1. File and Directory Operations Commands

File and directory operations are fundamental in working with the Linux operating system. Here are some commonly used File and Directory Operations commands:

Command	Description	Options	Examples
ls	List files and directories.	• **-l**: Long format listing. • **-a**: Include hidden files hidden ones • **-h**: Human-readable file sizes.	• **ls -l** displays files and directories with detailed information. • **ls -a** shows all files and directories, including • **ls -lh** displays file sizes in a human-readable format.
cd	Change directory.		• **cd /path/to/directory** changes the current directory to the specified path.
pwd	Print current working directory.		• **pwd** displays the current working directory.
mkdir	Create a new directory.		• **mkdir my_directory** creates a new directory named “my_directory”.
rm	Remove files and directories.	• **-r**: Remove directories recursively. • **-f**: Force removal without confirmation.	• **rm file.txt** deletes the file named “file.txt”. • **rm -r my_directory** deletes the directory “my_directory” and its contents.

Command	Description	Options	Examples
			- **rm -f file.txt** forcefully deletes the file “file.txt” without confirmation.
cp	Copy files and directories.	- **-r**: Copy directories recursively.	- **cp -r directory destination** copies the directory “directory” and its contents to the specified destination. - **cp file.txt destination** copies the file “file.txt” to the specified destination.
mv	Move/rename files and directories.		- **mv file.txt new_name.txt** renames the file “file.txt” to “new_name.txt”. - **mv file.txt directory** moves the file “file.txt” to the specified directory.
touch	Create an empty file or update file timestamps.		- **touch file.txt** creates an empty file named “file.txt”.
cat	View the contents of a file.		- **cat file.txt** displays the contents of the file “file.txt”.

Command	Description	Options	Examples
head	Display the first few lines of a file.	• **-n**: Specify the number of lines to display.	• **head file.txt** shows the first 10 lines of the file "file.txt". • **head -n 5 file.txt** displays the first 5 lines of the file "file.txt".
tail	Display the last few lines of a file.	• **-n**: Specify the number of lines to display.	• **tail file.txt** shows the last 10 lines of the file "file.txt". • **tail -n 5 file.txt** displays the last 5 lines of the file "file.txt".
ln	Create links between files.	• **-s**: Create symbolic (soft) links.	• **ln -s source_file link_name** creates a symbolic link named "link_name" pointing to "source_file".
find	Search for files and directories.	• **-name**: Search by filename. • **-type**: Search by file type.	• **find /path/to/search -name "*.txt"** searches for all files with the extension ".txt" in the specified directory.

2. File Permission Commands

File permissions on Linux and Unix systems control access to files and directories. There are three basic permissions: read, write, and execute. Each permission can be granted or denied to three different categories of users: the owner of the file, the members of the file's group, and everyone else.

Here are some file permission commands:

Command	Description	Options	Examples
chmod	Change file permissions.	• **u**: User/owner permissions. • **g**: Group permissions. • **o**: Other permissions. • **+**: Add permissions. • **–**: Remove permissions. • **=**: Set permissions explicitly.	• **chmod u+rwx file.txt** grants read, write, and execute permissions to the owner of the file.
chown	Change file ownership.		• **chown user file.txt** changes the owner of “file.txt” to the specified user.
chgrp	Change group ownership.		• **chgrp group file.txt** changes the group ownership of “file.txt” to the specified group.
umask	Set default file permissions.		• **umask 022** sets the default file permissions to read and write for the owner, and read-only for group and others.

3. File Compression and Archiving Commands

Here are some file compression and archiving commands in Linux:

Commands	Description	Options	Examples
[tar]()	Create or extract archive files.	- **-c**: Create a new archive. - **-x**: Extract files from an archive. - **-f**: Specify the archive file name. - **-v**: Verbose mode. - **-z**: Compress the archive with gzip. - **-j**: Compress the archive with bzip2.	- **tar -czvf archive.tar.gz files/** creates a compressed tar archive named "archive.tar.gz" containing the files in the "files/" directory.
[gzip]()	Compress files.	- **-d**: Decompress files.	- **gzip file.txt** compresses the file "file.txt" and renames it as "file.txt.gz".
[zip]()	Create compressed zip archives.	- **-r**: Recursively include directories.	- **zip archive.zip file1.txt file2.txt** creates a zip archive named "archive.zip" containing "file1.txt" and "file2.txt".

