Contents

1 The UNIX command line interface

The UNIX **command line interface**, or **CLI**, is the most basic way to interact with the UNIX operating system. UNIX is an old OS from the 1960's where C was first developed, therefore, it is a natural environment to develop C programs in.

GNU/Linux and macOS users can access the command line by opening a terminal emulator. These is because macOS is based on UNIX and GNU/Linux is a clone of UNIX.

Windows is not based on UNIX, so you will need a program such as MSYS2 to get a UNIX-like environment.

1.1 Terminology

Here is some terminology for the command line environment:

Prompt the text that appears before your command

Terminal a user interface made completely of monospaced text

Terminal emulator a program that simulates a terminal for modern OSes

Shell the program that reads the commands you write and executes them

Command line this refers to the line where you write your command in, or, generally, the entire terminal + shell environment

Working directory the directory that a shell user is in or a program is running from

There is no universal prompt, but a common prompt on GNU/Linux may look like this, where `user` is the name of the user, `hostname` is the computer name, `~` indicates the working directory and `$` indicates that the user is on a non-admin account:

```
user@hostname ~ $
```

There are many different shells you can use. Most commonly, `bash` is the shell found on Linux distros and `zsh` is the default shell used by macOS.

This file will document things you can do in `bash`, though most commands should work the same as in `zsh`.

1.2 Command arguments

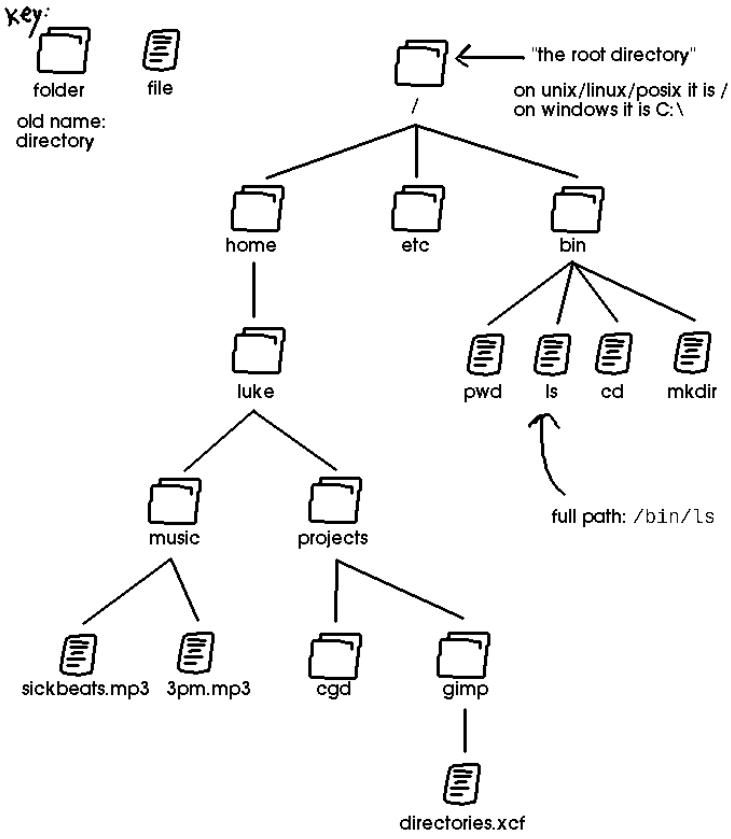
A command on the command line is composed of **arguments**, which are strings of text separated by space characters. Arguments are numbered from left to right, starting at number 0.

As an example, if I run the command `mount /dev/sdb1 /mnt/musics`, argument 0 is `mount`, argument 1 is `/dev/sdb1` and argument 2 is `/mnt/musics`.

Argument 0 is special because it is always the name of the program you are running from the shell.

1.3 Directories and basic commands (pwd, ls, cd, mkdir)

A **directory** is the same as a folder. They can store files and other directories. Here is a diagram illustrating some of the directories available on a UNIX system:



A **path** is a string of text that describes the sequence of directories to arrive to a final directory or file. On UNIX, directories and files are separated in a path's text with the `/` character.

The **root directory**, represented in text as `/`, is the directory that all other directories reside in.

When you are using the shell, the **working directory** is the directory you are currently inside of. To print it out, run the command `pwd` (print working directory) by typing it out and pressing enter.

The output should look something like this:

```
luke@goodpc ~ $ pwd
/home/luke
```

In this case, this would mean that I am inside the `home` directory in the root directory, and from there, in the `luke` directory.

You can use the `ls` command to list the files in the working directory.

The `cd` command will let you change your directory, but before we use it, try the `mkdir` command to make a directory to go into. The `mkdir` command requires an additional argument, which is the name of the directory to make. Here is an example shell session illustrating its use:

```
luke@goodpc ~ $ ls
luke@goodpc ~ $ mkdir thing
luke@goodpc ~ $ ls
thing
```

To use `cd`, run `cd dir`, where `dir` is a directory that will be searched for in the working directory, and if it exists, you will enter it:

```
luke@goodpc ~ $ ls
thing
luke@goodpc ~ $ pwd
/home/luke
luke@goodpc ~ $ cd thing
luke@goodpc ~/thing $ pwd
/home/luke/thing
```

If you run `cd` with no additional arguments, you will be taken to your user's home directory. The path to your home directory is shortened to `~` in `bash`.

If you run `cd /`, you will be taken to the root directory.

1.4 Hidden directories

Within every directory, there are always two hidden directories: `"."` and `".."`.

To show them, you can run `ls -a`. `-a` is an option or flag passed to `ls` that indicates we want to see all files and directories, including hidden ones whose names start with `"."`.

```
luke@goodpc ~/thing $ ls -a
. ..
```

`"."` will always point to the working directory, and `".."` will always point to the **parent directory** of the working directory, which is simply the directory that the working directory is inside of. `cd`'ing into `".."` is sometimes called "going up one directory". Try it out for yourself.

1.5 "Chaining together" directories

You can do what I call "chaining together" directories by separating directories with `/`. For example, if you want to move into a directory `a` and, from there, a directory `b`, you can run `cd a/b`.

Here is a shell session demonstrating this:

```
luke@goodpc /tmp/lwd $ mkdir projects
luke@goodpc /tmp/lwd $ mkdir projects/cgd
luke@goodpc /tmp/lwd $ cd projects/cgd
luke@goodpc /tmp/lwd/projects/cgd $ pwd
/tmp/lwd/projects/cgd
luke@goodpc /tmp/lwd/projects/cgd $ cd ..
luke@goodpc /tmp/lwd/projects $ pwd
/tmp/lwd/projects
luke@goodpc /tmp/lwd/projects $ cd ..
luke@goodpc /tmp/lwd $ pwd
/tmp/lwd
luke@goodpc /tmp/lwd $ cd ../..
luke@goodpc / $ cd ././.
luke@goodpc / $ cd /tmp/lwd/../lwd/../lwd/projects
luke@goodpc /tmp/lwd/projects $ pwd
/tmp/lwd/projects
```

1.6 Basic commands recap

Command	Action
pwd	print the working directory
ls	list files in the working directory
cd dir	enter the directory "dir"
cd	enter your home directory
cd /	enter the root directory
mkdir dir	make the directory "dir"