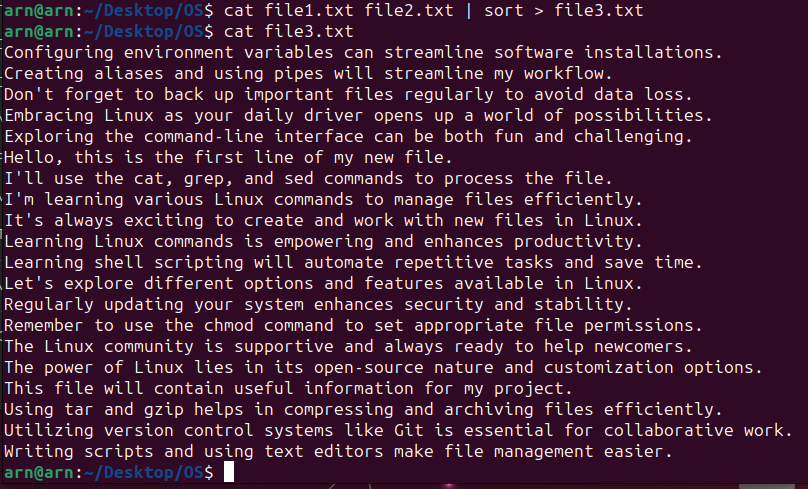
**grape**

**orange**

**The sort command alphabetically sorted the lines from both file1 and file2 and stored the result in file3.**

**Pipes are incredibly powerful as they enable you to chain multiple commands together, allowing you to process and transform data effectively on the command line. It's a fundamental concept in Linux and Unix-like operating systems that enhances the flexibility and efficiency of command-line usage.**

****

****

* **Grep:**

**The `grep` command in Linux is a powerful tool for searching and pattern matching in files. It allows you to search for one or more patterns in one or multiple files, and it can perform plain string, basic regular expression, and extended regular expression searching. Here are some common options and examples of how `grep` works:**

**1. Searching for a Pattern:**

**Syntax: `grep pattern file`**

**Example: Suppose you want to search for the word "apple" in a file named `fruits.txt`.**

**grep "apple" fruits.txt**

**This will display all lines in the file `fruits.txt` that contain the word "apple."**

**2. Ignoring Case (Case Insensitive Search) (-i):**

**Syntax: `grep -i pattern file`**

**Example: To search for the word "orange" in a file named `fruits.txt` while ignoring case:**

**grep -i "orange" fruits.txt**

**This will display all lines in the file `fruits.txt` that contain the word "orange," regardless of whether it is in uppercase or lowercase.**

**3. Displaying Line Numbers (-n):**

**Syntax: `grep -n pattern file`**

**Example: To search for the word "banana" in a file named `fruits.txt` and display line numbers along with occurrences:**

**grep -n "banana" fruits.txt**

**This will display the lines in the file `fruits.txt` containing "banana" along with their corresponding line numbers.**

**4. Counting Occurrences (-c):**

**Syntax: `grep -c pattern file`**

**Example: To count the number of occurrences of the word "grape" in a file named `fruits.txt`:**

**grep -c "grape" fruits.txt**

**This will display the total count of occurrences of "grape" in the file.**

**5. Displaying List of File Names (-l):**

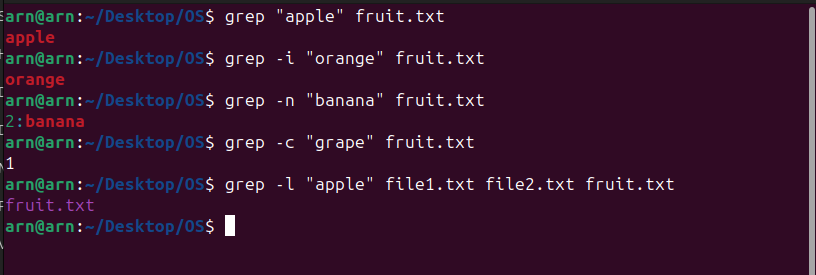
**Syntax: `grep -l pattern files...`**

**Example: To search for the word "apple" in multiple files (`file1.txt`, `file2.txt`, and `file3.txt`) and display only the names of the files containing the pattern:**

**grep -l "apple" file1.txt file2.txt file3.txt**

**This will display the names of the files that contain the word "apple."**

**These are some common options and examples of using the `grep` command in Linux. It's a versatile tool that can be used for a wide range of search and pattern matching tasks on the command line.**

****