CHAPTER 2 : COMMAND LINE EDITING

By Hansika Bhambhaney

Objectives:

1. Enable command-line editing in bash
2. Use and navigate the command history list
3. Operate in **emacs editing mode**
4. Operate in **vi editing mode**
5. Use the **`fc` command** to edit and re-execute history commands
6. Apply **history expansion** (e.g., `!!`, `!n`)
7. Customize editing keys using **`readline`** and **`bind`**
8. Build efficient **keyboard habits** for command-line work

2.1🔧 What is Command-Line Editing?

Command-line editing allows users to interactively edit commands as they type them into the terminal. This means instead of typing a long command from scratch again, we can move around within it, change parts of it, and even reuse previous commands — saving time and effort.

- **Two Editing Modes in bash**

1.**emacs mode** – This is the default mode in bash.

– Uses common keyboard shortcuts.

– Preferred by users familiar with Emacs-style controls.

2.**vi mode** – Must be manually enabled.

- Mimics behavior of the `vi` editor.
- Ideal for users comfortable with `vi`.

⚙️ How to Enable an Editing Mode?

To use emacs mode, we must ensure it is turned on. If our shell is set to something else (like vi mode), we can switch back. Similarly, to switch to vi mode, we must manually activate it.

Example 1: Using Emacs Mode (Default)

```bash
ls -l /usr/local/bin
```

Here we made a mistake and want to fix `/usr/local/bin` to `/usr/bin`.

Instead of deleting the entire line, in emacs mode, we can:

- **Press Ctrl + A to jump to the beginning of the line.**
- **Use the arrow keys or Ctrl + F to move forward through the line.**
- **Use Ctrl + D to delete one character.**
- **Use Ctrl + K to delete everything from the cursor to the end of the line.**
- **Press Enter to run the corrected command.**

Example 2: Using vi Mode

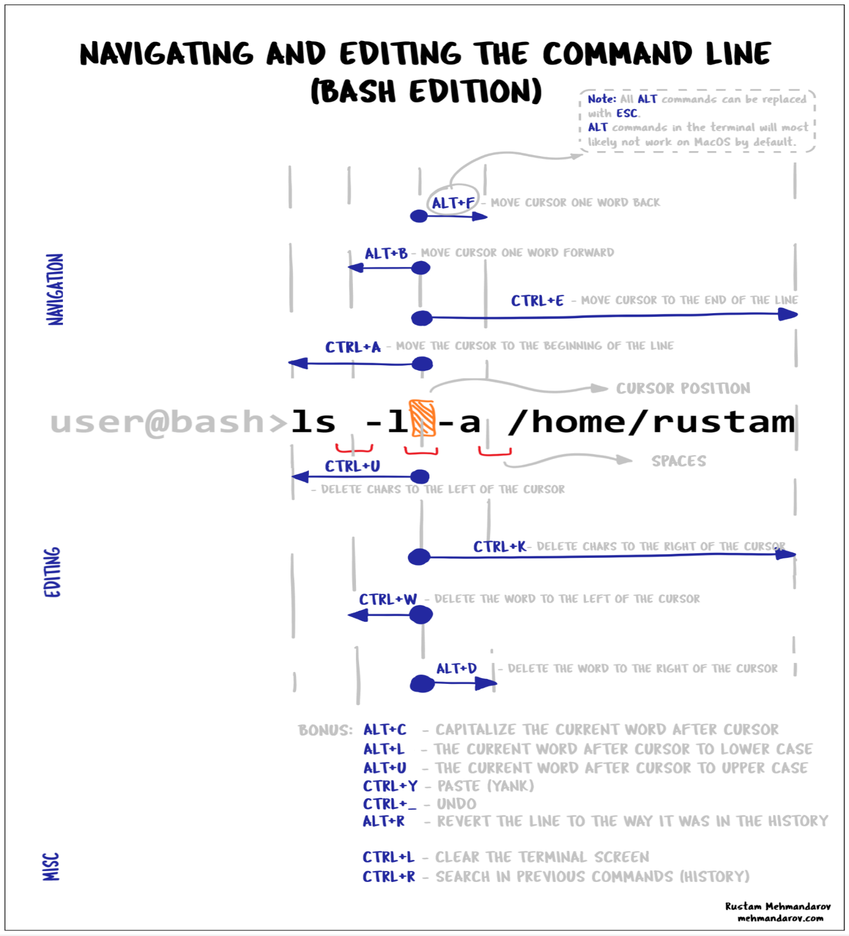
After switching to vi mode:

- When we type a command, we are in insert mode by default.
- To enter command mode, press the Escape key.
- Then, you can:
    - Press k to scroll to the previous command in history.
    - Press l and h to move right or left
    - Press x to delete a character.

```bash
cat document.tx
```

Here we missed the “t” in `.txt`, you can:

- Press Escape to enter command mode.
- Use the cursor to go to `.tx`
- Press **l** to move over the "x".
- Press **i** and type “t”.
- Press **Enter** to execute.


