While working on the computer, you need to communicate with the **Operating system** (OS) to get things done for you. For example, if you want to open a file, you have to tell the Operating system (Windows, Linux, or macOS) about it. There are two ways of interacting with the OS: one is text-based, the other is visual based. Both are important. These two methods are the command-line interface and the graphical user interface.

**What is the command line?**

The **command-line interface**or**CLI**is a way to interact with an OS via text commands. On the other hand, the **graphical user interface** or **GUI** provides an interface with many icons and menus. Here, you give commands to the operating system by clicking on these icons or menu items.

In the past, command-line interfaces were the only means of interacting with a computer. But why use it now, when you have a simple, familiar graphical interface? Well, generally command-line interfaces are much more flexible and have more options. For example, you can combine commands to create a new one, while you cannot do that via a graphical interface. Some software can even have only a command-line interface, thus requiring its user to know command line basics.

Besides, programs executable by the command-line interface can be written in a command language. They are called **shell scripts** on UNIX and UNIX-like systems such as GNU/Linux and macOS, and **batch files** on Windows.

All operating systems have command-line interfaces. Applications may have it as well. Also, modern programming languages provide an interactive command-line mode, in which you execute code line by line.

**Accessing command-line interpreter**

Usually, you don't have to go to the location of the **command-line interpreter**or**terminal** to open it. You can open it by simply searching *cmd* in Windows and *terminal* in Linux distributions.

If you feel more like an explorer and want to find the location on your own, then try the following paths:

* For Windows 10 or 8 at *Start→Windows System→Command Prompt*.
* For Windows 7, Vista or XP at *Start→All Programs→Accessories→Command Prompt*.
* For macOS at *Applications→Utilities→Other→Terminal*. Some Mac users prefer iTerm2, a replacement for Terminal, because it is slightly more user-friendly. You can find the details on the [iTerm official website](https://www.iterm2.com/downloads.html" \t "_blank) and install it on your computer.
* For Linux: it depends on your system, but usually, the CLI is located at *Applications→Accessories→Terminal* or at *Applications→System→Terminal*. If you don't find it here, just google how to access the command line on your system.

When you open it, you'll see a black (or white) window. If everything is okay, you'll see the **command prompt** where you'll be typing your command – an indicator that your computer is ready to accept commands. For Windows, the command prompt ends with >, while for Linux it's $, and for Mac OS it's $. To execute a command, type it and then press Enter.

**Learning commands**

It's time to learn some important commands. Let's open the command-line interpreter and type some commands. When you open it, you will see something similar to the below text.

C:\Users\name>

It means that you are in this directory and you can work in the CLI. Now let's try to use it.

Imagine you just woke up on the floor in a room unfamiliar to you. In fact, everything is unfamiliar to you, you don't remember anything, not even your name. There's only a computer with the open terminal and this guide, so you decide that it might be helpful to figure something out. So, now you will type your command next to this path.

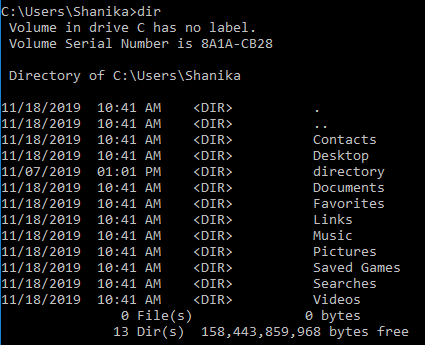
First, type whoami and press *Enter*. Unfortunately, it won't provide you with a deep and satisfying answer about who you really are, but you will see something like this:

desktop-qd7c3ju\shanika

Good, now you know your name, an imaginary one, at least. As you could already guess, the whoami command just returns the username you used on your machine. That's why you see the above output.

Next, type dir if you use Windows or ls if you use Linux or macOS and press *Enter*. Both of these commands return the list of non-hidden files and folders in your current directory. To show the list including hidden files and folders, just add an option -a, which stands for *all*. The ls command is one of the most widely used ones by developers, especially when they work on servers.

If you have Windows, you will see something similar to the image below.



There's no file that is helpful for your situation. Alas!

Isn't there a command to escape? Yes, there's one. Just type exit and you'll be out... of the command-line interpreter because this command lets you exit it. Good news, you've just learned some useful commands and completed the quest, Shanika! Now you can be free.

On the [SS64 website](https://ss64.com/), you can find a complete list of commands for [Windows](https://ss64.com/nt/), as well as for [Linux](https://ss64.com/bash/) and [macOS](https://ss64.com/osx/" \t "_blank)

**Conclusion**

The important thing to know is that excellent GUIs haven't made the CLI obsolete. It is still one of the quickest ways to get your job done. Especially if you are going to be a developer, it is very important to have good knowledge of what commands are available to you.

In this topic, we introduced the command-line interface: what it is, where it can be found, and how it can be used. We tried several commands as well: whoami, dir or ls, and exit. Later, we will learn CLI commands in detail, but now let's practice for a while!

