H62PEP Computing - Worksheet 1

P. Evans, K. Cools

The Windows Command Line

Most tasks in Windows can be accomplished by manipulation of the various user interface components that are rendered to the screen. This certainly 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 commands,…

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\scratchpad.txt` is more succint 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 scratchpad.txt, … Another reason is that even though the main options and most often used options to actions are usually represented through buttons, checkboxes, context menus, … the screen real estate is much more limited than the number of possible text commands and arguments the user can enter through the keyboard.

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

This week, 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 folder, … 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.

The interface through which the user communicates with the operating system by entering textual command 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 programmes, we will stick to the classic command prompt, which has more than enough features for our purposes.

Note: In this project you will be asked to design a portable piece of software. This means that you will have to support in addition to Windows either one of Linux or MacOS. These operating systems come with the **Born Again Shell** or bash for short. The commands for this shell are slightly different, but the general ideas are the same. We will explain the most important differences when they come up. To fill in the details you are referred to guides and tutorials you can find online.

The Filesystem

As I'm sure most of you know, the files an operating system manages are organised in a tree like structrure. 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:

```
C:\Temp\scratchpad.txt
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.

Note: On Linux and MacOS, the sperator between different parts of a path is the normal slash, *not* the backslash. Windows is happy to accept either the slash or the backslash. In the context of portable projects and web based applications, prefer to consistently use the slash! In other words, write the two examples above as:

```
C:/Temp/scratchpad.txt
D:/Users/myname/Desktop
```

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

```
C:\Users\ezzkc>cd
C:\Users\ezzkc
```

Note: This seems a bit pointless since the prompt already contains this path! However, this is not always the case, for example in bash on Linux and MacOS. On these systems you can *Print the Working Directory*:

```
myname@mypc MINGW64 ~
$ pwd
/home/myname
```

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/myname`, the relative paths

```
settings.txt
somedir/cookie.dat
../../Temp
```

are resolved as

```
C:/Users/myname/settings.txt
C:/Users/myname/somedir/cookie.dat
C:/Temp
```

Here we have demonstrated the meaning and use of the `..` relative directory. Regardless 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.

Navigating the File System

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:

```
>cd Subdir
>cd C:/Users/myname/Desktop
```

The resulting updated current directories are

```
C:/Temp/Subdir/
C:/Users/myname/Desktop/
```

Windows: By default it is impossible to change to a directory on a different drive. You can do so by supplying the /D flag to the cd command:

```
>cd /D D:\WhereIStoreMyMovies
```

Flags to command on Windows are preceded by a slash (e.g /D). On Unix they are preceded by a dash (e.g. -h). On Unix, multiple options can be passed after a single dash (e.g. –rf). Many options have a longer form that can be used for clarity; these long forms need to be preceded by a double dash (e.g. –help).

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

Creating Directories

To create a directory or folder: * `mkdir <dirname>`

Examples:

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

Note: 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, **omitting the angle brackets**!

Exercise: Last exercise left you in `C:\Users\xxxxx\Desktop`. Within this directory, create a subdirectory named Temp. Navigate to this directory.

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 double dot is shorthand for the directory one level up in the tree stucture that is the filesystem.

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

Listing the contents of a directory

- `dir`: list the contents of a directory *Examples*

```
C:\Users\myname>dir
 Volume in drive C is Windows
 Volume Serial Number is 2229-5747

 Directory of C:\Users\ezzkc

01/02/2017  16:11    <DIR>          .
01/02/2017  16:11    <DIR>          ..
02/01/2017  16:09    <DIR>          .android
02/01/2017  15:59    <DIR>          .AndroidStudio2.2
02/12/2016  11:44    <DIR>          .atom
02/12/2016  11:04    <DIR>          .conda
23/01/2017  19:19    <DIR>          .config
06/12/2016  18:20               210 .gitconfig
30/11/2016  14:33    <DIR>          .ipython
30/11/2016  17:06    <DIR>          .julia
30/01/2017  13:20            83,494 .julia_history
23/01/2017  15:30    <DIR>          .jupyter
13/01/2017  15:27    <DIR>          .matplotlib
```