NAVIGATING AND EDITING THE COMMAND LINE (BASH EDITION)
Note: All ALT commands can be replaced with ESC. ALT commands in the terminal will most likely not work on MacOS by default.
NAVIGATION
ALT+F - MOVE CURSOR ONE WORD BACK
ALT+B - MOVE CURSOR ONE WORD FORWARD
CTRL+E - MOVE CURSOR TO THE END OF THE LINE
CTRL+A - MOVE THE CURSOR TO THE BEGINNING OF THE LINE
CURSOR POSITION
user@bash>ls -l -a /home/rustam
SPACES
EDITING
CTRL+U - DELETE CHARS TO THE LEFT OF THE CURSOR
CTRL+K - DELETE CHARS TO THE RIGHT OF THE CURSOR
CTRL+W - DELETE THE WORD TO THE LEFT OF THE CURSOR
ALT+D - DELETE THE WORD TO THE RIGHT OF THE CURSOR
BONUS: ALT+C - CAPITALIZE THE CURRENT WORD AFTER CURSOR
ALT+L - THE CURRENT WORD AFTER CURSOR TO LOWER CASE
ALT+U - THE CURRENT WORD AFTER CURSOR TO UPPER CASE
CTRL+Y - PASTE (YANK)
CTRL+_ - UNDO
ALT+R - REVERT THE LINE TO THE WAY IT WAS IN THE HISTORY
MISC
CTRL+L - CLEAR THE TERMINAL SCREEN
CTRL+R - SEARCH IN PREVIOUS COMMANDS (HISTORY)
Rustam Mehmandarov
mehmandarov.com

2.2 The History List

The history list in `bash` is a built-in feature that stores the commands we've previously entered during our shell session. This allows us to:

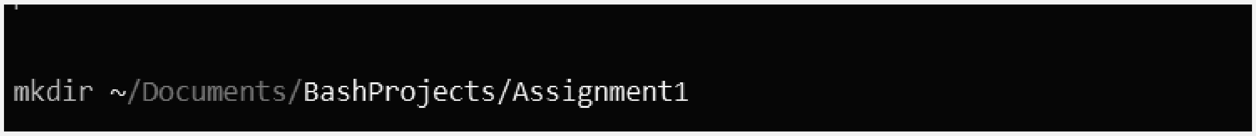
- Recall
- Edit
- Re-execute
  any past command without typing it again.

It's especially helpful when working with long or repetitive commands.

Whenever you type a command and press Enter, bash saves that command in a list called the history list.

- This list persists during your session.
- You can also configure bash to save it to a file, so it’s available the next time you open the terminal.

Example:



```
mkdir ~/Documents/BashProjects/Assignment1
```

Now we want to go back and repeat or modify that command.

- Press the Up Arrow until we see that command.
- Once it appears, we can:
  - Edit the folder name (e.g., change `Assignment1` to `Assignment2`)
  - Press Enter to execute it.

2.3 What Is emacs Editing Mode?

- `emacs` editing mode allows us to edit our command line using familiar `emacs`-style keyboard shortcuts.
- It is the default editing mode in bash.
- In this mode, we can:
  - Move around within the command line
  - Delete, insert, or change text
  - Recall and edit previous commands from history

emacs Mode Uses:

- It increases efficiency: we don’t have to retype long commands.
- We can fix typos, reuse complex commands, or quickly access history.
- Ideal for users comfortable with keyboard shortcuts.

<u>Key Functional Areas in emacs Mode:</u>

The section is divided into subtopics in the book, each covering a different type of command:

🔹 **2.3.1 – Basic Commands**

- **Move to start of the line** → `Ctrl + A`
- **Move to end of the line** → `Ctrl + E`
- **Delete from cursor to end of line** → `Ctrl + K`
- **Delete from cursor to start of line** → `Ctrl + U`
- **Delete the current character** → `Ctrl + D`
- **Clear the screen** → `Ctrl + L`

Example :
We're typing `grep 'error' /var/log/syslog`, but we want to quickly go to the beginning to add `sudo`. Press `Ctrl + A`, type `sudo`, then press space and continue.

