File Management in Linux

Unit-III

SCS281: Linux and Shell Programming

Mapped Course Outcomes (CO): CO4

Er. Anu Arora, Assistant Professor, GNA University

File Management in Linux

In Linux, most of the operations are performed on files. And to handle these files Linux has directories also known as folders which are maintained in a tree-like structure.

Though, these directories are also a type of file themselves. Linux has 3 types of files:

1. Regular Files
2. Directories
3. Special Files

Types of files

Regular Files:

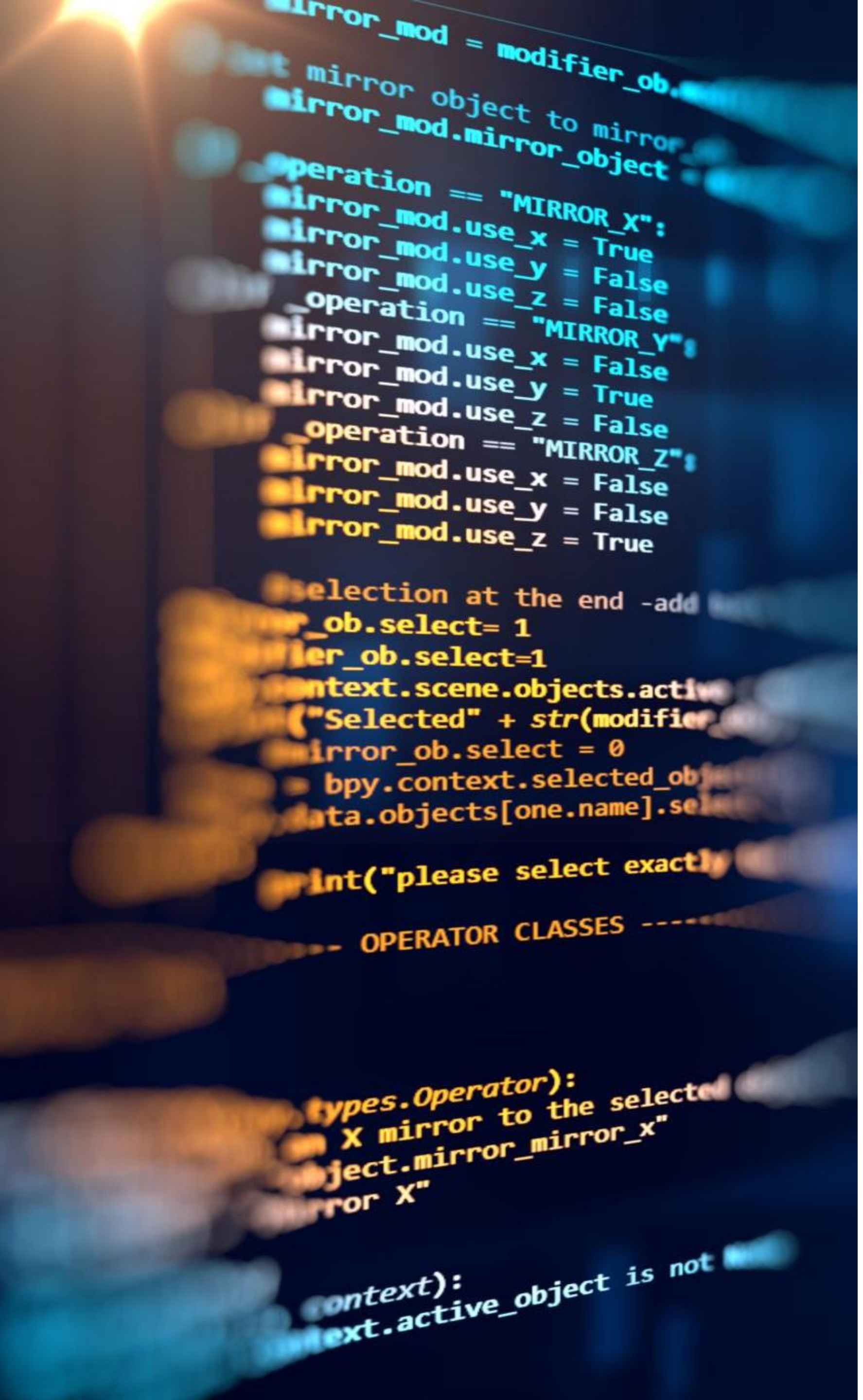
It is the common file type in Linux. it includes files like – text files, images, binary files, etc. Such files can be created using the touch command. They consist of the majority of files in the Linux/UNIX system. The regular file contains ASCII or Human Readable text, executable program binaries, program data and much more.

