

# CLI COMMANDS

---

Make the CLI do what you need



# PREFACE

When you launch your terminal (which then launches a shell to pass your commands to) we have the ability to type commands into the terminal and if they are correct, they take an action. Where do these commands come from?

When I type **mkdir** my shell understands this command, when I type **askjdgaslhgn** or any other random garbage my shell understands this is an invalid command. How does the shell know what a valid command vs an invalid command is?

When you type a command into your terminal, the shell is really just looking through a list of **executables** to see if any of their names match your typed command.

Where does this list of executables come from? Usually one of two places:

1. The shell's built-in commands (controlled by the shell)
2. The **PATH** list of locations (modifiable by you)

# PREFACE CONT.

1. The shell's built-in commands (controlled by the shell)
2. The **PATH** list of locations (modifiable by you)

The shells built in commands come with the shell that you installed. We will be using **bash** (Bourne Again Shell) as our shell. It comes installed on Linux, Mac, and Git Bash. This shell comes with a small list of commands that you get for free when you install the shell. By default Mac users will be using z-shell, but our commands will almost always be the same.

The **PATH** variable is something common to almost all operating systems. It is a list of **folder locations** on your machine. If any of these folders contain **executables**, your shell can execute them (if you have permission to do so). We will get experience manipulating this PATH variable soon, but for now just know it is simply a list of locations on your computer used to tell your shell where to look for executables.

# A WORD ABOUT YOUR FILE SYSTEM

Before we dive into rocking the command line, we need to clear up your understanding of your computer's file system a bit. This is the system that controls your files and folders.

You have probably never thought about how your computer's file system works, but it is very important and an understanding of this will help you navigate the command line more successfully.

We aren't going to talk about things like mounting drives and such as that is not needed for solid command line use. We are going to focus on how to think about folders and files on your system.

# DIRECTORIES

A directory is just another term for folder. Feel free to use them interchangeably, but understand chances are only people technically versed will understand directory.

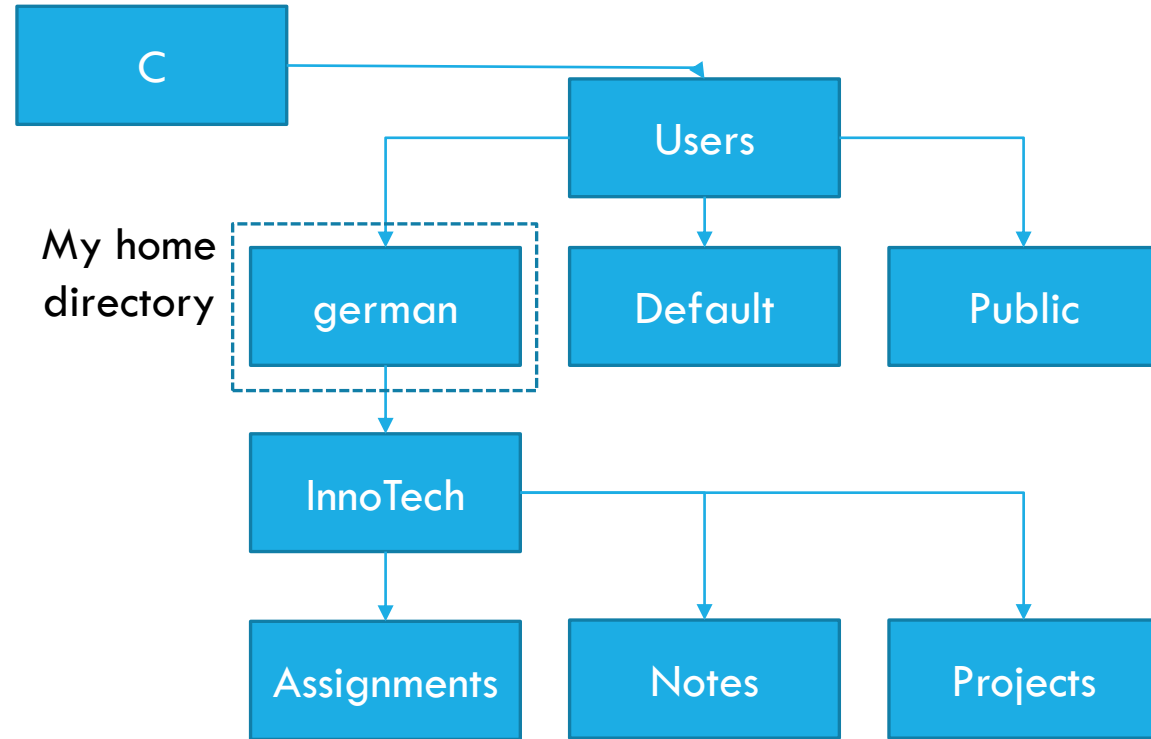
All operating systems organize themselves into directories. Directories have what is called a **parent-child relationship**. This is a fancy term that means directories can contain other directories, and they can organize themselves into a family tree structure.

