

# EEE2076 -Software Development Group Design Project

## Worksheet 1 - Working from the Command Line

P Evans

February 5, 2026

## Contents

<b>1</b>	<b>The Windows Command Line</b>	<b>1</b>
<b>2</b>	<b>The Filesystem</b>	<b>2</b>
<b>3</b>	<b>Navigating the Filesystem using the Command Line</b>	<b>3</b>
<b>4</b>	<b>Creating Directories</b>	<b>3</b>
<b>5</b>	<b>Listing the contents of a directory</b>	<b>4</b>
<b>6</b>	<b>Executing Programs</b>	<b>5</b>
<b>7</b>	<b>Moving Files and Directories</b>	<b>5</b>
<b>8</b>	<b>Printing text to the console window</b>	<b>5</b>
<b>9</b>	<b>The Path variable</b>	<b>6</b>
9.1	Environment variables . . . . .	7
9.2	Persistent Path Entries . . . . .	8
<b>10</b>	<b>Batch files</b>	<b>8</b>
<b>11</b>	<b>Further Reading</b>	<b>9</b>

## 1 The Windows Command Line

Most tasks in Windows can be accomplished using the graphical user interface components: windows, toolbars, menus, dialogs etc. This has its advantages: the user does not have to lookup the correct command or possible options to accomplish tasks, the meaning of certain actions can depend on the context in which the user is working, it is often faster to perform a sequence of clicks than to enter long and complicated command.

Nevertheless, there are scenarios where a scripts based approach is more appropriate. Mouse movements and the context in which they occur are very hard to record and reproduce. In usage cases where reproducibility is important this is a problem. In these cases it is more convenient to communicate actions by listing them in a script. Admittedly, a description like `del C:/Temp/file.txt` is more succinct and clear than saying something like: open explorer through the start menu, click My Computer, in the right pane click on the C drive, then the Temp folder, then the file file.txt.

Building and distribution software is by its very nature an activity where reproducibility is of the utmost importance. To avoid frustrating debug sessions with clients whose system might be slightly different from the development environment, it is important that the commands used to build and launch the program have the exact same meaning regardless the machine they are executed on. In particular (and as you are about to discover), building a program is an order of magnitude more complicated than deleting a file on your hard-drive.

In this worksheet, you will be introduced to the Windows command line. We will go over the most common actions such as navigating the filesystem, creating and deleting files and folders, etc. We will end with discussing how to write scripts that can help you to automate some of the more complicated tasks that will occur in the context of this project. Follow the examples on your computer

The interface through which the user communicates with the operating system by entering textual commands is referred to as the *command line* or the *shell*. There are several of these shells around, and their availability depends on the operating system. The default shell that comes with Windows is `cmd.exe` and can be opened by finding *Command Prompt* in the start menu. Alternatively, use *Win+R* to bring up the Run dialog window, enter `cmd`, and press enter.

*Note:* Recent versions of Windows come with an alternative shell, *PowerShell*. Even though this shell is more powerful and allows for interactions between system components and programs, we will stick to the classic command prompt, which has more than enough features for our purposes.

## 2 The Filesystem

As I'm sure most of you know, the files an operating system manages are organised in a tree like structure. This means that, in order to uniquely identify a file, you need to specify at each level of this tree, the branch you need to follow in order to arrive in the place where the file is stored. This sequence of branches is referred to as the *file path*. The final part of this specification is the name of the file itself. Different branches are separated by a special character (the backslash on Windows). Examples of file paths are:

```
1 C:\Temp\scratchpad .txt  
2 D:\Users\myname\Desktop\
```

Note that the first example is the path to a file, the second is a path to a directory or a folder. The first part of a path on Windows indicates the drive on which the file is stored. It comprises a single letter followed by a colon.

On Linux and MacOS, the separator between different parts of a path is the normal or forward slash, not the backslash and modern Windows is happy to accept either the forward slash or the backslash. Later we will use a program called *CMake* which runs on Windows, Linux and MacOS, CMake prefers the forward slash on all operating systems - it can start throwing errors if you use the backslash in paths!