Directories:

Windows call these directories as folders. These are the files that store the list of file names and the related information. The root directory(/) is the base of the system, /home/ is the default location for user's home directories, /bin for Essential User Binaries, /boot – Static Boot Files, etc. We could create new directories with mkdir command.

Special Files:

Represents a real physical device such as a printer which is used for IO operations. Device or special files are used for device Input/Output(I/O) on UNIX and Linux systems. You can see them in a file system like an ordinary directory or file.



ls Command in Linux

ls is a Linux shell command that lists directory contents of files and directories. It provides valuable information about files, directories, and their attributes.

Syntax of `ls` command in Linux

```
ls [option] [file/directory]
```

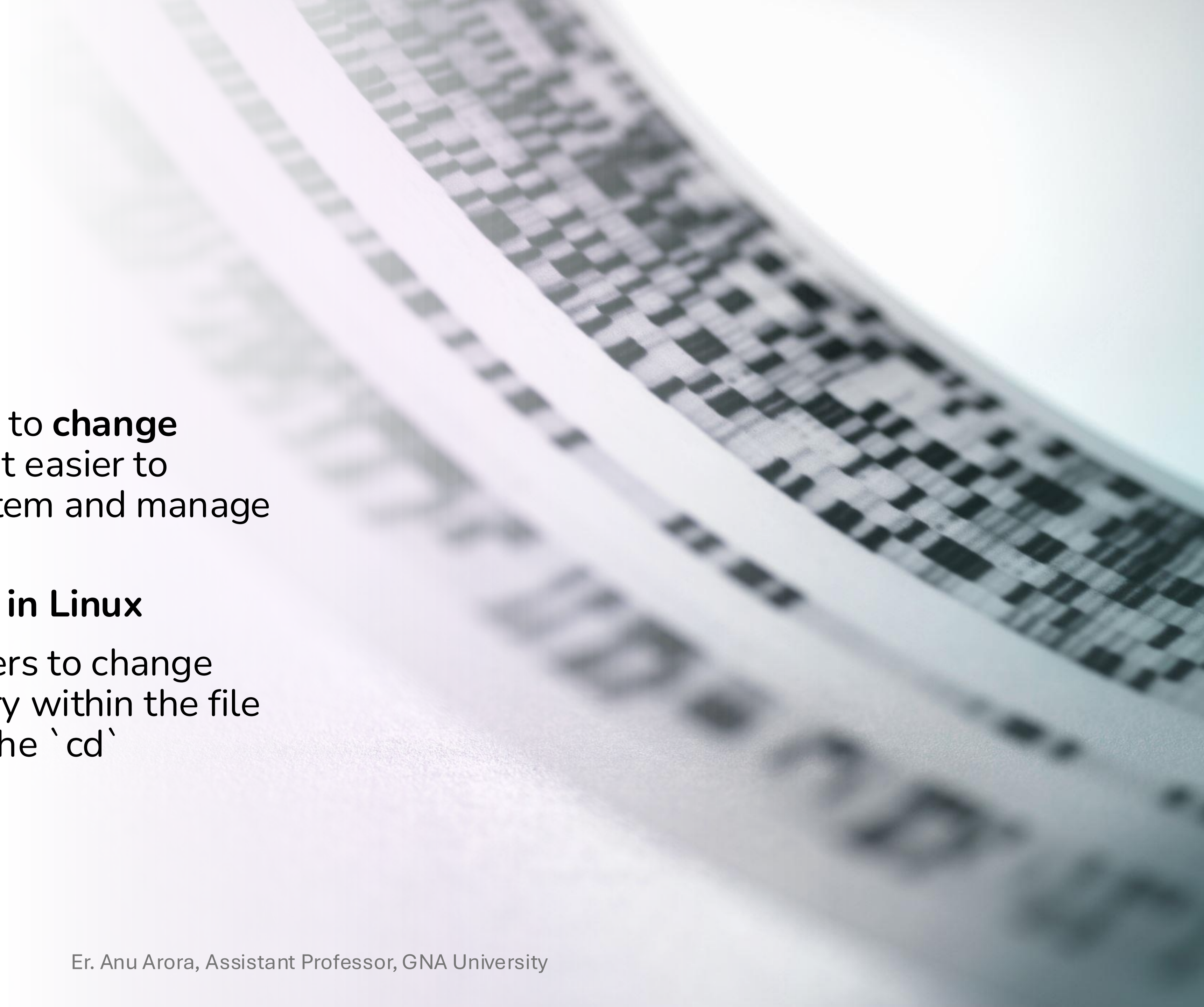
'ls' will display the contents of the current directory. By default, 'ls' lists files and directories in alphabetical order.

