Top Bash Commands (Syntax and Flags)

File and Directory Management

1. **`ls`** - List files and directories
   Syntax: `ls [options] [file...]`
   Flags:
   - `-l` : Long listing format
   - `-a` : Show all files, including hidden ones
   - `-h` : Human-readable file sizes
2. **`cd`** - Change directory
   Syntax: `cd [directory]`
   Flags:
   - `~` : Home directory
   - `..` : Parent directory
   - `-` : Previous directory
   - `/` : Root directory
3. **`pwd`** - Print the current working directory
   Syntax: `pwd`
4. **`mkdir`** - Create directories
   Syntax: `mkdir [options] directory...`
   Flags:
   - `-p` : Create parent directories as needed
   - `-v` : Verbose mode
5. **`rmdir`** - Remove empty directories
   Syntax: `rmdir [options] directory...`
   Flags:
   - `--ignore-fail-on-non-empty` : Ignore errors when directory is not empty
   - `-p` : Remove directories and their parents
6. **`cp`** - Copy files or directories
   Syntax: `cp [options] source destination`
   Flags:
   - `-r` : Recursive copy (for directories)
   - `-i` : Prompt before overwrite
   - `-v` : Verbose mode, shows files as they are being copied

7. **`mv`** - Move or rename files or directories
   Syntax: `mv [options] source destination`
   Flags:
   - `-i` : Prompt before overwrite
   - `-v` : Verbose mode, show files as they are being moved or renamed
   - `-n` : No clobber, don't overwrite existing files
8. **`rm`** - Remove files or directories
   Syntax: `rm [options] file...`
   Flags:
   - `-r` : Recursive removal (for directories and their contents)
   - `-i` : Interactive mode, prompt before removal
   - `-f` : Force removal without prompt
9. **`touch`** - Create an empty file or update the timestamp of an existing file
   Syntax: `touch [options] file...`
   Flags:
   - `-a` : Change only access time
   - `-m` : Change only modification time
   - `-c` : Do not create any files

Text and Data Manipulation

10. **`nano`** - Text editor for Unix-like systems
    Syntax: `nano [file]`
    Flags:
    - `-B` : Create a backup file before editing.
    - `-m` : Enable mouse support.
    - `-R` : Open the file in read-only mode.
11. **`cat`** - Concatenate and display the content of files
    Syntax: `cat [options] [file...]`
    Flags:
    - `-n` : Number all output lines
    - `-E` : Display `$` at the end of each line
    - `-s` : Squeeze multiple blank lines
12. **`grep`** - Search for patterns within files
    Syntax: `grep [options] pattern [file...]`
    Flags:
    - `-i` : Ignore case

    - `-r` : Recursive search
    - `-v` : Invert match, show lines that do not match
13. **`sed`** - Stream editor for filtering and transforming text
    Syntax: `sed [options] 's/pattern/replacement/' [file]`
    Flags:
    - `-i` : Edit files in place.
    - `-e` : Add the script to the commands to be executed.
    - `-n` : Suppress automatic printing of pattern space.
14. **`awk`** - Pattern scanning and processing language
    Syntax: `awk 'pattern { action }' [file]`
    Flags:
    - `-F [separator]` : Specify the field separator.
    - `-v [var=value]` : Assign a variable.
15. **`echo`** - Display a line of text
    Syntax: `echo [options] [string...]`
    Flags:
    - `-e` : Enable interpretation of backslash escapes
    - `-n` : Do not output the trailing newline
    - `-E` : Disable interpretation of backslash escapes
16. **`sort`** - Sort lines of text files
    Syntax: `sort [options] [file...]`
    Flags:
    - `-r` : Reverse the result of comparisons
    - `-n` : Compare according to string numerical value
    - `-o` : Write result to a file instead of standard output
17. **`head`** - Output the first part of files
    Syntax: `head [options] [file...]`
    Flags:
    - `-n [number]` : Print the first `number` lines
    - `-c [bytes]` : Print the first `bytes` bytes
18. **`tail`** - Output the last part of files
    Syntax: `tail [options] [file...]`
    Flags:
    - `-n [number]` : Output the last `number` lines
    - `-f` : Follow the file as it grows
    - `-c [bytes]` : Output the last `bytes` bytes

