Chapter 8 - Basic bash commands

Contents

Basic bash commands

As mentioned in chapter 1, this tutorial will be using Git Bash if you are on Windows or just any plain terminal in MacOS or Linux.

This short chapter will be explaining the very basics of bash - centering primarily around navigating and understanding the commands used throughout this guide.

As the terminal is primarily text-based, it is best to be familiar with the basic navigation commands to move about in your terminal.

This guide is by no means a comprehensive guide into all the available bash commands. It would be best if you performed your own research to understand various bash commands.

When you first open your terminal, you should be in your user directory. In Windows, this is usually `C:\Users\<user>\` and on UNIX-based machines, it is usually `/home/<user>/`. Regardless of where you start from, you can execute the bash commands described.

The file system in bash is similar to that of the traditional Windows explorer or file explorer. Instead of using your mouse to navigate your computer, you are now using your keyboard.

Note* the use of `\` and `/` as folder path delimiters are Windows/UNIX specific. For this guide, we will be using `/` as Git Bash and UNIX terminals.

pwd

`pwd` stands for "print working directory". It allows you to view the current directory that you are in.

```
pwd
```

For instance, if you run `pwd` in your starting directory, you will find the full path matching those described above.

cd

`cd` stands for "change directory". It is the cornerstone for navigating your file system. You are able to move backwards and forwards in your file system.

```
cd <file path>
```

For instance, if you wanted to move from the user directory to the `Desktop` folder, you could do so with:

```
cd Desktop
```

You can move out of a folder with `..`

```
cd ..
```

You can even chain these folder names together, using `/` as a delimiter. For instance, the commands below first move into the `Desktop` folder and then you move back one folder (to the user directory) and then into the `Documents` folder.

```
cd Desktop
cd ../Documents
```

Note* you cannot `cd` into a file. The path provided **must** be to a folder.

ls

`ls` stands for "list directory contents". It allows you to view the contents of a folder.

```
ls
```

`ls` will list out both files and folders in the current folder, which can be retrieved using `pwd`.

touch

`touch` has multiple purposes but it can be used to create new files.

```
touch <file name>
```

mkdir

`mkdir` stands for "make directory". It allows you to create new directories from the current directory.

```
mkdir <folder name>
```

echo

`echo` displays text in the terminal. It is similar to `print` in Python or `System.out.println` in Java.

```
echo "Hello world"
>>> Hello world
```

It can also be used to create text files with initial text in it. The example below creates a new file `hello-world.txt` and adds "Hello world" to it.

```
echo "Hello world" > hello-world.txt
```

It can also be used to append text into a file. The example below appends the text "This is a new line of text" to the end of `hello-world.txt`.

Note* to append text, you must use `>>`. If you are to use `>` to append text, the first line of the file will instead be replaced with the new line.

```
echo "This is a new line of text" >> hello-world.txt
```

> vs »

In the `echo` example, we used two interesting symbols - `>` and `>>`.

`>` - redirects output. In the `echo` example, we are redirecting the output of the `echo` to the file, thus creating a file with the text from `echo`.

`>>` - appends to a file. As described in the `echo` example, we are appending the output of `echo` to the file.

cat

`cat` stands for "concatenate files and print on the standard output". While that may seem like a mouthful, all it is doing is appending the contents of the input files together and printing them out.

For this guide, `cat` is used to display the content of a single file at a time, but it can be used along with other files.

```
cat <file 1> <file 2> ... <file n>
```