# Introduction to the Command Line: Takeaways

## Syntax

- The anatomy of commands:
  - `command parameter1 parameter2 ... parameterN`
  - `command -options arguments`

## Concepts

- A **command** is a text instruction given to the computer.
- Command behavior can be modified with options.
- A **Shell**, **Command Language Interpreter** or **Command-line Interface** is a type of program that interprets and runs text commands. In this course we are using Bash.
- A **Terminal Window**, **Terminal Emulator** or **Terminal** is a program that people can use to send input to the shell.

## Resources

- Ubiquitous options
- List of command language interpreters
- POSIX standards and The Open Group
- Windows Subsystem for Linux installation instructions
- Overview of the `diff` command
- The `history` command
- History expansion

# The Filesystem: Takeaways

## Syntax

- Listing the contents of a directory.
  - Listing the non-hidden contents of the current directory without any options: `ls`
  - Listing the non-hidden contents of path `/home/dq` : `ls /home/dq`
  - Listing the non-hidden contents of the current directory in long format: `ls -l`
  - Listing all contents of the current directory: `ls -a`
  - Listing all contents of the current directory except for the directories `.` and `..` : `ls -A`
  - Listing all contents of `/home/dq` in long format, except for the directories `.` and `..` : `ls -Al`
- Changing directories:
  - Change to directory `/home` : `cd /home`
  - Change to the parent directory of the current directory: `cd ..`
  - Change to the parent directory of the parent directory of the current directory: `cd ../..`
  - Change to your home directory: `cd`
  - Change your home directory: `cd ~`
  - Change to the home directory of user dq: `cd ~dq`
  - Change to the previous directory: `cd -`

## Concepts

- Files are organized in a **hierarchical directory structure**. It is an organizational system for files and directories, in which files and directories are contained in other directories.
- A **path** is a sequence of slashes and files and directory names that define the location of a file or directory.
  - An **absolute path** is any path that starts with a slash. It tells us how to go from the root directory to the location of the the file specified by the path.
  - All others paths are **relative paths**. They tell us how to go from the current directory to the location of the the file specified by the path.
- The **root directory** is defined by the path `/` . It is the only directory that isn't contained in any other directory.
- The **home directory** of user `<username>` is `/home/<username>` .

## Resources

- The Filesystem Hierarchy Standard as defined by the Linux Foundation here.

# Modifying the Filesystem: Takeaways

## Syntax

- Creating a directory called `my_dir` : `mkdir my_dir`
- Deleting an empty directory called `my_dir` : `rmdir my_dir`
- Creating a copy of file `my_file1` as `my_file2` : `cp my_file1 my_file2`
- Copying files interactively: `cp -i source destination`
- Create a copy of directory `my_dir1` as `my_dir2` : `cp -R my_dir1 my_dir2`
- Deleting file `my_file` : `rm my_file`
- Deleting the non-empty directory `my_dir` : `rm -R my_dir`
- Moving `my_file` to `my_dir` : `mv my_file my_dir` .
- Renaming `my_file` as `our_file` : `mv my_file our_file` .

## Concepts

- It's not easy to restore files after we delete them from the command line.
- We need to be very careful when using `rm` , `cp` , and `mv` as they might cause us to lose important files.

# Glob Patterns and Wildcards: Takeaways

## Syntax

- Wildcards:
    - `?` matches any single character.
    - `*` matches any string of characters.
    - `[list_of_characters]` matches any characters in `list_of_characters` .
    - `[!list_of_characters]` matches any characters **not** in `list_of_characters` .
    - `[[:alpha:]]` matches any letter.
    - `[[:digit:]]` matches any number.
    - `[[:alnum:]]` matches any letter or number.
    - `[[:lower:]]` matches any lowercase letter.
    - `[[:upper:]]` matches any uppercase letter.

## Concepts

- We can use **wildcards** to create patterns to match groups of filenames.
- These patterns, called **glob patterns**, work in a similar way to regular expressions, albeit with different rules.
- We can use glob patterns with most commands, making them an extremely powerful tool.
- Because they're very powerful, we need to be careful with them, especially when it comes to commands that modify the filesystem (like `rm` ).

## Resources

- Character classes in GNU.
- Globbing and Regex: So Similar, So Different.
- Glob patterns and regular expressions summary.
- The glob function.
- Locale.
- `find` :
    - How to Find a File in Linux Using the Command Line
    - 35 Practical Examples of Linux `find` Command
    - Unix Find Tutorial
- The `locate` command — an alternative to `find` :
    - Linux `locate` command
    - 10 Useful `locate` Command Practical Examples for Linux Newbies

# Users and Permissions: Takeaways

## Syntax

- Identifying users and their groups
    - `whoami`
    - `id`
    - `groups`
- See `file` 's metadata: `stat file`
- Changing permissions:
    - Symbolic notation: `chmod [ugoa][+-][rwx] files` .
        - Adding execution permission to the owner on `file` : `chmod u+x file` .
        - Removing writing permission to the primary group on `file` : `chmod g-w file` .
        - Setting read and execution permissions to others on `file` : `chmod o=rx file`
        - Changing several permissions simultaneously on `file` : `chmod u+w,g-x,o-r file` .
    - Octal notation: `chmod ddd` where `d` represents a digit between `0` and `7` .
        - `---` : `0` (no permissions)
        - `--x` : `1` (execute only permission)
        - `-w-` : `2` (write only permissions)
        - `-wx` : `3` (write and execute permissions)
        - `r--` : `4` (read only permissions)
        - `r-x` : `5` (read and execute permissions)
        - `rw-` : `6` (read and write permissions)
        - `rwx` : `7` (read, write, and execute permissions)
- Changing ownership on `file` : `chown [new_owner][:new_group] file`
    - Changing both the ownership and the group of `file1` : `sudo chown new_owner:new_group file` .
    - Changing the ownership of `file` while maintaining its group: `sudo chown new_owner file` .
    - Changing the group of `file` while maintaining its ownership: `sudo chown :new_group file` .
- Running command with superuser privileges: `sudo command`

## Concepts

- Operating systems implement the concept of users.
- In Unix-like systems, everything is a file.
- Files have owners and group owners.

- Permissions are limits to the actions that users can perform.
- Permissions are a property of both files and users.
- To facilitate managing permissions, there is also the concept of group (of users). Groups also have permissions.
- Some users (like the superuser) have permissions to do everything.
- Users can elevate their priveleges to that of the superuser. Extra care is needed when using this power.
- In *nix systems, users can elevate their privileges with `sudo` .

## Resources

- The origin of "Everything is a file".
- The setuid and setgid permission bits.
- Difference between symbolic link and shortcut
- Identifying file types in Linux
- POSIX standards on `chmod`
- The Uppercase X in `chmod`
- Effective user and real user
- Changing default permissions on file creation