You have probably never thought about your file system as a family tree structure, but I will prove to you it makes sense.

Beyond child directories, directories can also store files. This shouldn't be a shock to you, when you have a file you have to choose which folder to put it in.

# DIRECTORIES

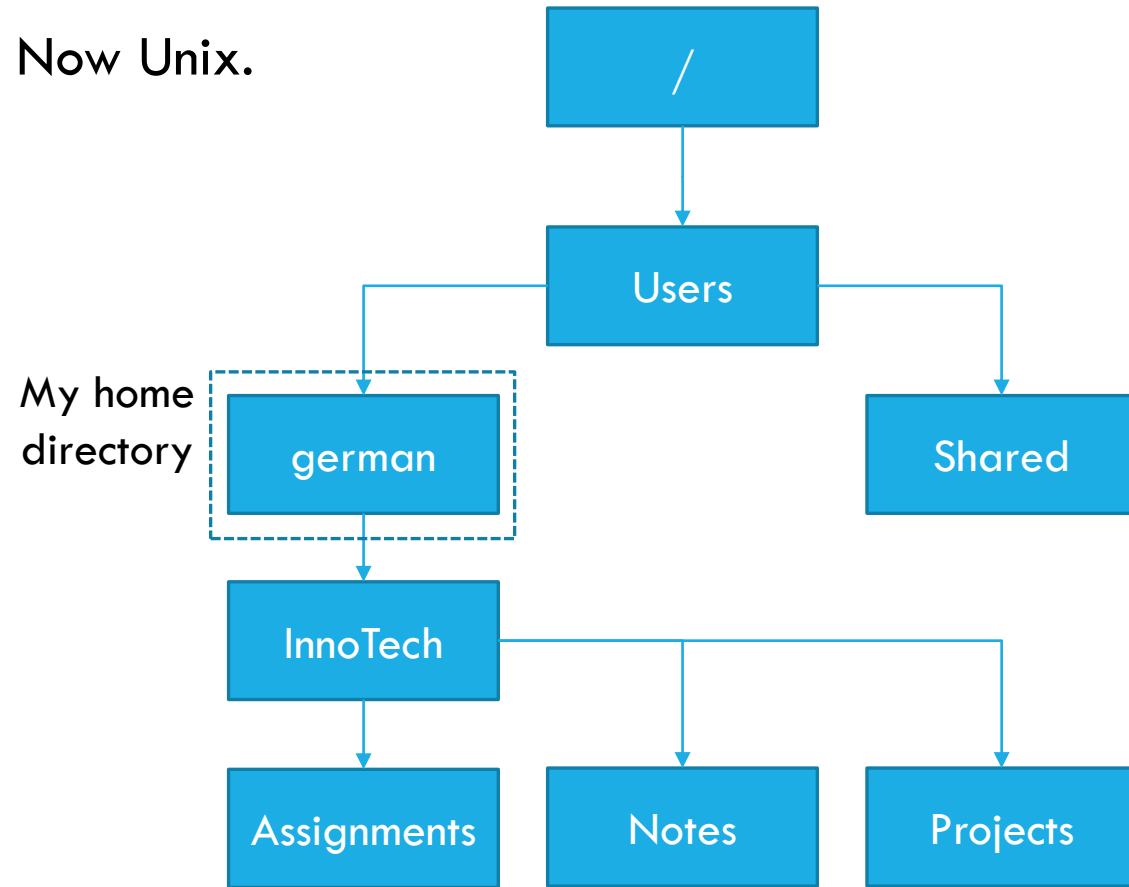
We will look at an example of both Windows and Unix (Mac or Linux) of your **home directory**. First Windows.