System Monitoring and Management

19. **`ps`** - Display information about running processes
    Syntax: `ps [options]`
    Flags:
    - `-aux` : Show all processes
    - `-ef` : Display processes in full-format listing
    - `-p` : Display processes by PID
20. **`htop`** - Interactive process viewer
    Syntax: `htop`
    Flags:
    - `-u [user]` : Show only the processes of a specific user.
    - `-p [PID]` : Show only the processes with a given PID.
    - `-s [column]` : Sort by a specific column.
21. **`kill`** - Send a signal to a process
    Syntax: `kill [options] PID`
    Flags:
    - `-9` : Force kill
    - `-l` : List signal names
    - `-s` : Send specified signal
22. **`df`** - Report file system disk space usage
    Syntax: `df [options] [file...]`
    Flags:
    - `-h` : Human-readable format
    - `-T` : Show file system type
    - `-i` : Display inode information
23. **`ping`** - Send ICMP ECHO_REQUEST to network hosts
    Syntax: `ping [options] destination`
    Flags:
    - `-c [count]` : Stop after sending `count` requests
    - `-i [interval]` : Wait `interval` seconds between sending each packet
    - `-t [ttl]` : Set the IP Time to Live
24. **`history`** - Show the command history
    Syntax: `history [options]`
    Flags:
    - `-c` : Clear the history list

    - `-d [offset]` : Delete the history entry at `offset`
    - `-w` : Write the current history to the history file
25. **`crontab`** - Schedule periodic commands
    Syntax: `crontab [options]`
    Flags:
    - `-e` : Edit the current user's crontab
    - `-l` : List the current user's crontab
    - `-r` : Remove the current user's crontab
26. **`env`** - Display, set, or remove environment variables
    Syntax: `env [options] [name=value...] [command [args...]]`
    Flags:
    - `-i` : Start with an empty environment
    - `--unset=[name]` : Remove variable from the environment

File and Data Transfer

27. **`scp`** - Securely copy files between hosts
    Syntax: `scp [options] [source] [user@host:destination]`
    Flags:
    - `-r` : Recursively copy entire directories.
    - `-P [port]` : Specify the port number for the remote host.
    - `-C` : Enable compression during transfer.
28. **`wget`** - Retrieve files from the web
    Syntax: `wget [options] [URL]`
    Flags:
    - `-c` : Continue getting a partially downloaded file.
    - `-r` : Download files recursively.
    - `-P [prefix]` : Set the directory prefix for saving files.
29. **`curl`** - Transfer data from or to a server
    Syntax: `curl [options] [URL]`
    Flags:
    - `-o [file]` : Write output to a file instead of stdout.
    - `-O` : Save the file with the same name as the remote file.
    - `-I` : Fetch the headers only.

Text and Data Search

30. **`find`** - Search for files and directories in a directory hierarchy
    Syntax: `find [path...] [expression]`
    Flags:
    - `-name` : Search by name
    - `-type` : Search by type (e.g., `f` for files, `d` for directories)
    - `-exec` : Execute a command on the found items

Permissions and Ownership

31. **`chmod`** - Change the permissions of a file
    Syntax: `chmod [options] [who][operator][permission] file...`
    Flags:
    - `-R` : Recursive change
    - `-v` : Verbose mode
    - `--reference` : Use file or directory permissions as reference

    Explanation:

    - **Who**: `u` (User/owner), `g` (Group), `o` (Others), `a` (All)
    - **Operator**: `+` (Add), `-` (Remove), `=` (Set exact)
    - **Permission**: `r` (Read), `w` (Write), `x` (Execute)

    Alternatively, permissions can be set using three digits representing:

    - `r` (Read) = 4
    - `w` (Write) = 2
    - `x` (Execute) = 1
    - The three digits correspond to the user, group, and others.
32. **`chown`** - Change the ownership of a file
    Syntax: `chown [options] [owner][:group] file...`
    Flags:
    - `-R` : Recursive change
    - `-v` : Verbose mode
    - `--reference` : Use file or directory ownership as reference

    Explanation:

    - **Owner**: The user who owns the file.

- **Group**: The group that owns the file.
- You can change just the owner, just the group, or both by specifying `[owner][:group]`.

Archiving and Compression

33. **`tar`** - Archive files
    Syntax: `tar [options] [archive] [file...]`
    Flags:
    - `-c` : Create a new archive
    - `-x` : Extract files from an archive
    - `-z` : Compress with gzip
    - `-f` : Specify name of the archive file

Remote Access

34. **`ssh`** - Connect to a remote server via SSH
    Syntax: `ssh [options] user@host`
    Flags:
    - `-i` : Specify identity file
    - `-p` : Specify port
    - `-L` : Local port forwarding

CONDITIONALS AND LOOPS

1. `if-else` - Conditional statements

Syntax:

```
if [ condition ]; then
    # commands
elif [ condition ]; then
    # commands
else
    # commands
fi
```

Flags:

- `-eq` : Equal to
- `-ne` : Not equal to
- `-gt` : Greater than
- `-lt` : Less than

When to Use: Use `if-else` when you need to execute a block of code based on a specific condition. It's ideal for decision-making where you have one or more conditions to check.