4. Process Management Commands

In Linux, process management commands allow you to monitor and control running processes on the system. Here are some commonly used process management commands:

Commands	Description	Options	Examples
[ps]()	Display running processes.	- **-aux**: Show all processes.	- **ps aux** shows all running

Commands	Description	Options	Examples
			processes with detailed information.
top	Monitor system processes in real-time.		- **top** displays a dynamic view of system processes and their resource usage.
kill	Terminate a process.	- **-9**: Forcefully kill a process.	- **kill PID** terminates the process with the specified process ID.
pkill	Terminate processes based on their name.		- **pkill process_name** terminates all processes with the specified name.
pgrep	List processes based on their name.		- **pgrep process_name** lists all processes with the specified name.
grep	used to search for specific patterns or regular expressions in text files or streams and display matching lines.	- **-i**: Ignore case distinctions while searching. - **-v**: Invert the match,	- **grep -i "hello" file.txt** - **grep -v "error" file.txt** - **grep -r "pattern" directory/**

Commands	Description	Options	Examples
		displaying non-matching lines. - **-r or -R**: Recursively search directories for matching patterns. - **-l**: Print only the names of files containing matches. - **-n**: Display line numbers alongside matching lines. - **-w**: Match whole words only, rather than partial matches. - **-c**: Count the number of matching lines instead of displaying them. - **-e**: Specify multiple patterns to search for. - **-A**: Display lines after the matching line. - **-B**: Display lines before	- **grep -l “keyword” file.txt** - **grep -n “pattern” file.txt** In these examples we are extracting our desirec output from filename (file.txt)

Commands	Description	Options	Examples
		the matching line. - **-C**: Display lines both before and after the matching line.	

5. System Information Commands

In Linux, there are several commands available to gather system information. Here are some commonly used system information commands:

sudCommand	Description	Options	Examples
uname	Print system information.	- **-a**: All system information.	- **uname -a** displays all system information.
whoami	Display current username.		- **whoami** shows the current username.
df	Show disk space usage.	- **-h**: Human-readable sizes.	- **df -h** displays disk space usage in a human-readable format.
du	Estimate file and directory sizes.	- **-h**: Human-readable sizes. - **-s**: Display total size only.	- **du -sh directory/** provides the total size of the specified directory.
free	Display memory usage information.	- **-h**: Human-readable sizes.	- **free -h** displays memory usage in a human-readable format.

sudCommand	Description	Options	Examples
uptime	Show system uptime.		- **uptime** shows the current system uptime.
lscpu	Display CPU information.		- **lscpu** provides detailed CPU information.
lspci	List PCI devices.		- **lspci** List PCI devices.
lsusb	List USB devices.		- **lsusb** lists all connected USB devices.

6. Networking Commands

In Linux, there are several networking commands available to manage and troubleshoot network connections. Here are some commonly used networking commands:

Command	Description	Examples
ifconfig	Display network interface information.	- **ifconfig** shows the details of all network interfaces.
ping	Send ICMP echo requests to a host.	- **ping google.com** sends ICMP echo requests to “google.com” to check connectivity.
netstat	Display network connections and statistics.	- **netstat -tuln** shows all listening TCP and UDP connections.

Command	Description	Examples
ss	Display network socket information.	- **ss -tuln** shows all listening TCP and UDP connections.
ssh	Securely connect to a remote server.	- **ssh user@hostname** initiates an SSH connection to the specified hostname.
scp	Securely copy files between hosts.	- **scp file.txt user@hostname:/path/to/destination** securely copies “file.txt” to the specified remote host.
wget	Download files from the web.	- **wget http://example.com/file.txt** downloads “file.txt” from the specified URL.
curl	Transfer data to or from a server.	- **curl http://example.com** retrieves the content of a webpage from the specified URL.

7. IO Redirection Commands

In Linux, IO (Input/Output) redirection commands are used to redirect the standard input, output, and error streams of commands and processes. Here are some commonly used IO redirection commands:

Command	Description
cmd < file	Input of cmd is taken from file.
cmd > file	Standard output (stdout) of cmd is redirected to file.
cmd 2> file	Error output (stderr) of cmd is redirected to file.
cmd 2>&1	stderr is redirected to the same place as stdout.

Command	Description
cmd1 <(cmd2)	Output of cmd2 is used as the input file for cmd1.
cmd > /dev/null	Discards the stdout of cmd by sending it to the null device.
cmd &> file	Every output of cmd is redirected to file.
cmd 1>&2	stdout is redirected to the same place as stderr.
cmd >> file	Appends the stdout of cmd to file.