Commonly Used Options in `ls` command in Linux

Options	Description
-l	known as a long format that displays detailed information about files and directories.
-a	Represent all files Include hidden files and directories in the listing.
-t	Sort files and directories by their last modification time, displaying the most recently modified ones first.
-r	known as reverse order which is used to reverse the default order of listing.
-S	Sort files and directories by their sizes, listing the largest ones first.
-R	List files and directories recursively, including subdirectories.
-i	known as inode which displays the index number (inode) of each file and directory.
-g	known as group which displays the group ownership of files and directories instead of the owner.
-h	Print file sizes in human-readable format (e.g., 1K, 234M, 2G).
-d	List directories themselves, rather than their contents.

cd command in Linux

- The **cd command** allows you to **change directories in Linux**, making it easier to navigate through the file system and manage your files efficiently.
- **Syntax of the CD Command in Linux**
- The ‘cd’ command allows users to change their current working directory within the file system. The basic syntax of the `cd` command is as follows:
- cd [directory]



1. Move Inside a Subdirectory in Linux Using CD Command

To move inside a subdirectory in Linux we use the CD Linux Command. Here, replace [directory_name] with the desired directory you want to move in.

```
cd [directory_name]
```

For Example: If we want to move to a subdirectory named “Documents”

```
cd Documents
```

2. Using `/` As an Argument in the CD Command in Linux

By using `/` as an argument in `cd` we can change the directory to the root directory.

The root directory is the first directory in your filesystem hierarchy.

```
cd /
```

3. Move Inside a Directory From a Directory

This command is used to move inside a directory from a directory. Here, replace "dir_1/dir_2/dir_3" with the subdirectory name or location you want to move in.

```
cd dir_1/dir_2/dir_3
```

For Example: We are in the "/home/anuarora" directory and we want to move to its sub-directory location (path) "Documents/linuxworks/example"

```
cd Documents/linuxworks/example
```

4. Change Directory to Home Directory From Any Location in Linux

`~` This argument is used in the `cd` command to change the directory to the home directory from any location in the Linux System.

```
cd ~
```

For Example: We are in location “/home/anuarora/Documents/linuxworks/example” and want to move to the home directory. We can use the following command.

```
cd ~
```

5. Move to Parent or One Level Up from the Current Directory in Linux

`We use `..` this as an argument in the `cd` command which is used to move to the parent directory of the current directory, or the directory one level up from the current directory. “..” represents the parent directory.

```
cd ..
```

For Example: We are in location “/home/anuarora/Documents/linuxworks/example” and want to move to the parent or one level up in the directory. We can use the following command.

```
cd ..
```

6. Change Directory by Using DIR NAME Command in Linux

This command is used to navigate to a directory with white spaces. Instead of using double quotes, we can use single quotes then also this command will also work. Here, replace “dir name” with the directory name you want.

```
cd "dir name”
```

For Example: If we want to move to “dir name” = “My songs”. We use the following command.

```
cd "My songs”
```

We can also use `\` in between if we don’t want to use double or single quotes.

```
cd My\ songs
```

mkdir command in Linux

The mkdir command in Linux is used to create directories (or folders). It's one of the most commonly used commands, as directories are essential for organizing files on a system. Let's break down its usage in more detail:

Basic Syntax:

mkdir [OPTION] DIRECTORY...

- mkdir: Command to create a directory.
- [OPTION]: Optional flags or switches that modify the behavior of the command.
- DIRECTORY: The name or path of the directory to be created. You can also specify multiple directories to be created at once.

Examples of Usage:

1. Creating a Single Directory:

```
mkdir my_folder
```

This will create a directory called my_folder in the current working directory (where you are currently in the file system). You can confirm its creation using the ls command.