I have left out many directories on purpose (there are simply too many to show) but this still illustrates the point. The **Users** directory on my computer has 3 children directories in **german**, **Default** and **Public**. The parent to the **InnoTech** directory is the **german** directory.

# DIRECTORIES

Now Unix.



Again I have left out many directories to simplify the graph, but parent-child relationship remains. In the GUI, you are used to simply clicking on the folder in your OS's file explorer to change directories. We will learn in later slides how to accomplish this on the command line.

# MOVING AROUND

Before we get into command line stuff to move around, let's all just make sure we can navigate our folders in the GUI.

On Mac:

1. Open the Finder
2. Press CMD + Shift + G
3. Type / into the popup
4. Click through the folders until you reach your InnoTech Folder

On Windows:

1. Open the File Explorer
2. Type C:\ into the bar at the top
3. Click through the folders until you reach your InnoTech Folder



# PATHS

A short note on *paths*. A path is really just any valid way to traverse your file structure. There are two types of paths, **relative** and **absolute**.

A relative path is one that starts from the directory I am currently in. For example, if I am currently in the directory **/Users/german** from the previous slide, a relative path would be **InnoTech/Notes/**.

An absolute path is one that starts at the *root* of your computer. You can tell it is an absolute path when it starts with a **/** character. For example, an absolute path from the previous slide would be **/Users/german/InnoTech/**. It does not matter what my current directory is, that path is always the same.

The distinction between the types of paths will become important when running commands.

# PATHS CONT.

A nice way to think of the two different paths is as follows:

- A relative path means you are moving **through direct relatives**.
  - This means you can move into your parent or any children directories, but you must pass through a direct line **relative to your current location** to move anywhere.
- An absolute path means you disregard your current location and always start at the root of the family tree.
  - This means you don't have to start your path with either your parent or any children, you always start at the root. An absolute path is just that, absolute. The path to your home directory on Mac is always `/Users/username` and doesn't change based on your location.

# COMMAND ANATOMY

Before we learn any specific commands, let's go over what a generic command looks like.

***CommandName* <User Options> <User Input>**

Let's break this down:

1. **CommandName** – Simply the name of the executable you want to run. This changes depending on what you are trying to achieve
2. **<User Options>** - Most commands take options that change the way the executable behaves. Each command has different options available.
3. **<User Input>** - Most (but not all) commands need some kind of input from the user to work. For example, the command **mkdir** which creates a directory (folder) needs to be given the name it should give the folder.

Please note that the angled brackets <> are just there to make the generic example more clear, they are not actually used in practice.

# WARNING

This message is for those of you that like to put spaces in their file/directory names, **STOP!**

In the command line environment, spaces are usually used to denote the end of a command or option. So when your file and folder names contain spaces, you have to give them special treatment to get things to work which is doable, but very annoying.

Save yourself the trouble, instead of naming things with spaces, simply use either the `-` or the `_` characters instead.

Also, the mouse does not help you in the CLI. You need to use the arrow keys to move your cursor instead of being able to click like you are used to in things like document editors.

# COMMAND USAGE

Before we start learning any specific commands, you must learn a very important principle about commands: **everything is documented.**

In older times, if you were to approach a senior engineer asking how to use a specific command, they could ask you to kindly RTFM, which of course is a suggestion to use the command **man <command name>**. (Windows users, you are out of luck, so just use **<command> --help** or Google).


No one expects anyone to remember the syntax and all the available options for every single available command, so it's okay to take a few times before the command you are trying to run actually does what you want it to do (with a few notable irreversible exceptions).

# BASIC COMMANDS

We are starting to get a little bit more comfortable with the idea of the CLI, so let's get our hands dirty. The next handful of slides go into detail explaining some of the basic commands that will get you comfortable doing basic operations.

There are **many** commands to learn, and an absurd amount of commands that you can download and add to your PATH yourself. The goal is to understand how commands work in general. With this understanding, you can learn any command by simply reading about what the command does and how to use it (which options and input are needed).

Before we jump in, one thing that might be a little odd to you is that in many cases when a command works in this world, there is no feedback. This is something you will have to get used to at first (nothing is good, errors are bad).



The `cd` command is short for “change directory”. As the name suggests, this will allow you to *change* which directory (folder) you are currently in.

- A small side note, when you open your terminal, it always defaults to starting you in your home directory.