8. Environment Variable Commands

In Linux, environment variables are used to store configuration settings, system information, and other variables that can be accessed by processes and shell scripts. Here are some commonly used environment variable commands:

Command	Description
export VARIABLE_NAME=value	Sets the value of an environment variable.
echo $VARIABLE_NAME	Displays the value of a specific environment variable.
env	Lists all environment variables currently set in the system.
unset VARIABLE_NAME	Unsets or removes an environment variable.
export -p	Shows a list of all currently exported environment variables.
env VAR1=value COMMAND	Sets the value of an environment variable for a specific command.

Command	Description
printenv	Displays the values of all environment variables.

9. User Management Commands

In Linux, user management commands allow you to create, modify, and manage user accounts on the system. Here are some commonly used user management commands:

Command	Description
who	Show who is currently logged in.
sudo adduser username	Create a new user account on the system with the specified username.
finger	Display information about all the users currently logged into the system, including their usernames, login time, and terminal.
sudo deluser USER GROUPNAME	Remove the specified user from the specified group.
last	Show the recent login history of users.
finger username	Provide information about the specified user, including their username, real name, terminal, idle time, and login time.
sudo userdel -r username	Delete the specified user account from the system, including their home directory and associated files. The -r option ensures the removal of the user’s files.
sudo passwd -l username	Lock the password of the specified user account, preventing the user from logging in.

Command	Description
su – username	Switch to another user account with the user's environment.
sudo usermod -a -G GROUPNAME USERNAME	Add an existing user to the specified group. The user is added to the group without removing them from their current groups.

10. Shortcuts Commands

There are many shortcuts commands in Linux that can help you be more productive. Here are a few of the most common ones:

10.1: Bash Shortcuts Commands:

Navigation	Description	Editing	Description	History	Description
Ctrl + A	Move to the beginning of the line.	**Ctrl + U**	Cut/delete from the cursor position to the beginning of the line.	**Ctrl + R**	Search command history (reverse search).
Ctrl + E	Move to the end of the line.	**Ctrl + K**	Cut/delete from the cursor position to the end of the line.	**Ctrl + G**	Escape from history search mode.
Ctrl + B	Move back one character.	**Ctrl + W**	Cut/delete the word before the cursor.	**Ctrl + P**	Go to the previous command in history.

Navigation	Description	Editing	Description	History	Description
Ctrl + F	Move forward one character.	**Ctrl + Y**	Paste the last cut text.	**Ctrl + N**	Go to the next command in history.
Alt + B	Move back one word	**Ctrl + L**	Clear the screen.	**Ctrl + C**	Terminate the current command.
Alt + F	Move forward one word.				

10.2: Nano Shortcuts Commands:

File Operations	Description	Navigation	Description	Editing	Description	Search and Replace	Description
Ctrl + O	Save the file.	**Ctrl + Y**	Scroll up one page.	**Ctrl + K**	Cut/delete from the cursor position to the end of the line.	**Ctrl + W**	Search for a string in the text.
Ctrl + X	Exit Nano (prompt to save if	**Ctrl + V**	Scroll down one page.	**Ctrl + U**	Uncut/restore the last cut text.	**Alt + W**	Search and replace a string in the text.

File Operations	Description	Navigation	Description	Editing	Description	Search and Replace	Description
	modified).						
Ctrl + R	Read a file into the current buffer.	**Alt + \**	Go to a specific line number.	**Ctrl + 6**	Mark a block of text for copying or cutting.	**Alt + R**	Repeat the last search.
Ctrl + J	Justify the current paragraph.	**Alt + ,**	Go to the beginning of the current line.	**Ctrl + K**	Cut/delete the marked block of text.		
		Alt + .	Go to the end of the current line.	**Alt + 6**	Copy the marked block of text.		

10.3: VI Shortcuts Commands:

Command	Description
cw	Change the current word. Deletes from the cursor position to the end of the current word and switches to insert mode.
dd	Delete the current line.

Command	Description
x	Delete the character under the cursor.
R	Enter replace mode. Overwrites characters starting from the cursor position until you press the Escape key.
o	Insert a new line below the current line and switch to insert mode.
u	Undo the last change.
s	Substitute the character under the cursor and switch to insert mode.
dw	Delete from the cursor position to the beginning of the next word.
D	Delete from the cursor position to the end of the line.
4dw	Delete the next four words from the cursor position.
A	Switch to insert mode at the end of the current line.
S	Delete the current line and switch to insert mode.
r	Replace the character under the cursor with a new character entered from the keyboard.
i	Switch to insert mode before the cursor.
3dd	Delete the current line and the two lines below it.
ESC	Exit from insert or command-line mode and return to command mode.
U	Restore the current line to its original state before any changes were made.