Every program, including the command prompt has an associated *current directory* or *working directory* associated to it. To learn the current directory in `cmd`:

```
1 C:\Users\ezzpe> cd  
2 C:\Users\ezzpe
```

This seems a bit pointless since the prompt already contains this path in this case(the part before the '`>`' on the command line)! However, this is not always the case.

Files and directories can be described not only by an absolute path as demonstrated above, but also by a relative path. The relative path starts not with a slash or a drive letter, but with a filename or directory name, which is interpreted by the system as a file in the current directory or as the name of a subdirectory, respectively. If the current directory is '`C:\Users\ezzpe`', the relative paths

```
1 settings.txt  
2 somedir/cookie.dat  
3 ..\..\Temp
```

are resolved as

```
1 C:\Users\ezzpe\settings.txt  
2 C:\Users\ezzpe\somedir\cookie.dat  
3 C:/Temp
```

Here we have demonstrated the meaning and use of the ‘..’ relative directory. Regardless of the current directory, ‘..’ always refers to the directory that resided one level up (i.e. closer to the root) in the file system.

Relative paths are powerful and can provide a mechanism to change settings or defaults simply by navigating to a simple directory. Another application is expressing paths that remain valid, even if the project directory is moved or copied to a different location or to a different machine. However be careful when using relative paths in scripts, especially when you do not know where these scripts will be launched from.

### 3 Navigating the Filesystem using the Command Line

Probably the most often used command by far is `cd` (change directory). It takes a single argument: the absolute or relative path to the directory you want to become the new working directory. Say the working directory is `C:\Temp\`.

Example invocations are:

```
1 >cd Subdir  
2 >cd C:\Users\ezzpe\Desktop
```

The resulting updated current directories are

```
1 C:\Temp\Subdir\  
2 C:\Users\ezzpe\Desktop\  
3
```

On Windows, it is impossible to change to a directory on a different drive without explicitly saying you want to do this. You can do so by supplying the `/D` flag to the `cd` command:

```
1 >cd /D D:\Documents
```

Flags to command on Windows are preceded by a slash (e.g `/D`). (if you have used Linux/MacOS they are preceded by a dash e.g. `-h`).

**Open the command window and navigate using `cd` to the Desktop subdirectory under your homedir** (by default this is `C:\Users\loginname` in Windows).

### 4 Creating Directories

The command used to create directories from the command line is usually described as:

```
1 mkdir <dirname>
```

In syntax summaries such as `mkdir <dirname>`, the angle brackets denote a required argument. In usage `<dirname>` can be replace with whatever name you want for the directory. Examples of this include:

```
1 >mkdir projects  
2 >mkdir Z:\luggage
```

You should now be in C:\Users\xxxxx\Desktop. Within this directory, create a subdirectory named Temp (`mkdir`). Navigate to this directory(`cd`).

There are two directory names that have a special meaning. This meaning is relative, meaning that it depends on what the current directory is. The two special names are:

- . - a single dot is shorthand for the current directory
- .. - a single dot is shorthand for the parent directory (one level up in the filesystem tree)

Use `cd` and the appropriate special name just introduced to move from C:\Users\xxxxx\Desktop\Temp to C:\Users\xxxxx.

## 5 Listing the contents of a directory

`dir` is used to list the contents of a directory on Window and on Linux / Mac the `ls` command is used. Both `dir` and `ls` support many flags that affect the amount of information you get and how it is formatted.

```
1 Microsoft Windows [Version 10.0.17763.1935]
2 (c) 2018 Microsoft Corporation. All rights reserved.
3
4 C:\Users\ezzpe>dir
5 Volume in drive C is Windows
6 Volume Serial Number is BEEA-C413
7
8 Directory of C:\Users\ezzpe
9
10 28/07/2022 09:20 <DIR> .
11 28/07/2022 09:20 <DIR> ..
12 04/01/2021 16:20 <DIR> .cache
13 12/03/2020 18:08 <DIR> .CLion2019.3
14 05/05/2022 14:46 <DIR> .dbus-keyrings
15 20/06/2022 16:16 <DIR> .maplesoft
16 04/07/2022 11:31 <DIR> .ms-ad
17 18/05/2022 08:43 248 .qt-license
18 22/09/2020 14:31 <DIR> .TI-trace
19 17/02/2022 12:51 1,042 .viminfo
20 14/10/2021 15:38 <DIR> 3D Objects
21 01/09/2022 15:15 <DIR> Advanced Electric Machines Ltd
22 10/08/2021 13:45 <DIR> CLionProjects
23 25/08/2022 18:21 <DIR> Desktop
24 25/08/2022 17:17 <DIR> Documents
25 07/09/2022 15:35 <DIR> Downloads
26 12/03/2020 12:52 <DIR> Dropbox
27 16/09/2021 16:03 <DIR> dynamorio
28 24/10/2020 09:09 <DIR> OneDrive
29 06/09/2022 14:54 <DIR> OneDrive – The University of Nottingham
30 21/09/2020 14:36 <DIR> ti
31 29/10/2020 09:17 <DIR> workspace_v9
32 2 File(s) 1,290 bytes
32 32 Dir(s) 35,346,710,528 bytes free
34
35 C:\Users\ezzpe>
```