2. Creating Multiple Directories:

```
mkdir dir1 dir2 dir3
```

You can create multiple directories at once by specifying their names in a space-separated list. In this example, directories dir1, dir2, and dir3 will be created in the current directory.

3. Creating Nested Directories:

```
mkdir -p parent_dir/child_dir/grandchild_dir
```

The -p option allows you to create parent directories if they do not already exist. Without this option, if you try to create a nested directory structure and the parent directories don't exist, mkdir will return an error.

Common Options for mkdir:

-p, --parents	-m, --mode	-v, --verbose	--help
No error if the directory already exists; make parent directories as needed.	Set the file permissions (mode) of the new directory.	Print a message for each created directory.	Display help information about the mkdir command.

pwd command in Linux

The pwd command in Linux stands for **Print Working Directory**. It is used to display the current directory you are in. This command is crucial for navigation in the Linux file system, as it shows the full, absolute path of the current working directory.

Basic Syntax:

pwd [OPTION]

How It Works:

When you open a terminal, you are placed in a specific directory (usually your home directory). As you move between directories using the cd (change directory) command, it is essential to know where you are in the directory structure. This is where pwd comes in—it displays the full path of your current location.

Absolute vs. Relative Paths

Absolute path: This is the full path from the root directory (/). For example, /home/user/documents is an absolute path.

Relative path: This refers to the path relative to the current directory. For example, if you're in /home/user and you cd documents, that is a relative path. But when you use pwd, it will always show the absolute path (/home/user/documents).

Key Options in pwd:

There are a few options you can use with pwd to modify its behavior:

1. **-L (Logical Path)**: This is the default option, which prints the **logical** current working directory, accounting for symbolic links (symlinks). A symlink is a reference to another directory or file, which can obscure the actual physical path.

Example: If you have a symlink /mydir pointing to /home/user/myfolder, and you cd /mydir, using pwd with the -L option will print:

/mydir

It shows the symlinked path, not the real location.

Key Options in pwd:

2. -P (Physical Path): The -P option prints the **physical** path by resolving all symbolic links, showing the actual location on disk.

Example: If /mydir is a symlink to /home/user/myfolder, and you cd /mydir, using pwd with the -P option will display:

```
/home/user/myfolder
```

Command:

```
pwd -P
```

This distinction is useful when working with symlinks and when you need to understand the real location of files and directories.

touch command in Linux

The touch command in Linux is primarily used to create empty files or update the timestamps (such as access and modification times) of existing files without modifying their contents. It’s one of the simplest and most useful commands for quickly creating files in a Linux environment.

Basic Syntax:

```
touch [OPTION]... FILE...
```

- touch: The command itself.
- [OPTION]: Optional flags that modify the behavior of the command.
- FILE: The name of the file(s) to be created or whose timestamp you want to update. You can specify multiple files separated by spaces.

Common Uses of the touch Command:

1. Creating an Empty File:

The most common use of touch is to create an empty file. If the file does not exist, touch creates it.

```
touch newfile.txt
```

This creates a new, empty file named newfile.txt in the current directory.

2. Creating Multiple Files:

You can also create multiple empty files at once by specifying their names separated by spaces.

```
touch file1.txt file2.txt file3.txt
```

This will create three empty files named file1.txt, file2.txt, and file3.txt.

Common Uses of the touch Command:

3. Updating Timestamps of Existing Files:

If a file already exists, touch updates its **access** and **modification** timestamps to the current time without altering the content of the file.

```
touch existingfile.txt
```

If existingfile.txt exists, its last modified and last accessed times will be updated to the current time.

4. Prevent Creating a New File (-c or --no-create):

If you don’t want to create a file if it doesn’t exist but still want to update the timestamps of existing files, you can use the -c option.

```
touch -c existingfile.txt
```

If existingfile.txt exists, the timestamps will be updated. If it does not exist, touch does nothing.

Common Uses of the touch Command:

5. Set Specific Timestamp (-t or --time):

You can manually set a specific timestamp for a file using the -t option. The format is [[[CC]YY]MMDD]hhmm[.ss], where:

CC: Century (optional)

YY: Year

MM: Month

DD: Day

hh: Hour

mm: Minutes

ss: Seconds (optional)

Example:

```
touch -t 202310150830.45 myfile.txt
```

This sets the timestamp of myfile.txt to **October 15, 2023**, at **08:30:45 AM**.