We hope that you already know how to open the command-line interpreter and run some basic commands. Now, let's take a step further and learn how to expand the functionality of the commands and how to get more information about them.

## Commands with parameters

Sometimes, using just one command is not enough. Let's take a look at the command mkdir, which is used to create a new folder in the current directory. If you try to use it as it is, you will get an error. The terminal needs to know how to name a new folder! That's where parameters come in handy. A **parameter** is some additional information that you give to the command. Simply put, parameters are variables that commands can take.

Now, type the command mkdir with a parameter papers. We use this command to create a folder named papers:

C:\users\student> mkdir papers

Although the current directory stays the same, if you follow this path, you will see that the new folder papers was created in the student directory.

All examples in this topic are for Windows OS, but the listed commands are relevant for Linux and macOS too. Note that the path separator on Windows is a backslash, but in Linux/macOS it's a forward slash.

Now let's change our location and go to the folder you've just created! Use the cd command with the path to the papers folder as a parameter.

C:\users\student> cd C:\users\student\papers

C:\users\student\papers>

Another useful parameter of the cd command is ... It allows you to go to the **parent directory**, the directory one level above the current one.

C:\users\student\papers> cd ..

C:\users\student>

You can also go back to the **root folder**, a top-level directory in the file system. To go back to the root directory, you can use the/parameter:

C:\users\student> cd /

C:\

Thanks to commands and parameters, it seems like we are back to the roots! Actually, without parameters, most commands would be useless.

## Options

If you google anything about commands and a terminal, you'll encounter the term **options**. Don't be afraid of it! Let's briefly explore what it means.

**Options**, as the name suggests, are usually optional and are used to somehow change the common behavior of the command. If you use Windows and are already sick and tired of exploring the current drive, you can change it by adding the /d option to cd. Don't forget to set the path you want to follow as the parameter, for example, F:\Codepen snippets:

C:\users\student\Desktop> cd /d F:\Codepen snippets

F:\Codepen snippets>

Now you see that with options and parameters, you can transform a simple command into something complicated.

To sum up: what are options and parameters? Both of them are just two particular types of arguments. While an **option** changes the behavior of a command, a **parameter** is used to assign information to either a command or one of its options. One of the key differences between them is that the number of possible values in options is limited and locked in the code, while with parameters users have more freedom as they don't have such limitations.

## Help Manual

No one can remember all the existing commands, options, and parameters. Don't worry about that. The help command is there for you. Type it in Windows, and you will get a list of commands available to you.

For Linux and macOS, the way to get information about the commands depends on the shell you use. The simplest way for Linux is the --help flag. There is also the man command, short for manual. You can use the man command in Linux similar to the help command in Windows: help mkdir.

That's not all. The help command can take any command as a parameter and return all the available options. Let's try. We will use the simplest command we've learned so far, the cd command.

C:\users\student> help cd

Displays the name of or changes the current directory.

CD [/D] [drive:][path]

CD [..]

.. Specifies that you want to change to the parent directory.

Type CD drive: to display the current directory in the specified drive.

Type CD without parameters to display the current drive and directory.

Use the /D switch to change the current drive in addition to changing the current directory for a drive.

<...>

As you can see, these are all the details you need to know about the cd command. We call this description the **help manual**.

Let's discuss what the help manual includes. First, it states what the command is supposed to do. For the cd command, it reads, "Displays the name of or changes the current directory". Then it returns all the combinationsof that command along with all possible parameters that you can use. You can also notice that on Windows, commands are case-insensitive, unlike on Linux and macOS. Let's look at the example from the manual:

CD [/D] [drive:][path]

So, the above command has three parts. CD is the command name. [/D] is an option, and [drive:][path] a parameter. You might wonder what these [] brackets mean. Well, they are just notations, which means that the parameters are optional to the commands. You shouldn't add these brackets when you use commands.

You can read [this article](https://www.lifewire.com/how-to-read-command-syntax-2618082) for Windows or the manual for the [cat command](https://www.hscripts.com/tutorials/linux-commands/cat.html) on Linux/macOS to learn more about the command-line syntax and look through the examples.

## Conclusion

Let's summarize what you've learned so far:

* You can use options and parameters to extend the functionality of commands.
* You can pass different values with the parameters.
* You can get a full list of commands using the help and man commands.
* You can open a help manual for a command by typing help [command\_name] or man [command\_name]. This manual explains how to use a command properly and what options and parameters it has, if any.

Although you may feel that using these commands would slow down developers' work and that they are less efficient, we would still urge you to try them out. You have to get used to these commands as early as possible. Once you get accustomed to working with them, you will find that using them is much easier than resorting to the GUI on many occasions.

Change your directory to any other than the root directory and run it in your terminal:

# Windows

pushd \

popd

# Linux/macOS

pushd /

popd

Hint

Its using the stack data structure. Push and Pop. first in and first out  You are pushing 'pushd /' in this case root dir and changing to that dir Then popping 'pop' back the original dir that you started with.