```
13/12/2016  13:12    <DIR>          .ngrok2
06/12/2016  15:33    <DIR>          .ssh
06/12/2016  18:20             1,390 .viminfo
20/12/2016  10:56    <DIR>          Application Data
24/12/2016  11:19    <DIR>          Roaming
01/12/2016  11:42    <DIR>          Tracing
               5 File(s)         85,311 bytes
              24 Dir(s)  354,761,969,664 bytes free
```

Linux / Mac: The `ls` command is used to list the contents of a directory.

Both dir on and ls support many flags that affect the amount of information you get and how it is formatted.

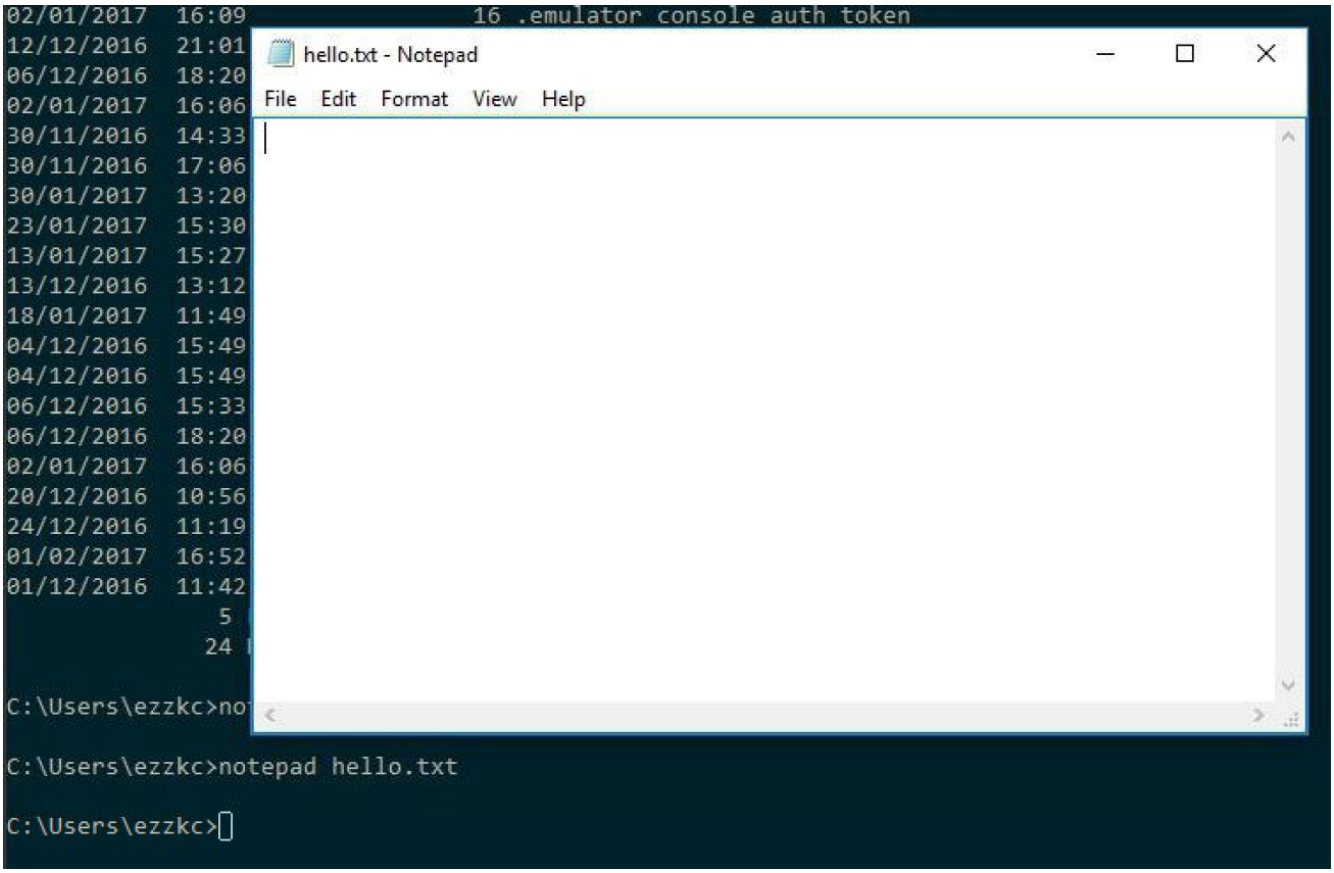
Executing Programmes

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

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

```
C:\Users\myname>notepad hello.txt
```



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

Exercise: Use the command line to open a file named first_note.txt inside `C:\Users\ezzkc\Desktop\Temp`. Don't worry: if the file does not already exist, notepad will offer to create it for you.

Moving Files and Directories

- `move`: you guessed it! Move files around.

```
C:\Users\myname>move file1.txt somedir
C:\Users\myname>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.

Note: 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!

Linux / Mac: The move command is simply named `mv`.

Printing text to o the console window

To print text to the terminal: * `echo <msg>` Even though this programme seems rather pointless when you executed in interactive mode, the echo command is important in programmes that write data to file or provide the user with progress information.