Options for touch:

- **-a**: Change only the access time.
- **-m**: Change only the modification time.
- **-c or --no-create**: Do not create the file if it does not exist.
- **-r or --reference=FILE**: Use the timestamp from another file.
- **-t [[CC]YY]MMDDhhmm[.ss]**: Set a custom time.
- **-d or --date=STRING**: Parse a date string to set a specific timestamp (e.g., "tomorrow," "2 hours ago").

cat command in Linux

The cat command in Linux is short for "concatenate" and is one of the most frequently used commands. Its primary purpose is to **display the contents of files** to the terminal, but it can also be used for combining multiple files, creating new files, and appending data to files.

Basic Syntax:

```
cat [OPTION] [FILE]...
```

- cat: The command itself.
- [OPTION]: Various options to modify the behavior of the command.
- [FILE]...: The name(s) of the file(s) to be displayed or manipulated. You can specify multiple files, separated by spaces.

Common Uses of cat

1. Display the Content of a File:

The most basic use of cat is to display the contents of a file. This outputs the file's content directly to the terminal.

```
cat file.txt
```

If file.txt contains:

```
Hello, World! Welcome to Linux.
```

Common Uses of cat

2. Display Multiple Files:

You can concatenate and display multiple files one after another by listing them:

```
cat file1.txt file2.txt
```

3. Create a New File:

You can use cat to create a new file by redirecting the output into a file.

```
cat > newfile.txt
```

This opens a prompt where you can type the contents of the new file. Once you're done typing, press Ctrl+D to save and close the file.

Common Uses of cat

4. Append Content to an Existing File:

You can append new content to an existing file using >> (append operator):

```
cat >> existingfile.txt
```

This will open a prompt where you can add new content to existingfile.txt. After typing, press Ctrl+D to close.

5. Redirect Output of Multiple Files to Another File:

To combine multiple files and save the result into a new file, use the > operator:

```
cat file1.txt file2.txt > combined.txt
```

This creates a new file combined.txt with the contents of file1.txt and file2.txt combined.

Common Uses of cat

6. View Contents with Line Numbers:

If you want to display the contents of a file along with line numbers, you can use the -n option:

```
cat -n file.txt
```

7. Display Non-printable Characters:

The -v option can be used to show non-printing characters, such as tabs or line feeds, represented by visible symbols.

```
cat -v file.txt
```

Important cat Options:

-n: Number all output lines.

-b: Number only non-blank lines.

-s: Squeeze multiple adjacent blank lines into one.

-E: Display a $ at the end of each line.

-T: Display tabs as ^I characters.

-v: Show non-printing characters.

more Command in Linux

The more command is a simple pager utility that allows you to view the contents of a file one screenful at a time. It’s useful when dealing with large files because it prevents the content from scrolling off the screen.

Basic Syntax:

```
more [options] [file]
```

Example:

```
more largefile.txt
```

Navigation in more:

- **Spacebar**: Move forward by one page.
- **Enter**: Move forward by one line.
- **b**: Move backward by one page.
- **q**: Quit and exit more.
- **/pattern**: Search for a pattern (e.g., /word will search for "word").
- **n**: Repeat the search in the same direction (forward).
- **h**: Display help information.

Common Options:

-d: Display an error message for unrecognized commands instead of beeping.

-c: Clears the screen before displaying the next page of text.

+n: Start at the nth line of the file.

+/pattern: Start from the first occurrence of the search pattern.

Limitations of more:

- You cannot scroll backward through the file unless you've explicitly enabled backward movement.
- Limited functionality compared to less (e.g., no advanced searching, no direct line navigation).

less Command in Linux

The less command is an improved version of more and provides more powerful features for viewing files. It allows both forward and backward navigation through files, and you can perform searches, navigate directly to specific lines, and even load large files without having to wait for the entire file to load.

Basic Syntax:

```
less [options] [file]
```

Example:

```
less largefile.txt
```

Navigation in less:

- **Spacebar**: Move forward by one page.
- **Enter**: Move forward by one line.
- **b**: Move backward by one page.
- **y**: Move backward by one line.
- **q**: Quit and exit less.
- **/pattern**: Search forward for a pattern (e.g., /word to search for "word").
- **?pattern**: Search backward for a pattern.
- **n**: Repeat the search in the same direction (forward if / was used, backward if ? was used).
- **N**: Repeat the search in the opposite direction.
- **g**: Go to the beginning of the file.
- **G**: Go to the end of the file.
- **CTRL+g**: Display file information (file name, current line number, total number of lines, etc.).