2. `case` - Multi-way branch statement

Syntax:

```
case value in
    pattern1)
        # commands
        ;;
    pattern2)
        # commands
        ;;
    *)
        # default commands
        ;;
esac
```

Flags:

- `*` : Default case
- `|` : Or pattern separator
- `;;` : Terminate a block of commands

When to Use: Use `case` when you need to compare a single value against multiple patterns. It's useful for handling multiple conditions with more clarity and less repetition than using multiple `if-elif-else` statements.

3. `for` - Loop over a list of items

Syntax:

```
for item in [list]; do
    # commands
done
```

Flags:

- `{1..10}` : Sequence expression
- `*` : All files in directory
- `$@` : All command-line arguments

When to Use: Use `for` loops when you want to iterate over a list of items, such as files in a directory, numbers, or command-line arguments. It's perfect for running the same set of commands for each item in a list.

4. `while` - Loop while a condition is true

Syntax:

```
while [ condition ]; do
    # commands
done
```

Flags:

- `true` : Infinite loop
- `break` : Exit the loop
- `continue` : Skip to the next iteration

When to Use: Use `while` loops when you want to repeat a set of commands as long as a condition remains true. It's suitable for situations where the number of iterations isn't known in advance.

5. `until` - Loop until a condition becomes true

Syntax:

```
until [ condition ]; do
```

```
    # commands
done
```

Flags:

- `true` : Infinite loop
- `break` : Exit the loop
- `continue` : Skip to the next iteration

When to Use: Use `until` loops when you want to repeat a set of commands until a condition becomes true. It's the opposite of `while` and is useful when you need the loop to continue until a specific condition is met.

GIT COMMANDS

1. `gh auth login` - Authenticate with GitHub

Syntax:

```
gh auth login
```

Flags:

- `--web` : Authenticate via web browser.
- `--with-token` : Authenticate using a GitHub token.

When to Use: Use `gh auth login` to authenticate your GitHub account via the command line. This command sets up your GitHub CLI with the necessary credentials to interact with your GitHub repositories.

2. `git init` - Initialize a new Git repository

Syntax:

```
git init [repository-name]
```

When to Use: Use `git init` to create a new Git repository in an existing directory or a new repository. This command sets up all necessary files for Git to track changes.

3. `git clone` - Clone a repository into a new directory

Syntax:

```
git clone [repository-url] [directory-name]
```

Flags:

- `-q` : Operate quietly.

When to Use: Use `git clone` to copy an existing Git repository from a remote server to your local machine. This is typically the first step when you want to work on a project hosted on platforms like GitHub.

4. `git add` - Add file contents to the staging area

Syntax:

```
git add [file...]
```

Flags:

- `.` : Add all files in the current directory.

When to Use: Use `git add` to stage changes that you want to include in the next commit. This command is necessary before committing changes with `git commit`.

5. `git commit` - Record changes to the repository

Syntax:

```
git commit -m "commit message"
```

Flags:

- `-m` : Add a message to your commit.

When to Use: Use `git commit` to save your staged changes in the repository. Every commit creates a snapshot of your project, which you can revert to if needed.

6. `git status` - Show the working tree status

Syntax:

```
git status
```

When to Use: Use `git status` to view the current state of your working directory and the staging area. This command shows which files have been modified, which are staged for commit, and which are untracked.

7. `git log` - Show the commit history

Syntax:

```
git log
```

Flags:

- `--oneline` : Display a compact version of the log.

When to Use: Use `git log` to view the commit history of the repository. This command is useful for tracking changes over time and understanding the project's history.

8. `git branch` - List, create, or delete branches

Syntax:

```
git branch [branch-name]
```

Flags:

- `-d` : Delete a branch.

When to Use: Use `git branch` to manage branches in your repository. Branches allow you to work on different features or fixes independently of the main codebase.

9. `git checkout` - Switch branches or restore working tree files

Syntax:

```
git checkout [branch-name]
```

Flags:

- `-b` : Create and switch to a new branch.

When to Use: Use `git checkout` to switch between branches in your repository. This command is essential for working on different parts of your project without affecting the main codebase.

10. `git merge` - Join two or more development histories together

Syntax:

```
git merge [branch-name]
```

When to Use: Use `git merge` to integrate changes from one branch into another. This command is commonly used to bring feature branches into the main branch.

11. `git pull` - Fetch and merge changes from the remote repository

Syntax:

```
git pull [remote-name] [branch-name]
```

When to Use: Use `git pull` to update your local repository with changes from a remote repository. This command combines `git fetch` and `git merge` into one step.

12. `git push` - Update the remote repository with local commits

Syntax:

```
git push [remote-name] [branch-name]
```

When to Use: Use `git push` to upload your local commits to a remote repository. This command is essential for sharing your work with others