## 6 Executing Programs

Programs can be thought of as files that contain machine instructions instead of e.g. readable data. The command shell recognises these files by the .exe extension (Windows) or file attributes (on Unix).

On the command line, programs are executed using the filename (excluding the extension) as a command. Strictly speaking, you would need to enter the entire path of the file, but often it is sufficient to simply use the filename itself without the path indicating where on the file system it resides.

Many programs take arguments. For programs that offer a GUI the argument is often the name of another file that the user wants to open in the program. For example, to open a file hello.txt in notepad, the command reads:

```
1 C:\Users\ezzpe>notepad hello.txt
```

In this scenario, it is assumed that hello.txt resides in the current directory (C:\Users\ezzpe in this example). If the file does not exist it is created.

**Use the command line to open a file named note.txt inside C:\Users\username\Temp.**

## 7 Moving Files and Directories

The command to move files is:

```
1 move <file> <destination>
```

Examples of usage:

```
1 C:\Users\ezzpe>move file1.txt somedir  
2 C:\Users\ezzpe>move file1.txt Z:\luggage
```

Both the file you want to move and the location you want to move to can be relative or absolute paths. Note that moving a file simply means that it is referred to from a different branch of the file system. No actual data is moved on the disk. This means that moving a file is a cheap operation and that the time it takes to move a file does not depend on the size of that file. Think about it: moving a file and renaming a file are essentially the same thing. The data remains in place; you simply change how and where the data is referred in the data structure that is the filesystem.

The above remark ceases to be true if the origin and destination of a move operation are on two different hard disks. Obviously actual copying over of the data comprising the file is required in this case!

On Linux / Mac the move command is named mv.

## 8 Printing text to the console window

To print text to the terminal:

```
1 echo <message>
```

Even though this program seems rather pointless when you executed in interactive mode, the echo command is important in programs that write data to file or provide the user with progress information.

Example:

```
1 C:\Users\ezzpe>echo hello  
2 hello  
3  
4 C:\Users\ezzpe>
```

### Try the following:

- create a directory named ‘archive’
- create a file in notepad called ‘oldstuff.txt’. Fill it with some example text and save. Close notepad.
- move the file into subdirectory ‘archive’
- change the current folder to ‘archive’
- list the contents of ‘archive’

## 9 The Path variable

We’ve seen we can open notepad issuing the command ‘notepad’. There is a file somewhere on the filesystem named ‘notepad.exe’ that contains the machine instructions required to run the ‘notepad’ application. A simple dir reveals that this file does not reside in the current directory. Does this mean that each program can be run regardless where it is stored in the file system? To test this hypothesis, try running ‘wordpad’:

```
1 C:\Users\ezzpe>wordpad somefile.txt
2 wordpad is not recognized as an internal or external command, operable program or batch
   file .
3
4 C:\Users\ezzpe>
```

Clearly the answer is no. So which are the programs that can be run simply by issuing their file name as a command?

To discover where a reachable program resides on disk, you can use the Windows command ‘where’. Running e.g. ‘where notepad’ reveals the file is in ‘C:\Windows\System32\’ (Fig. ??)