Common Options:

-N: Display line numbers.

-S: Truncate long lines instead of wrapping them.

+n: Start viewing the file from the nth line.

-X: Prevent the terminal from clearing the screen after quitting less.

/pattern: Start searching for the pattern from the beginning.

Advanced Features of less

Scrolling Backward: Unlike more, you can scroll backward at any time using b or y.

Multiple Files: You can view multiple files with less and navigate between them using the :n (next file) and :p (previous file) commands.

Buffers Large Files: less doesn’t load the whole file into memory at once, making it efficient for very large files.

Viewing Command Output: You can pipe the output of a command directly to less using |

Example:

```
ls -l | less
```

Key Differences Between more & less

Feature	more	less
Backward Scrolling	Limited	Full support
Search Functionality	Basic (/pattern)	Advanced (/ and ?)
Efficient with Large Files	Loads file sequentially	Loads in chunks, better for large files
Memory Usage	Higher for large files	Lower, as it loads files in chunks
Navigation	Basic	Advanced (jump to line, buffer multiple files)
Wrapping Long Lines	Wraps lines by default	Can truncate lines with -S

Linux Operations

Overview of Linux Operations

1. **find**: Used to search for files based on various attributes such as name, size, and modification date.
2. **cp**: Copies files and directories.
3. **mv**: Moves or renames files and directories.
4. **rm**: Deletes files and directories.
5. **ln**: Creates links between files, either as hard links or symbolic links.

find Command

The find command is used to search for files and directories in a directory hierarchy based on various criteria like name, size, modification time, etc.

Syntax:

```
find [path] [expression]
```

Examples:

Find all .txt files in the current directory and subdirectories:

```
find . -name "*.txt"
```

- .: Current directory
- -name "*.txt": Finds files with the .txt extension

find Command

- **Find files modified in the last 7 days:**
- find /home/user -mtime -7
  - /home/user: Directory to search
  - -mtime -7: Files modified in the last 7 days

Common Options:

- -name "pattern": Finds files matching the pattern.
- -type d: Finds directories.
- -type f: Finds regular files.
- -size +1M: Finds files larger than 1MB.
- -exec command {} \;: Executes a command on each file found, where {} is the placeholder for the file.

cp Command

The cp command is used to copy files and directories from one location to another.

Syntax:

```
cp [options] source destination
```

Examples:

Copy a file to a new location:

```
cp file1.txt /home/user/documents/
```

- file1.txt: Source file
- /home/user/documents/: Destination directory

Copy a directory and its contents recursively:

- `cp -r /home/user/docs /home/user/backup/`
  - -r: Copies the directory and its contents recursively

Common Options:

- -r: Copy directories recursively.
- -i: Prompts before overwriting an existing file.
- -v: Displays verbose output (shows files being copied).
- -u: Copies only when the source file is newer than the destination file or when the destination file does not exist.

mv Command

The mv command is used to move files and directories. It can also be used to rename files.

Syntax:

```
mv [options] source destination
```

Examples:

Move a file to a different directory:

```
mv file1.txt /home/user/documents/
```

- Moves file1.txt to /home/user/documents/

Rename a file:

```
mv oldname.txt newname.txt
```

Renames oldname.txt to newname.txt

Move multiple files to a directory:

```
mv file1.txt file2.txt /home/user/backup/
```

Common Options

- -i: Prompts before overwriting an existing file.
- -v: Displays verbose output showing the files being moved.
- -n: Prevents overwriting files that already exist.

rm Command

The rm command is used to remove files and directories. It can delete single files, multiple files, or entire directories.

Syntax:

```
rm [options] file
```

Examples:

Remove a file:

```
rm file1.txt
```

- Deletes file1.txt

Remove multiple files:

```
rm file1.txt file2.txt file3.txt
```

Remove a directory and its contents recursively:

```
rm -r /home/user/docs
```

- -r: Deletes the directory and all its contents

Common Options

- -r: Recursively deletes directories and their contents.
- -f: Forces deletion without asking for confirmation.
- -i: Prompts for confirmation before each deletion.

Warning: The rm command is destructive and permanent; be cautious while using rm -r or rm -f.

ln Command

The ln command is used to create links between files. There are two types of links:

Hard Link: Directly points to the physical data on disk, sharing the same inode.