2.3.2 – Word Commands

- **Move forward one word** → `Meta + F`
- **Move backward one word** → `Meta + B`
- **Delete next word** → `Meta + D`
- **Delete previous word** → `Ctrl + W` or `Meta + Backspace`

Example :
If typed `ls /home/hansika/Documents` but want to delete "Documents", place the cursor before it and press `Meta + D`.

2.3.3 – Line Commands

These involve killing and yanking text (similar to cutting and pasting):

- Kill (cut) text from cursor to end → `Ctrl + K`
- Kill entire line → `Ctrl + U`
- Yank (paste) killed text → `Ctrl + Y`
- Undo last edit→ `Ctrl + _` (underscore)

2.3.4 – Moving Around in the History List

- **Previous command** → `Ctrl + P` (or Up Arrow)
- **Next command** → `Ctrl + N` (or Down Arrow)

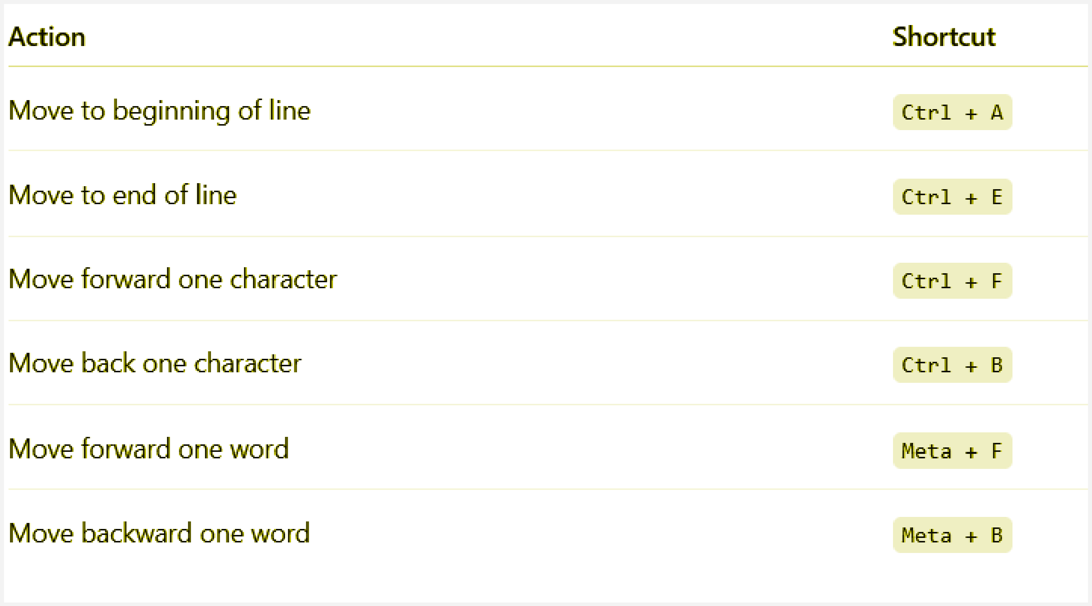
Example:
We're editing a command but want to recall the previous one. Press `Ctrl + P` to see the last executed command.

2.3.5 – Textual Completion

- **Auto-complete a filename or command** → `Tab`
  If multiple matches exist, pressing Tab again shows all possibilities.