Command	Description
~	Switch the case of the character under the cursor.
a	Switch to insert mode after the cursor.
C	Delete from the cursor position to the end of the line and switch to insert mode.

10.4: Vim Shortcuts Commands:

Normal Mode	Description	Command Mode	Description	Visual Mode	Description
i	Enter insert mode at the current cursor position.	**:w**	Save the file.	**v**	Enter visual mode to select text.
x	Delete the character under the cursor.	**:q**	Quit Vim.	**y**	Copy the selected text.
dd	Delete the current line.	**:q!**	Quit Vim without saving changes.	**d**	Delete the selected text.
yy	Copy the current line.	**:wq** or :x:*x*	Save and quit Vim.	**p**	Paste the copied or deleted text.
p	Paste the copied or	**:s/old/new/g**	Replace all occurrences		

Normal Mode	Description	Command Mode	Description	Visual Mode	Description
	deleted text below the current line.		of “old” with “new” in the file.		
u	Undo the last change.	**:set nu** or **:set number**	Display line numbers.		
Ctrl + R	Redo the last undo.				

Conclusion

In conclusion, Linux is a widely used operating system for development, and as a developer, you should have knowledge of Linux and its basic commands. In this Cheat Sheet, we covered all commands like creating directories, file compression and archiving, process management, system information, networking and more. In addition to that, this Linux Cheat Sheet is organized and categorized, making it easy for developers to quickly find the commands they need for specific use cases. By utilizing this resource, developers can enhance their productivity and efficiency in working with Linux, leading to smoother and more successful development projects.

> ***PS.*** *Don’t miss our other Python cheat sheet for data science that covers Scikit-Learn, Bokeh, Pandas and Python basics.*

Linux Commands Cheat Sheet – FAQs

What is Linux Cheat Sheet?

> *When your memory fails or you prefer not to rely on “linux –* ***help****?” in the Terminal, this linux cheat sheet comes to the rescue. It is hard to memorize all the important linux Commandsby heart, so print this out or save it to your desktop to resort to when you get stuck.*

What are the basics of Linux?

- ***Kernel***. *The base component of the OS. Without it, the OS doesn't work. …*
- ***System user space***. *The administrative layer for system-level tasks like configuration and software install. …*
- ***Applications***. *A type of software that lets you perform a task.*

What is 777 in Linux command?

*You might have heard of chmod 777. This command will **give read, write and execute permission to the owner, group and public**.*

How do I see what users are doing in Linux?

Using the w Command, *w command in Linux shows logged-in users and their activities.*

Sed(command in unix)

sed (Stream Editor) is a powerful command-line utility in Unix used for text manipulation and transformation. It reads input from files or standard input (streams) and applies various operations like search, replace, delete, and more.

Basic Syntax:

arduino
Copy code

```
sed [options] 'command' file
```

Common Use Cases of sed:

1. **Search and Replace (Substitution):** Replace the first occurrence of a string on each line.

   Example:

   bash
   Copy code

   ```
   sed 's/old_string/new_string/'  file.txt
   ```

Replace all occurrences of a string on each line by using the g (global) flag:

bash
Copy code
```
sed 's/old_string/new_string/g'  file.txt
```

2. **In-Place Replacement:** To edit the file directly and save the changes (without printing to the terminal), use the -i option.

   Example:

   bash
   Copy code
   ```
   sed -i 's/old_string/new_string/g'  file.txt
   ```

3. **Delete Lines:** Delete a specific line, e.g., the second line of a file.

   Example:

   bash
   Copy code
   ```
   sed '2d' file.txt
   ```

   Delete a range of lines, e.g., from line 2 to line 4.

   Example:

   bash
   Copy code
   ```
   sed '2,4d' file.txt
   ```

4. **Print Specific Lines:** Print only certain lines from a file (like line 2).

   Example:

   bash
   Copy code
   ```
   sed -n '2p' file.txt
   ```

   Print lines that match a pattern.

   Example:

bash
Copy code
```
sed -n '/pattern/p' file.txt
```

5. **Insert or Append Text:** Insert a line before a specific line (before line 3).