## CD

As we learnt about before, directories have a **parent-child relationship**. To navigate through directories, we simply follow family tree lines.

The `cd` command takes the name of the directory you want to change into, and moves you there.

Try using **cd** combined with `~`, `/`, `.` or `..` as arguments. Check where you ended up, what happens?

# CD CONT.

## Examples:

### **cd ~/InnoTech/Assignments/Week1**

- This will move me into the directory supplied (/c/Users/german/InnoTech/Assignments/Week1)

### **cd js**

- This will change you into the folder js as long as it exists in your **current working directory**

### **cd ..**

- This will change you the **parent directory** of the current directory. This is also known as **moving up** a directory

### **cd**

- When nothing is provided to your cd command, it will put you in your **home directory**. This is useful if you ever get really lost.



Because we don't have a GUI to show an icon or picture of the contents of a folder, we need a command that shows all of the files and directories inside a directory.

This is where the **ls** (short for *list*) command comes into play. Simply type **ls** into your terminal and it will list all visible files and directories in your current directory.

## LS

If you want to see the contents of a folder you are not currently in, you can simply pass the name path to the directory to the **ls** command.

### Example:

**ls /c/Users/german**

- This will print the contents of my home directory to the terminal.

### **ls**

- This will print the contents of which ever directory I am currently in.

**Useful option:** **-a** will include all hidden files

# PWD

If you ever get lost and want to see exactly where you are on your computer, the `pwd` command will bail you out.

The **`pwd`** command is short for **print working directory** and it is handy if you ever need to confirm where you are.

## Example:

```
pwd
```

- This will print the **full path** to the directory you are currently in.

# KNOWLEDGE CHECK

The combination of `cd`, `ls` and `pwd` is enough to get you moving around in your terminal environment. Just as before, let's try to get to our InnoTech folder from the **root** of our computers. Open up VSCode, pull up your terminal and try the following:

## Windows:

1. Type `cd /c` which will put you in the root of your computer.
2. Using `cd`, `ls` and `pwd` make your way to your InnoTech folder

## Mac:

1. Type `cd /` which will put you in the root of your computer
2. Using `cd`, `ls` and `pwd` make your way to your InnoTech folder

If you ever get lost, just use `pwd` to see where you are and `ls` to see where you can go. If you ever get really lost, just go back to step one and start at the root.

# KNOWLEDGE CHECK CONT.

This time everyone can start the same:

1. Move to your home directory
2. Move to your Downloads directory
3. List all the files and folders in here on the terminal
4. Go back to your home directory
5. Navigate to your InnoTech directory
6. Go into your Assignments directory
7. Print the **current path** onto the terminal
8. Go **back up** into your InnoTech directory
9. Print the **current path** onto the terminal

# TOUCH

There will be many instances where you will want to create a file. The **touch** command does exactly that. It will create a file at the given location.

Note this command does **not** make directories. It will only make files.

**Important:** If your input contains a space character, you must wrap the input with quotes “.

## Example:

### **touch index.html**

- This will create the file in whichever directory you are currently in.

### **touch /c/Users/german/InnoTech/Assignments/index.html**

- This will create a file named *index.html* in the directory path provided. Note that this path can be **relative** or **absolute**.

# MKDIR

Just as you will need to make files, we will also need to make directories to organize our computers. The **mkdir** command accomplishes this.

Simply pass the name of a directory to create the directory.

**Important:** If your input contains a space character, you must wrap the input with quotes “”.

## Example:

### **mkdir assignments**

- This will create a directory named *assignments* in whichever directory you are currently in.

### **mkdir /c/Users/german/InnoTech/Assignments/week\_one**

- This will create a directory named *assignments week one* in the directory path provided.

**Useful option:** -p creates the entire tree at once

# RM

If you ever need to remove a file or directory, **rm** will get the job done for you. Simply pass the name of the file or directory you want to delete and watch it disappear.

**Warning** the rm command is (mostly) non-reversible. It is not like your GUI deletion where it goes to a recycle bin. If you **rm** a file or directory it should be considered gone. You have been warned.