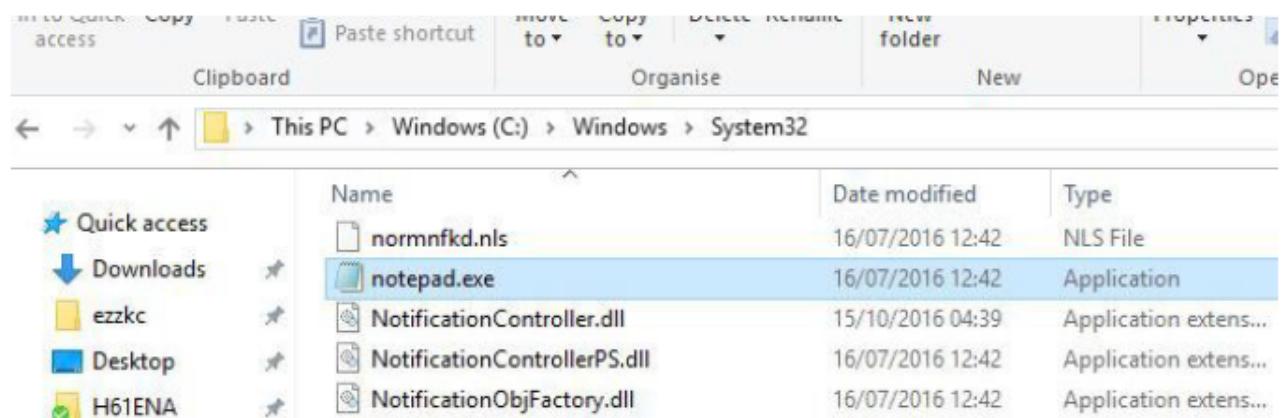


Figure 1: Location of notepad.exe

To find out where ‘wordpad.exe’ resides we cannot use ‘where’. Some googling around leads us to the following location:

Back to the problem at hand: why can we run programs in C:\Windows\System32 and not those in C:\Program Files\Windows NT\Utilities? The answer is that the command shell stores a special variable PATH that contains a list of folders that will be searched when a command is issued. The echo command can be used to print the contents of this variable:

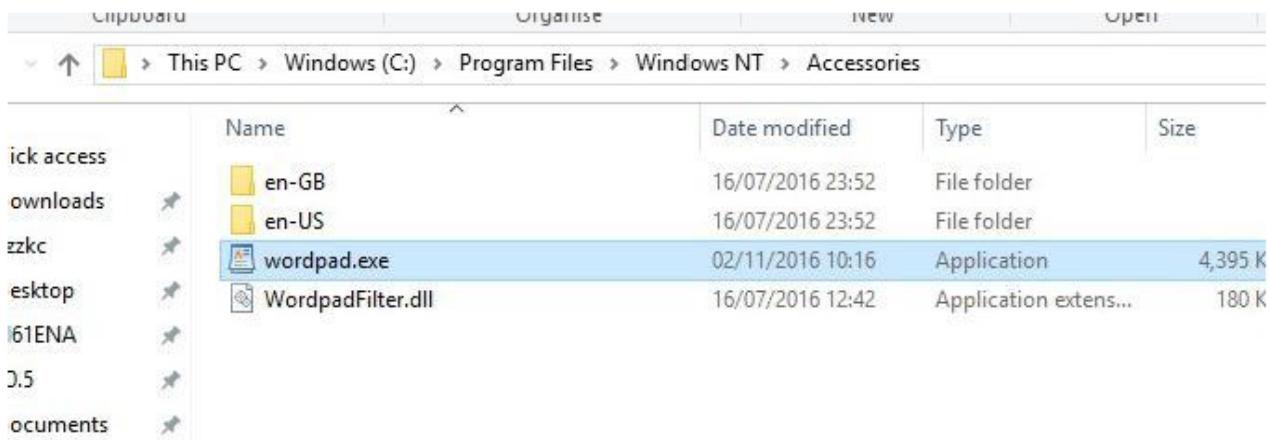


Figure 2: Location of wordpad.exe

```

1 C:\Users\ezzpe>echo %path%
2 D:\Program Files\MATLAB\R2021a\bin;D:\Program Files\MATLAB\R2021a\bin\win64;C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Git\cmd;C:\WINDOWS\System32\OpenSSH\;C:\Program Files\MiKTeX 2.9\miktex\bin\x64\;C:\Program Files\PuTTY\;C:\Program Files\Pandoc\;C:\Program Files\Microsoft SQL Server\120\Tools\Binn\;D:\Program Files\MATLAB\R2021a\runtime\win64;D:\Program Files\doxygen\bin;C:\Program Files\CMake\bin;C:\Users\ezzpe\AppData\Local\Microsoft\WindowsApps;C:\Program Files (x86)\Dr. Memory\bin\;C:\Program Files (x86)\VTK\bin;D:\Qt\6.2.4\msvc2019_64\bin;C:\Program Files\PGI\win64\19.10\bin;D:\msys64\mingw64\bin1;C:\Program Files (x86)\NSIS\Bin;C:\Users\ezzpe\AppData\Local\Microsoft\WindowsApps;C:\Program Files (x86)\Dr. Memory\bin\;C:\Program Files (x86)\VTK\bin;D:\Qt\6.2.4\msvc2019_64\bin;C:\Program Files\PGI\win64\19.10\bin;D:\msys64\mingw64\bin1;C:\Program Files (x86)\NSIS\Bin;D:\OpenVR\bin\win64;

```

This is a list of directories the system will search for file names of executables that correspond to a command you issued. To print the contents of a variable such as PATH using echo, the variable needs to be enclosed by percentage signs. If you forget to do this, you will simply print the word PATH, not the content of the variable.

Like most modern OSs, Windows support file paths containing spaces but when specifying paths that contain spaces on the command line, you may need to enclose the entire path with double quotes ("").

## 9.1 Environment variables

- PATH is just one of many so called environment variables
- Every program runs in a so called environment
- The values of variables in the environment affect which program is run
- They can also affect various options and settings for programs

We can solve the wordpad launching problem by either specifying the full path to wordpad.exe every time you run it, or by adding the location of 'wordpad.exe' to the path environment variable.

Any environment variable can be set using... `set`:

```
1 C:\Users\ezzpe\somedir>set MY_VARIABLE=hello
2 C:\Users\ezzpe\somedir>echo %MY_VARIABLE%
3 hello
```

To add something to the end of an existing variable we can use the following syntax:

```
1 C:\Users\ezzpe\somedir>set MY_VARIABLE=%MY_VARIABLE%; goodbye
2 C:\Users\ezzpe\somedir>echo %MY_VARIABLE%
3 hello; goodbye
```

It is very easy to forget about this and accidentally replace the ‘PATH’ with the single new path you wanted to add. If this happens the easiest solution is to close the command shell you are working in and open a new one. So let’s add the path to ‘wordpad.exe’ to the list of paths in ‘PATH’:

```
1 C:\Users\ezzpe\somedir>set PATH=%PATH%; C:\Program Files\Windows NT\Windows NT\
   Accessories
2 C:\Users\ezzpe\somedir>wordpad . exe
```

## 9.2 Persistent Path Entries

The annoying thing about the path as we have been using and modifying it is that after you close the command window you were working in, all modifications are forgotten. To fix this, we need to automate adding the entries every time we log into the system. On Windows path entries can be added as shown in Fig. 3. You can find the relevant dialog by searching for *path* in the Windows search bar.

## 10 Batch files

Commands such as `cd`, `mkdir`, `move`, `echo`, `set` can be used to create scripts in batch files. These are text files with a .bat extension. Each line of the file typically contains one command. An example batch file is shown below, put this content in a file called *test.bat*. The `@echo off` on the first line prevents the batch file from displaying all output from its commands, only output from the `echo` command will be shown. The `rem` command is a remark - i.e. a comment and the `>` operator feeds output of one command (`echo`) into the file *new\_file.txt*.

```
1 @echo off
2
3 rem create a directory
4 mkdir new_dir
5
6 rem cd into directory
7 cd new_dir
8
9 rem create a new file and put some the word hello in it
10 echo hello > new_file.txt
11
12 rem open the new file with notepad
13 notepad new_file.txt
14
15 rem Inform that script has finished
16 echo Batch file finished executing.
```

Running this gives the following output:

```
1 C:\Users\ezzpe>test
2 Batch file finished executing.
```

code.

This could happen for example when new or modified silent assumptions that other code relies upon. This is how bugs can be created without actually writing more, and much more mundane reasons.