   Example:

   bash
   Copy code
   ```
   sed '3i\This is the inserted line.' file.txt
   ```

   Append a line after a specific line (after line 3).

   Example:

   bash
   Copy code
   ```
   sed '3a\This is the appended line.' file.txt
   ```

6. **Replace a Word Only at the Beginning or End of a Line:** To replace a word at the beginning of a line:

   bash
   Copy code
   ```
   sed 's/^old_word/new_word/'  file.txt
   ```

   To replace a word at the end of a line:

   bash
   Copy code
   ```
   sed 's/old_word$/new_word/'  file.txt
   ```

Example Use Case:

Let’s say you have a file file.txt with the following content:

Copy code
```
apple
banana
apple
orange
```

1. To replace "apple" with "grape" in the file:

   bash
   Copy code
   ```
   sed 's/apple/grape/g' file.txt
   ```

   Result:

   Copy code
   ```
   grape
   banana
   grape
   orange
   ```

2. To delete the second line:

   bash
   Copy code
   ```
   sed '2d' file.txt
   ```

   Result:

   Copy code
   ```
   apple
   apple
   orange
   ```

Options in sed:

- -i: Edit files in place.
- -n: Suppress automatic printing; use with p to print specific lines.
- -e: Add multiple editing commands.
- -f: Execute commands from a file instead of the command line.

Would you like more advanced examples or use cases for sed?

4o

ChatGPT can make mistakes. Check important info.
?

Grep(command)

grep is a command-line tool in Unix/Linux used to search for text patterns within files. It searches line by line and prints lines that match the given pattern.

Basic Syntax:

bash
Copy code

```
grep [options] 'pattern' file_name
```

Common Options:

- -i: Perform case-insensitive search.
- -v: Invert the match, showing lines that **do not** match the pattern.
- -r: Recursively search in a directory and all its subdirectories.
- -n: Show line numbers along with matching lines.
- -c: Count the number of matching lines.
- -l: Display only the names of files containing the pattern.
- -w: Match the whole word only.
- -A [num]: Show num lines **after** the matching line.
- -B [num]: Show num lines **before** the matching line.
- -E: Enable extended regular expressions (like egrep).
- -F: Treat the pattern as a fixed string (like fgrep).

Examples:

1. **Basic Search:** Search for the word "error" in logfile.txt:

   bash
   Copy code

   ```
   grep 'error' logfile.txt
   ```

2. **Case-Insensitive Search:** Search for "error" in any case (e.g., ERROR, Error):

   bash
   Copy code

   ```
   grep -i 'error' logfile.txt
   ```

3. **Recursive Search:** Search for "error" in all files within a directory and subdirectories:

   bash
   Copy code
   ```
   grep -r 'error' /var/log
   ```

4. **Count Occurrences:** Count how many lines contain "error":

   bash
   Copy code
   ```
   grep -c 'error' logfile.txt
   ```

5. **Show Line Numbers:** Display matching lines with their line numbers:

   bash
   Copy code
   ```
   grep -n 'error' logfile.txt
   ```

6. **Invert Match:** Show lines that do **not** contain the word "error":

   bash
   Copy code
   ```
   grep -v 'error' logfile.txt
   ```

7. **Match Whole Word:** Search for the exact word "error" (not as a substring):

   bash
   Copy code
   ```
   grep -w 'error' logfile.txt
   ```

8. **Search and Show Context:** Show 3 lines after the matching line:

   bash
   Copy code
   ```
   grep -A 3 'error' logfile.txt
   ```

   Show 2 lines before the matching line:

   bash
   Copy code
   ```
   grep -B 2 'error' logfile.txt
   ```

9. **Search in Multiple Files:** Search for "error" in multiple files:

   bash
   Copy code

   ```
   grep 'error' logfile1.txt logfile2.txt
   ```

10. **Display Only File Names:** List files that contain the word "error":

    bash
    Copy code

    ```
    grep -l 'error' *.txt
    ```

Summary:

- grep is very versatile for searching text in files using regular expressions.
- It supports options to modify search behavior (case-sensitivity, recursive search, etc.), display context, or count occurrences.

Fgrep

fgrep (Fixed String Global Regular Expression Print) is a variant of grep used for searching **fixed strings** rather than patterns with regular expressions. This means that special characters like ., *, ?, etc., are treated as literal characters, not as regular expression metacharacters.

In many modern Unix/Linux systems, fgrep is often replaced by grep -F, which functions in the same way. Still, both are widely understood and used interchangeably.