Symbolic Link (Soft Link): A pointer to another file or directory, similar to a shortcut.

Syntax:

```
ln [options] target link_name
```

Examples:

Create a hard link:

```
ln file1.txt link_to_file1.txt
```

- Creates a hard link named link_to_file1.txt pointing to file1.txt

Create a symbolic (soft) link:

```
ln -s /home/user/file1.txt /home/user/link_to_file1.txt
```

- -s: Creates a symbolic link

Common Options:

-s: Creates a symbolic (soft) link.

-f: Forces the creation of the link, overwriting existing files.

Filters

- In Linux, **filters** are commands or programs that receive data from standard input (like a file or another command), process that data in a specific way, and then send the output to standard output.
- Filters are incredibly useful for text processing, allowing users to manipulate and transform data from one form to another or extract meaningful insights from large text files.

Common Uses of Filters in Linux

1. **Viewing Specific Parts of Files**:
    1. **head**: Displays the beginning lines of a file.
    2. **tail**: Shows the last lines of a file, commonly used with -f to follow log files in real-time.
2. **Selecting or Extracting Data**:
    1. **cut**: Extracts specific columns or fields based on delimiters. Useful for data processing in structured files (like CSV files).
    2. **grep**: Searches for lines that match a specified pattern, ideal for finding specific information within files.
    3. **egrep** and **fgrep**: Extended and fixed-string versions of grep for advanced pattern matching.

Common Uses of Filters in Linux

1. **Sorting and Arranging Data**:
    1. **sort**: Sorts lines alphabetically, numerically, or in reverse, allowing ordered data representation.
    2. **uniq**: Filters out adjacent duplicate lines in sorted data, useful for identifying unique entries.
2. **Combining Data**:
    1. **paste**: Merges lines of multiple files side by side, helpful for combining related data into a single view.
3. **Formatting Output**:
    1. **pr**: Formats text into pages, adding headers and footers for better readability in printouts.
4. **Performing Calculations or Counting**:
    1. **wc** (word count): Counts lines, words, and characters in files, providing basic statistics about text data.

head Command

Displays the first few lines of a file. By default, it shows the first 10 lines.

Syntax:

```
head [options] [file]
```

Example:

```
head -n 5 file.txt
```

- Shows the first 5 lines of file.txt.

Common Options:

- -n [number]: Specifies the number of lines to display.

tail Command

Displays the last few lines of a file. Like head, it defaults to 10 lines.

Syntax:

tail [options] [file]

Example:

tail -n 3 file.txt

- Shows the last 3 lines of file.txt.

Common Options:

- -n [number]: Specifies the number of lines to display.
- -f: Follows the file, useful for watching log files in real-time.

pr Command

Formats text files for printing. It can add headers, footers, and paginate content.

Syntax:

```
pr [options] [file]
```

Example:

```
pr -h "Report" file.txt
```

- Adds the header "Report" to each page of file.txt.

Common Options:

- -h [text]: Adds a custom header.
- -l [length]: Specifies the page length.
- -d: Double-spaces the output.

cut Command

Extracts specific columns from a file or text.

Syntax:

```
cut [options] [file]
```

Example:

```
cut -d ',' -f 2 file.csv
```

- Extracts the second field from file.csv, using a comma as the delimiter.

Common Options:

- -d [delimiter]: Sets the delimiter character.
- -f [fields]: Specifies which fields to cut.

paste Command

Merges lines from multiple files horizontally.

Syntax:

```
paste [options] file1 file2
```

Example:

```
paste file1.txt file2.txt
```

- Merges each line of file1.txt with the corresponding line of file2.txt.

Common Options:

- -d [delimiter]: Sets a custom delimiter.

sort Command

Sorts lines in a text file.

Syntax:

```
sort [options] [file]
```

Example:

```
sort -r file.txt
```

- Sorts file.txt in reverse order.

Common Options:

- -r: Sorts in reverse.
- -n: Sorts numerically.
- -u: Removes duplicates in sorted output.

uniq Command

Filters out adjacent duplicate lines from a sorted file.

Syntax:

```
uniq [options] [file]
```

Example:

```
sort file.txt | uniq
```

- Sorts and removes duplicates from file.txt.