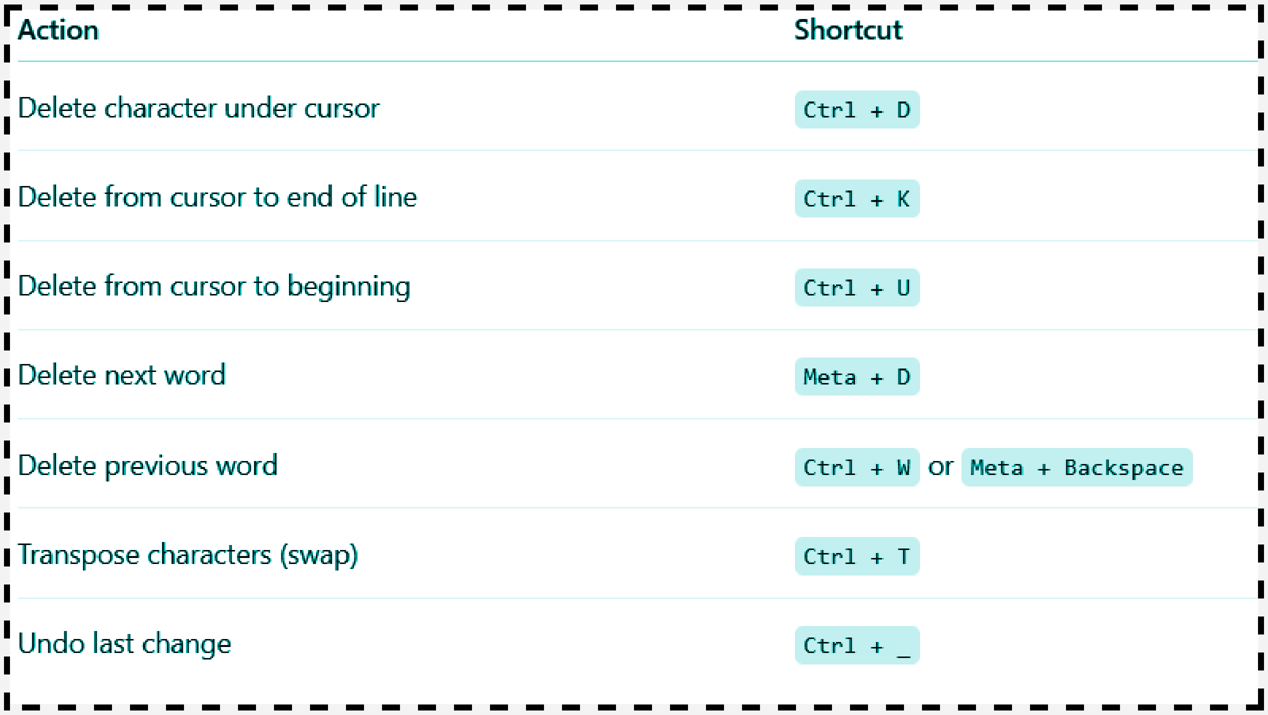
  Example:
  So type `cd Doc` and press `Tab`; bash completes it to `cd Documents/` if that's the only match.

CURSOR MOVEMENT :



Action	Shortcut
Move to beginning of line	`Ctrl + A`
Move to end of line	`Ctrl + E`
Move forward one character	`Ctrl + F`
Move back one character	`Ctrl + B`
Move forward one word	`Meta + F`
Move backward one word	`Meta + B`

EDITING TEXT:



Action	Shortcut
Delete character under cursor	`Ctrl + D`
Delete from cursor to end of line	`Ctrl + K`
Delete from cursor to beginning	`Ctrl + U`
Delete next word	`Meta + D`
Delete previous word	`Ctrl + W` or `Meta + Backspace`
Transpose characters (swap)	`Ctrl + T`
Undo last change	`Ctrl + _`

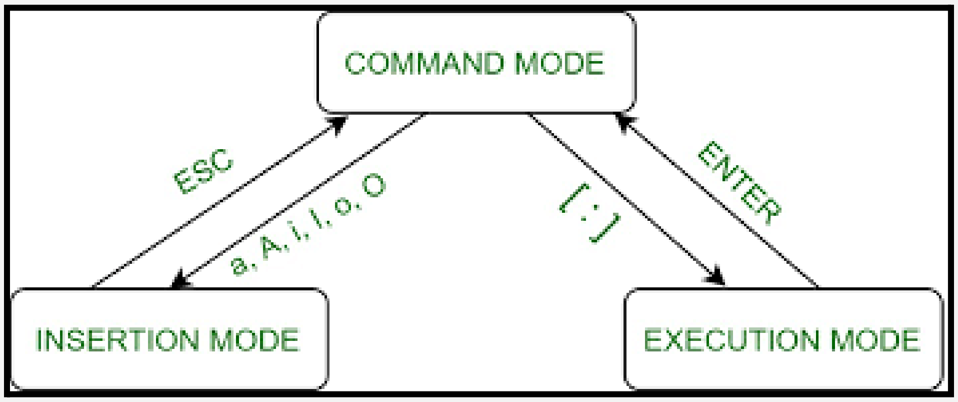
2.4 – vi Editing Mode

- `vi` editing mode lets you edit command lines using the same keystrokes as the `vi` editor.
- This mode is powerful but requires familiarity with `vi`-style navigation.
- Unlike `emacs` mode, vi mode has two modes:

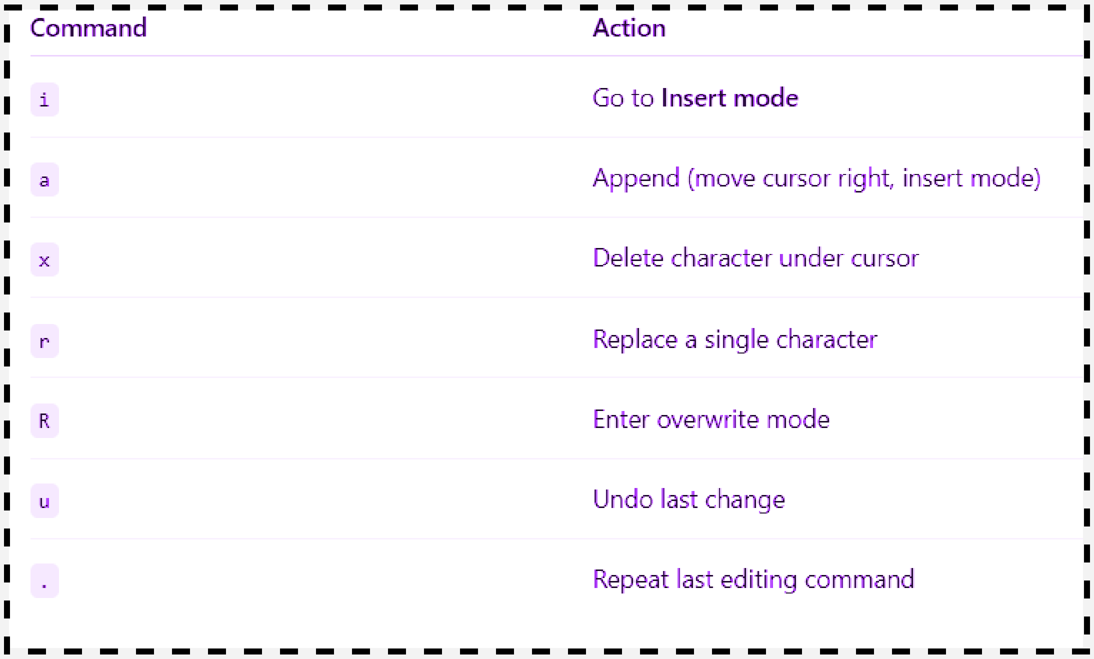
  1. Insert Mode – You type and edit commands here (default when starting a command).
  2. Command Mode – You can move, delete, or modify text using `vi` commands.

Switching Between Modes:

- When you start typing a command, you're in insert mode.
- To switch to command mode, press the Escape (Esc) key.
- From command mode, press i to return to insert mode.



2.4.1 – Simple Control Mode Commands



Command	Action
`i`	Go to **Insert mode**
`a`	Append (move cursor right, insert mode)
`x`	Delete character under cursor
`r`	Replace a single character
`R`	Enter overwrite mode
`u`	Undo last change
`.`	Repeat last editing command

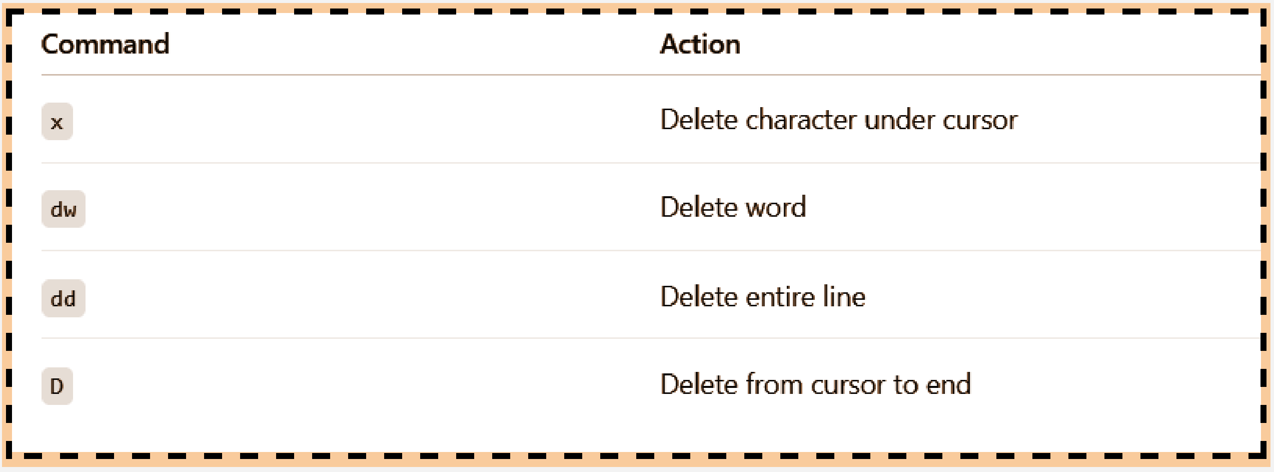
2.4.2 – Entering and Changing Text

We can:

- Use `cw` to change a word
- Use `C` to change to end of line
- Use `cc` to change the entire line

Example:
We typed `ls /usr/bin` but want to change it to `/usr/local/bin`. Press `Esc`, then `0cw` to change the first word (`ls`), and type your new text.

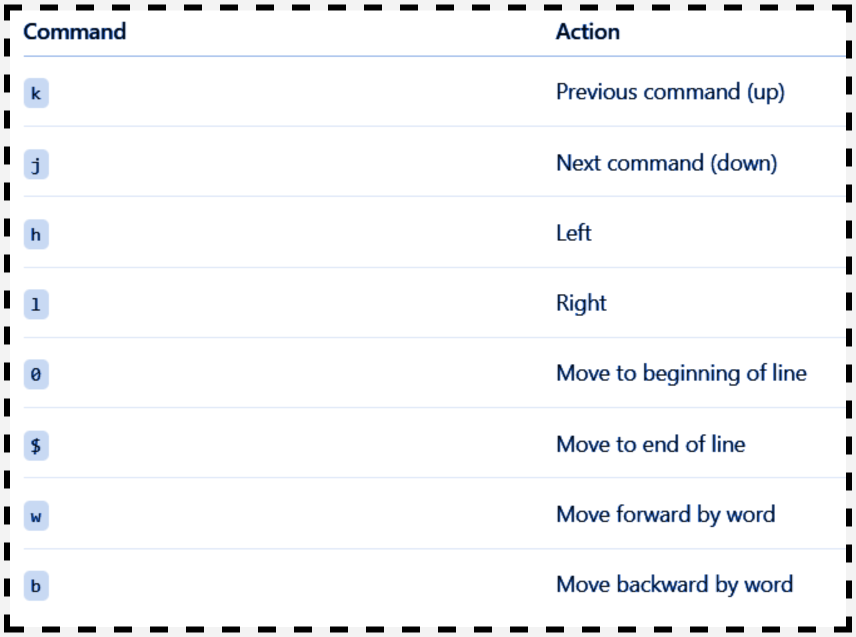
2.4.3 – Deletion Commands



Command	Action
`x`	Delete character under cursor
`dw`	Delete word
`dd`	Delete entire line
`D`	Delete from cursor to end

Example:
We want to remove everything from the cursor to the end of the line.
Press `D`.

2.4.4 – Moving Around in the History List



Command	Action
`k`	Previous command (up)
`j`	Next command (down)
`h`	Left
`l`	Right
`0`	Move to beginning of line
`$`	Move to end of line
`w`	Move forward by word
`b`	Move backward by word

Example: To go to the beginning of the line, press `0`. To move to the next word, press `w`.

2.4.5 – Character-Finding Commands

Command	Action
`fx`	Move forward to character `x`
`Fx`	Move backward to character `x`
`tx`	Move forward **before** character `x`
`Tx`	Move backward **before** character `x`

Example:
We're in command mode and want to jump to the next `/` character — press `f/`.

Thus,the vi editing mode in bash offers powerful and precise control over command-line editing, making it a valuable tool for experienced users. It operates using two distinct modes: *insert mode*, where you type and edit commands as usual, and *command mode*, where you can perform efficient navigation, deletion, and text manipulation using `vi`-style commands. This dual-mode structure enables quick edits without retyping entire commands. However, to use vi mode effectively, one must be familiar with the basic `vi` commands and keybindings. It is particularly well-suited for users who are already comfortable working in the `vi` or `vim` environment.

2.5 – The `fc` Command (Fix Command)

1.The `fc` command, short for "fix command," is a built-in bash utility that helps in retrieving and editing previously executed commands from the history list.

2.When I type `fc`, it opens the last executed command in the default text editor (usually `vi`). After editing and saving, the modified command runs automatically.

3.This is especially useful when I make a mistake in a long command. Instead of retyping it, I can easily correct it through the editor.

4.I can also use `fc` with arguments to specify which command I want to edit. For example, `fc 90` lets me edit the command with history number 90.

