Notes 7 - Wildcards

Wildcards are special characters that let you match patterns instead of exact text. They're used in commands,searches, and file operations to quickly grab multiple items at once.

The * Wildcard

- **Definition:** The `*` wildcard means "match any number of characters." It can stand for zero,one,or many characters. Think of it as **"everything that fits this pattern."**
- **Examples:**
  - `*.txt` - matches all files ending with `.txt` (`notes.txt, todo.txt. etc`)
  - `pro*` - matches anything that starts with `pro` (`project` , `process`, `profile`, etc)
  - `*2024*` - matches anything with `2024` anywhere in the name.
  - `*` by itself - matches everything in a folder.
  - `ls Downloads/*` - list all files in the downloads directory
  - `ls Downloads/*.txt` - list all `.txt` files in Downloads
  - `ls Downloads/f*.txt` - lists `.txt` files starting with the letter **f**
  - `ls *file*` - lists all files containing the word **file** anywhere in the name

The ? Wildcard

- **Definition:** The `?` wildcard matches exactly one character. It like saying "I don't care what this one character is, but it must be there."
- **Examples:**
  - `files?.txt`- matches `file1.txt` or `fileA.txt`, but not `file10.txt`.
  - `??.png` - matches any two-character name ending in `.png (example ab.png)`
  - `report?.pdf` - matches `report1.pdf`, `report2.pdf`, etc.

The [set] Wildcard

- **Definition:** The [set] wildcard matches one character from a specific list or range that define inside the brackets. This gives you precise control over which characters are allowed.
- **Examples:**
  - `file[1-3].txt` - matches `file1`, `file2.txt`, `file3.txt`
  - `photo[a-c].jpg` - matches `photoa.jpg`, `photob.jpg`, `photoc.jpg`
  - `report[AB].pdf` - matches `reportA.pdf` or `reportB.pdf

- `[!a-z]*` - matches files not starting with a lowercase letter (the `!` means "not")

Brace Expansion

Definition:
Brace expansion is a shortcut that lets you create **many folders or files at the same time**.
Whatever you put inside `{ }` gets expanded into multiple names.
This saves you from typing the same command over and over.

Think of it like saying:
“Do the same thing, but for all of these options.”

Example 1 - Make several folders at once

```
mkdir project/{code,images,notes}
```

- This creates three folders:
  - project/code
  - project/images
  - project/notes
- **Why this helps:** One command creates three folders instantly.

```
mkdir -p class/{week1,week2,week3}/assignments
```

The creates:

- class/week1/assignments
- class/week2/assignments
- class/week3/assignments

Why this helps: You build a whole structures in one step instead of typing multiple commands.

```
mkdir logs/{1..5}
```

This creates the following folder:

- logs/1
- logs/2
- logs/3
- logs/4
- logs/5

Why this helps: Perfect for organizing files by number, date, or week.

`..` means **through** like `1-5`

`mkdir` - make folder