Syntax:

bash
Copy code

```
fgrep [options] 'search_string' file_name
```

Key Features:

- **Fixed-String Search**: fgrep does not interpret any regular expressions, so it's faster when searching for simple text or fixed patterns.
- **Speed**: Since it skips regular expression parsing, fgrep is typically faster than grep when searching for fixed strings.

Common Options:

- -i: Case-insensitive search.
- -v: Invert the match; show lines that **do not** match the string.
- -r: Search recursively in all files in a directory and its subdirectories.
- -l: Print only the filenames where the match was found.
- -n: Print line numbers along with matching lines.
- -c: Count the number of matching lines.
- -f file: Read the patterns from a file.
- -o: Print only the matched parts of matching lines.

Example Use Cases:

1. **Basic Search**: Search for the fixed string "apple" in fruits.txt:

   bash
   Copy code

   ```
   fgrep 'apple' fruits.txt
   ```

   Output (if fruits.txt contains the string "apple"):

   bash
   Copy code

   ```
   apple
   ```

2. **Case-Insensitive Search**: Search for "apple" in any case (e.g., Apple, APPLE):

   bash
   Copy code

   ```
   fgrep -i 'apple' fruits.txt
   ```

3. **Search Recursively**: Search for "apple" in all files under a directory:

   bash
   Copy code

   ```
   fgrep -r 'apple' /path/to/directory
   ```

4. **Search Multiple Strings from a File**: If you have a file patterns.txt that contains:

   Copy code

   ```
   apple
   banana
   ```

orange

You can search for any of these strings in fruits.txt using:

bash
Copy code
```
fgrep -f patterns.txt fruits.txt
```

This will find and print lines from fruits.txt that contain any of the words listed in patterns.txt.

5. **Count Matches**: To count how many lines contain "apple":

   bash
   Copy code
   ```
   fgrep -c 'apple' fruits.txt
   ```

6. **Print Filenames with Matches**: List all files in a directory that contain the word "apple":

   bash
   Copy code
   ```
   fgrep -l 'apple' *.txt
   ```

7. **Inverted Match**: Show lines that **do not** contain the word "apple":

   bash
   Copy code
   ```
   fgrep -v 'apple' fruits.txt
   ```

8. **Search for a String with Special Characters**: Unlike grep, fgrep allows you to search for strings with special characters without needing to escape them. For example, if you want to search for the string "a.b*c":

   bash
   Copy code
   ```
   fgrep 'a.b*c' file.txt
   ```

   This will treat the string a.b*c as a literal string, not as a regular expression.

9. **Search Only Matched Parts**: Print only the parts of the line that match the pattern:

bash
Copy code
```
fgrep -o 'apple' fruits.txt
```

This will print apple for every match in the file without showing the entire line.

10. **Print Line Numbers**: Display matching lines along with their line numbers:

    bash
    Copy code
    ```
    fgrep -n 'apple' fruits.txt
    ```

    Output:

    makefile
    Copy code
    ```
    3:apple
    7:apple
    ```

Example Output:

Given a file fruits.txt with the following content:

Copy code
```
apple
banana
orange
grapefruit
apple
```

- Basic Search:

  bash
  Copy code
  ```
  fgrep 'apple' fruits.txt
  ```

  Output:

  Copy code
  ```
  apple
  apple
  ```

- Case-Insensitive Search:

  bash
  Copy code
  ```
  fgrep -i 'APPLE' fruits.txt
  ```

  Output:

  Copy code
  ```
  apple
  apple
  ```

- Count Matches:

  bash
  Copy code
  ```
  fgrep -c 'apple' fruits.txt
  ```

  Output:

  Copy code
  ```
  2
  ```

- List Files Containing Matches:

  bash
  Copy code
  ```
  fgrep -l 'apple' *.txt
  ```

When to Use fgrep:

- When you need to search for **fixed strings** without the overhead of regular expressions.
- When searching for special characters that are usually treated as regex operators in grep.
- When you want to perform fast searches over large files with plain strings.

Deprecation Note:

- fgrep has been deprecated in many modern Unix/Linux systems in favor of grep -F, which has the same functionality. You can replace fgrep with grep -F like this:

bash
Copy code

```
grep -F 'apple' fruits.txt
```

Conclusion:

fgrep is a handy tool for searching fixed strings efficiently, without worrying about regular expression syntax. Although it’s deprecated in favor of grep -F, its behavior remains widely used.