Common Options:

- -c: Counts the number of occurrences.
- -d: Displays only duplicate lines.

grep Command

- Searches for patterns in files. It supports regular expressions.
- **Syntax**:
- grep [options] pattern [file]
- **Example**:
- grep "error" file.txt
  - Searches for "error" in file.txt.
- **Common Options**:
  - -i: Ignores case.
  - -v: Inverts the match, showing lines that don’t match.
  - -r: Searches recursively in directories.

egrep Command

Extended grep that supports additional regular expressions like | (OR), +, and ?.

Syntax:

```
egrep [options] pattern [file]
```

Example:

```
egrep "error|fail" file.txt
```

- Searches for "error" or "fail" in file.txt.

Common Options:

- Same as grep.

fgrep Command

Fixed grep that treats patterns as fixed strings rather than regular expressions. It’s faster for simple text matches.

Syntax:

```
fgrep [options] pattern [file]
```

Example:

```
fgrep "error" file.txt
```

- Searches for the literal word "error" without interpreting any special characters.

Common Options:

- Same as grep.

Text Editors-vi, vim in linux

vi and vim are powerful, widely used text editors in Linux, essential for file editing, coding, and system administration tasks. While both editors operate in a command-line interface, vim (short for "Vi IMproved") offers more advanced features, building on the functionality of vi.

vi Editor

vi (Visual Interface) is the traditional text editor for Unix and Linux systems. It’s known for its minimalistic interface, keyboard-driven commands, and modal editing. vi has two main modes:

Command Mode: This is the default mode when you open vi. You can navigate, delete, and manipulate text using specific commands in this mode.

Insert Mode: This is the mode where you can enter and edit text. You enter this mode by pressing i or a from command mode.

Basic Commands in vi

Starting vi: Open a file with:

```
vi filename
```

Switch to Insert Mode: Press i to start inserting text.

vi Editor

Saving and Exiting:

- :w: Save the file.
- :q: Quit the editor.
- :wq: Save and quit.
- :q!: Quit without saving.

Navigation Commands

- h, j, k, l: Move the cursor left, down, up, and right, respectively.
- 0 (zero): Move to the beginning of the line.
- $: Move to the end of the line.
- G: Move to the last line of the file.
- gg: Move to the first line of the file.

vim Editor

- vim is an enhanced version of vi with more powerful editing features. It retains compatibility with vi but includes improvements like syntax highlighting, multi-level undo, better navigation, and scripting capabilities.

- **Installing vim** (if it's not installed by default):

```
sudo apt-get install vim
```

Additional Features in vim

Syntax Highlighting: Colors code or text based on syntax, which helps in reading and editing code.

Multi-level Undo: Allows multiple undos, which vi does not support.

Extended Commands and Plugins: vim has more commands and supports plugins for customized workflows.

Auto-indentation: Supports smart indentation for code.

Basic vim Commands

Most vi commands work in vim, but vim also offers some additional commands:

Entering Insert Mode:

- i: Enter insert mode at the cursor.
- I: Enter insert mode at the beginning of the line.
- A: Enter insert mode at the end of the line.

Exiting vim:

- :x or :wq: Save and quit.
- :q!: Quit without saving.

Undo/Redo:

- u: Undo the last change.
- Ctrl + r: Redo the last undone change.

Basic vim Commands

Copy and Paste (Yank and Put):

- yy: Copy (yank) the current line.
- p: Paste (put) the copied text after the cursor.
- dd: Delete (cut) the current line.

Search and Replace:

- /pattern: Search for a pattern.
- :s/old/new/: Replace the first instance of old with new on the current line.
- :%s/old/new/g: Replace all instances of old with new in the entire file.

Why Use vi or vim?

Both editors are highly efficient for text and code editing, especially for experienced users. Here are some advantages:

- **Keyboard Navigation**: Allows hands to remain on the keyboard, avoiding the need for a mouse.
- **Customization**: vim can be customized with .vimrc settings and plugins.
- **Low Resource Use**: Both editors use minimal system resources, making them fast and efficient, even for large files.

Basic Workflow Example in vim

To illustrate a typical workflow in vim:

1. Open the file: vim file.txt
2. Enter Insert Mode by pressing i and type your text.
3. Press Esc to return to Command Mode.
4. Save changes with :w and exit with :q or combine them as :wq.

- vi and vim are core tools in the Linux environment, offering unmatched control and efficiency for text editing, making them essential for developers and administrators alike.