5.The `fc` command allows listing past commands using the `-l` option. For instance, `fc -l 10 15` lists commands from number 10 to 15.

6.Adding the `-n` option removes line numbers from the listing, while `-r` reverses the list order.

7.If I prefer a different editor, I can use `fc -e nano` to open the command in `nano` instead of `vi`.

8.To change the default editor permanently, I set the environment variable by adding `export FCEDIT=nano` in my `.bashrc` file.

9.Overall, `fc` is a convenient way for me to revisit, edit, and rerun commands efficiently without unnecessary retyping.

Example 1:

```bash
cp file1.txt /wrong/directory
```

But the directory was incorrect. We can fix this by typing:

```bash
fc
```

2.7 – readline

- The `readline` library is what bash uses to handle command-line editing. It enables me to move around the command line, edit text, search history, and more.
- It supports both emacs and vi editing modes, allowing flexible navigation and text manipulation based on my preference.
- `readline` makes features like command history, line editing, and key bindings work seamlessly in bash.

2.7.1 – The `readline` Startup File

- I can customize how `readline` behaves using a special file called `.inputrc` located in my home directory.
- This file lets me remap keys, enable features like case-insensitive completion, or change editing behavior.

2.7.2 – Key Bindings Using `bind`

- The `bind` command lets me change what certain keys do in the terminal.
- For example, I can assign a new shortcut that opens the current command in my favorite text editor when I press certain keys together like Ctrl + X followed by Ctrl + E.
- This is helpful when I'm typing a long command and suddenly want to switch to a full-screen editor to make changes.
- I can also use bind to check all the current keyboard shortcuts that are active, so I know which keys are already in use and which are free.
- If I prefer different keys for moving around or deleting text, I can remap them using `bind` so that bash feels more natural and comfortable for me.

2.8 KEYBOARD HABITS

1.Use Ctrl + A to move to the beginning of the line.

2.Use Ctrl + E to move to the end of the line.

3.Use Ctrl + F to move the cursor forward by one character.

4.Use Ctrl + B to move the cursor backward by one character.

5.Use Delete (DEL) or your terminal’s Erase key to delete the character before the cursor.

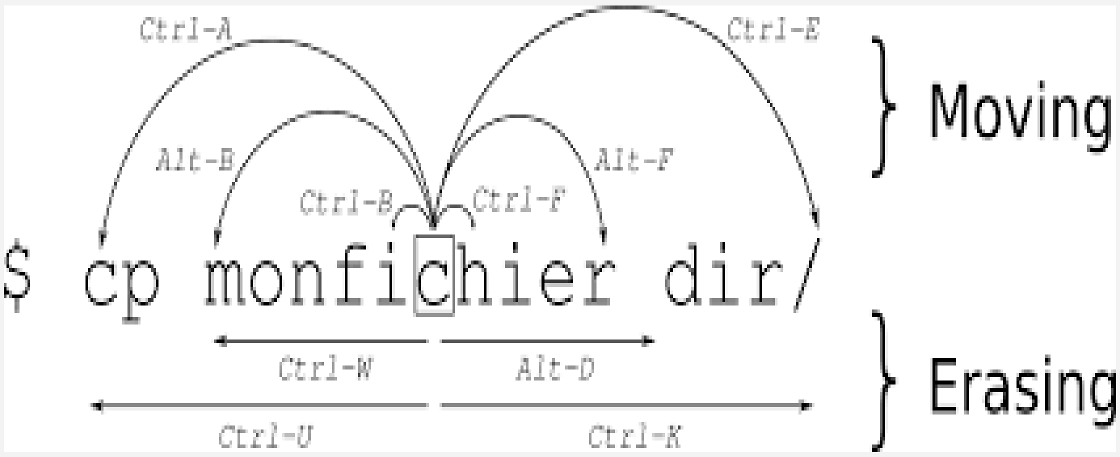
6.Use Ctrl + D to delete the character under the cursor.

7.Hold Ctrl + F, Ctrl + B, or Ctrl + D to repeat the movement or deletion continuously.

8.Use Ctrl + K to delete from the cursor to the end of the line.