**Important:** If your input contains a space character, you must wrap the input with quotes “”.

## Example:

### **rm index.html**

- If the file index.html exists in the current working directory, it will be deleted

### **rm -r projectOne**

- **Note:** The **-r** stands for recursive, and it tells the rm command to also delete directories and everything inside of them.
- This will delete either a directory or file name *projectOne* in your current working directory

# KNOWLEDGE CHECK

1. Move to your home directory
2. Navigate to your InnoTech directory
3. Create a **folder** called **Temp**
4. Go into that new folder
5. Create a **file** called index.html
6. Delete that file
7. Navigate **back** to your InnoTech Directory
8. Delete the **folder** called **Temp**



# MV

There will be times where you want to move a file from one directory to another. The **mv** command will help you in this scenario.

The **mv** command takes **two** inputs. The first is the file or directory you want to move, the second is the destination directory and they are separated by a space.

**Important:** If your input contains a space character, you must wrap the input with quotes “”.

## Example:

### **mv index.html assignments/week1**

- If the file index.html exists in the current working directory, it will be moved into the assignments/week1 directory
- Note these are both **relative paths** meaning both these paths must be located in the current working directory

### **Useful options:**

- n will prevent overwriting a file if one with an identical name exists in the destination.
- u will only overwrite if the source file is newer

# MV CONT.

The mv command can also be used to rename a file, all you have to do is give the original filename, and the new file name separated by a space. The only constraint is the original file must exist.

## Example:

**mv index.html about.html**

- If the file index.html exists in the current working directory, it will be renamed to about.html

# CAT

If you need to quickly look at a file, the **cat** command will print a file to the terminal for you. Just pass the path to the file you want to print out and the content of the file will show up on the terminal.

**Important:** If your input contains a space character, you must wrap the input with quotes “”.

## Example:

**cat assignments/week1/about.html**

- If the file about.html exists in the given path, the contents of the file will print to the terminal

# KNOWLEDGE CHECK

1. Move to your home directory
2. Navigate to your InnoTech directory
3. Create a **file** called **temp.html**
4. **Rename** that file to be new.html
5. Open this file with VSCode
6. Add whatever content you want
7. In the terminal, display the contents of this file
8. Delete the new.html file

# ECHO

If you want something to print out to the terminal, the **echo** command is your friend. This is handy when you start writing scripts that should give feedback to the user.

Within “ characters, pass the message to echo to the terminal and it will print for you. This seems useless, but as we will learn in the next and when scripting this is actually very handy.

## Example:

**echo ‘Finished! Go create something awesome!’**

- You will see the message print on the terminal.

>

The last commands we are going to look at is slightly more advanced. The `>` command is used to **send the output of one command into a file**. This is handy in practice when you want to capture error message for logs and such.

The form using `>` is as follows: *command > path*

**Important:** If your input contains a space character, you must wrap the input with quotes “”.

**Warning:** This command will overwrite anything currently in the file. Be careful.

**Example:**

**echo ‘Hey there!’ > greeting.txt**

- If the `greeting.txt` file exists, it will **overwrite** the content and replace it with *Hey there*. If the file does not exist, it will be created.

>>

If you don't want to overwrite the content of a file and instead just add some content to the end (appending) you can use >> instead.

The form using >> is as follows: *command >> path*

**Important:** If your input contains a space character, you must wrap the input with quotes “”.

### Example:

**echo 'Hey there, again!' >> greeting.txt**

- If the greeting.txt file exists, it will **append** the existing content with this new content at the end of the file. If the file does not exist, it will be created.

# HELP YOURSELF

If you are on Windows, almost all commands will come with a `--help` option. On mac, you can usually type `man <command>` and it will launch you into the manual.

- `ls --help`
- `man ls`

On windows it will simply print out all the options to the terminal and you can read them. On Mac, you launch into the manual program which is a dedicate help tool. Same idea, you can scroll through and read the documentation but when you are ready to exit, press `q` (for quit).

Another good option is to Google the command, but there might be rare cases where this isn't an option.



# A NOTE ABOUT ERRORS & OUTPUT

There are chances your command errors for any number of reasons. When this happens, there will usually be an error message printed to the terminal. **READ THIS MESSAGE CAREFULLY!**

The error message will usually tell you exactly what went wrong with the basic commands. Give it a read and see if you can remedy the situation.

There are also many commands that give **no output at all** when they work. This is just something you have to get used to.

Well that was a lot. The only way to start to get better and the command line is to use it, that is why we are introducing it so early in the course.

There is a cheat sheet in your week 1 resources on canvas that you can use as a reference for many of these commands!