All of this makes creating software a very better solution than going back in time. Even though most modern compilers and highly automated build systems

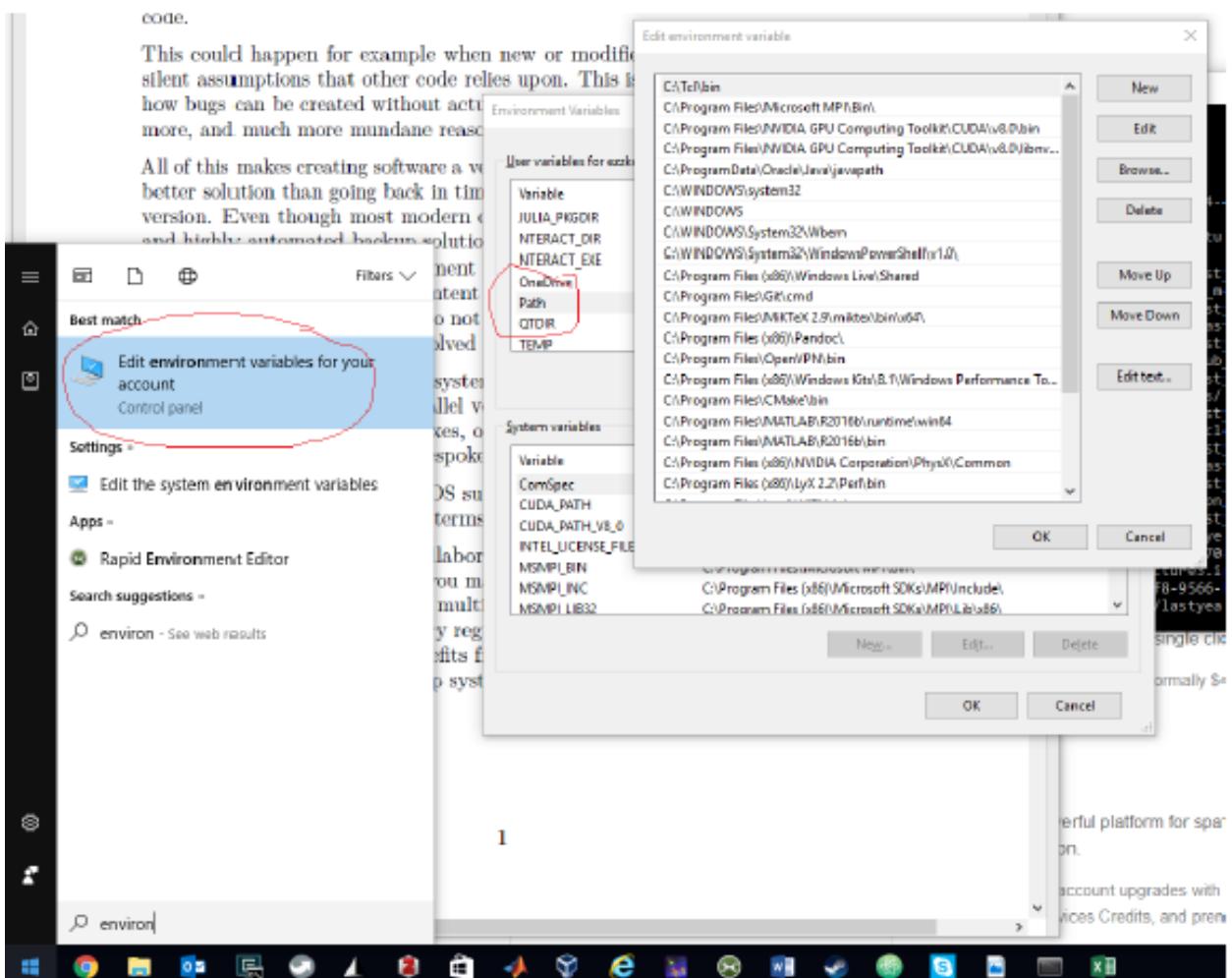


Figure 3: Location of wordpad.exe

```
3
4 C:\Users\ezzpe\new_dir>
```

## 11 Further Reading

- 1 Windows Command Prompt in 15 Minutes
- 2 Jansen, guide to Windows batch scripting