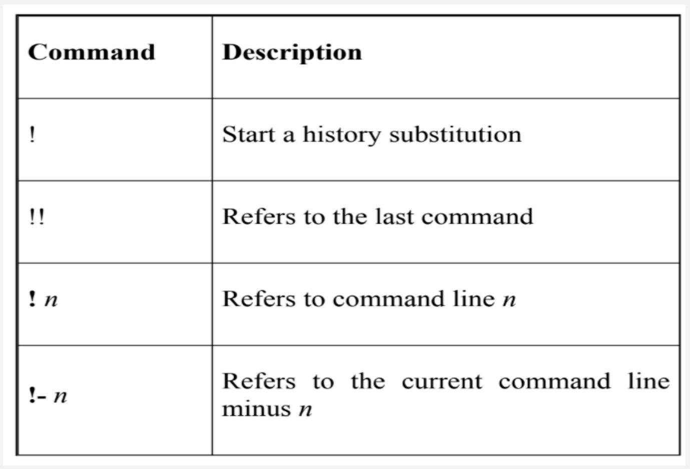
9.Use Ctrl + P to move to the previous command in history.

10.Use Ctrl + N to move to the next command in history.



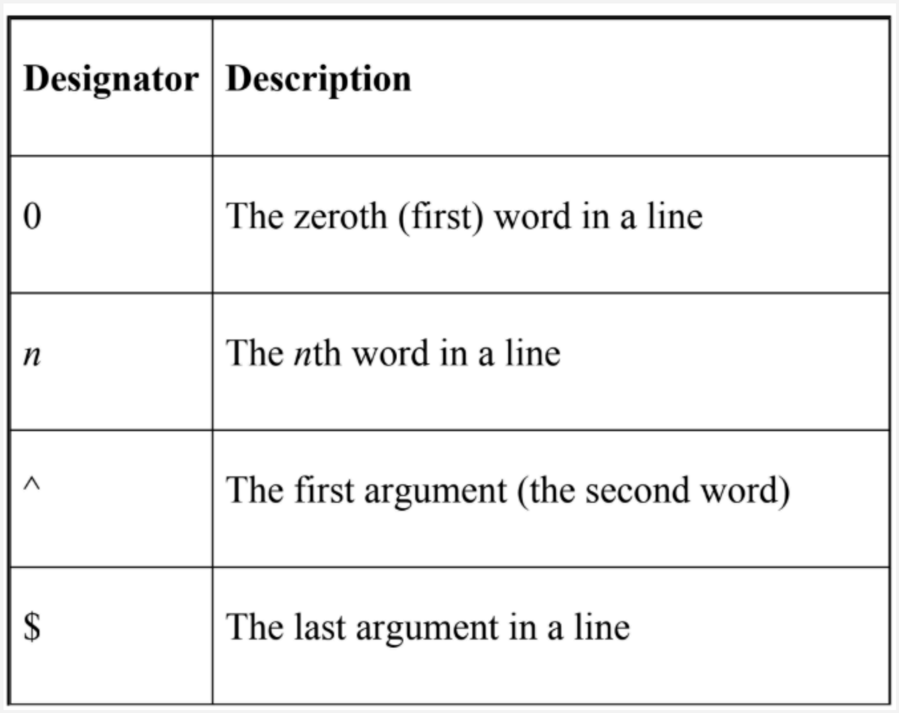
2.9 HISTORY EXPANSION

History expansion is a primitive way to recall and edit commands in the history list. The way to recall commands is by the use of event designators. Table gives a complete list.



Command	**Description**
!	Start a history substitution
!!	Refers to the last command
! *n*	Refers to command line *n*
!- *n*	Refers to the current command line minus *n*

It's also possible to refer to certain words in a previous command by the use of a word designator. Table lists available designators. Note that when counting words, bash (like most UNIX programs) starts counting with zero, not with one.



Designator	**Description**
0	The zeroth (first) word in a line
n	The *n*th word in a line
^	The first argument (the second word)
$	The last argument in a line

Summary

This chapter introduces the concept of command-line editing in bash, which allows users to interactively edit commands without retyping them entirely. It explains the two main editing modes supported by bash:

- **emacs mode** (default), which uses familiar keyboard shortcuts for efficient editing,

- **vi mode**, which mimics the vi editor and includes both insert and command modes for advanced control.

The chapter discusses how to navigate the command history list, reuse past commands, and improve productivity using built-in tools like the `fc` command, which lets users recall and edit previous commands in an editor.

It also covers history expansion features such as `!!` or `!n` to quickly re-execute past commands.

The readline library is introduced as the core component managing bash's interactive editing features. Users can customize this behavior using the `.inputrc` file and `bind` commands to remap keys and improve interaction.

Finally, the chapter emphasizes building good keyboard habits to minimize reliance on the mouse and improve command-line efficiency.