Example:

```
C:\Users\ezzkc>echo hello
hello

C:\Users\ezzkc>
```

Exercise

- 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`

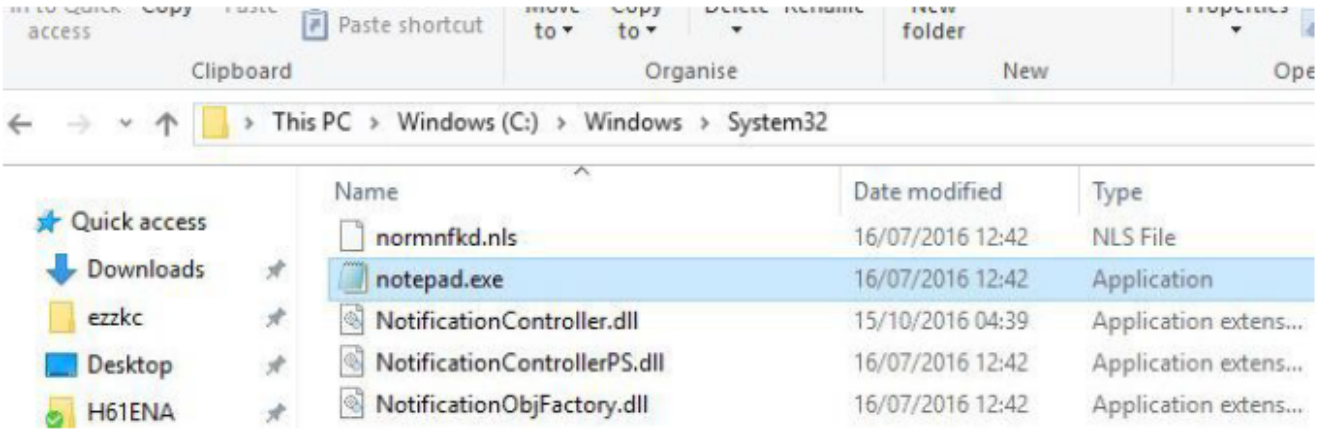
The Path variable

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 programme can be run regardless where it is stored in the file system? To test this hypothesis, try running `wordpad`:

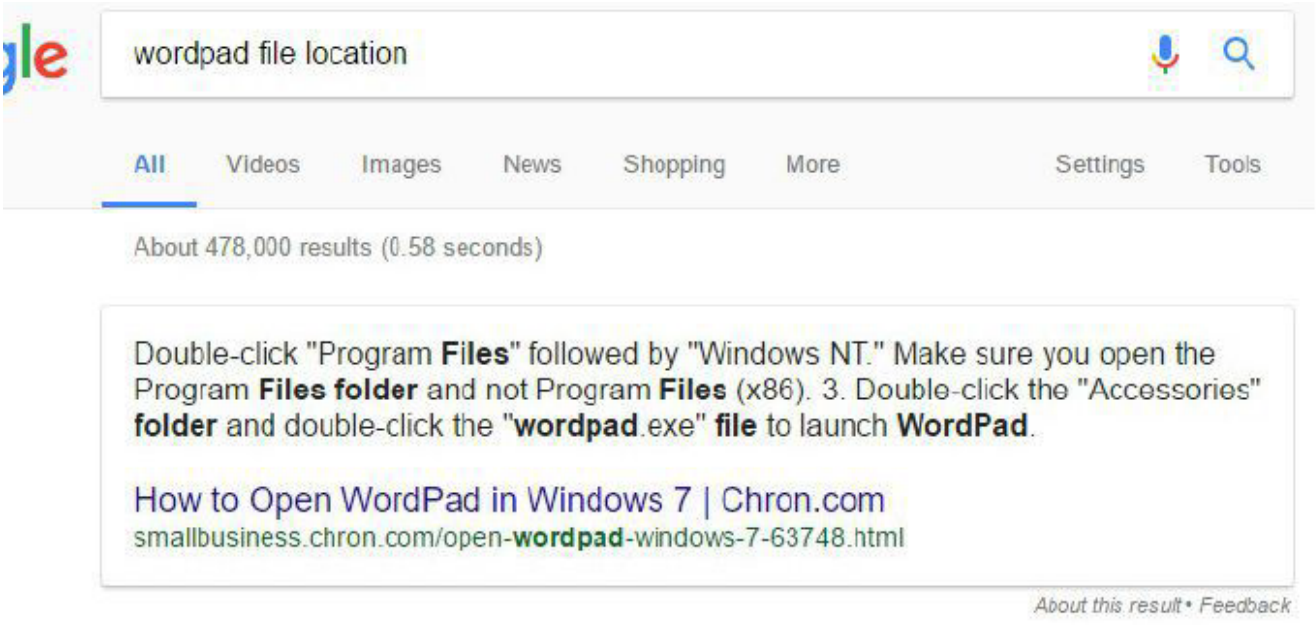
```
C:\Users\ezzkc>wordpad somefile.txt
wordpad is not recognized as an internal or external command, operable program or batch file.

C:\Users\ezzkc>
```

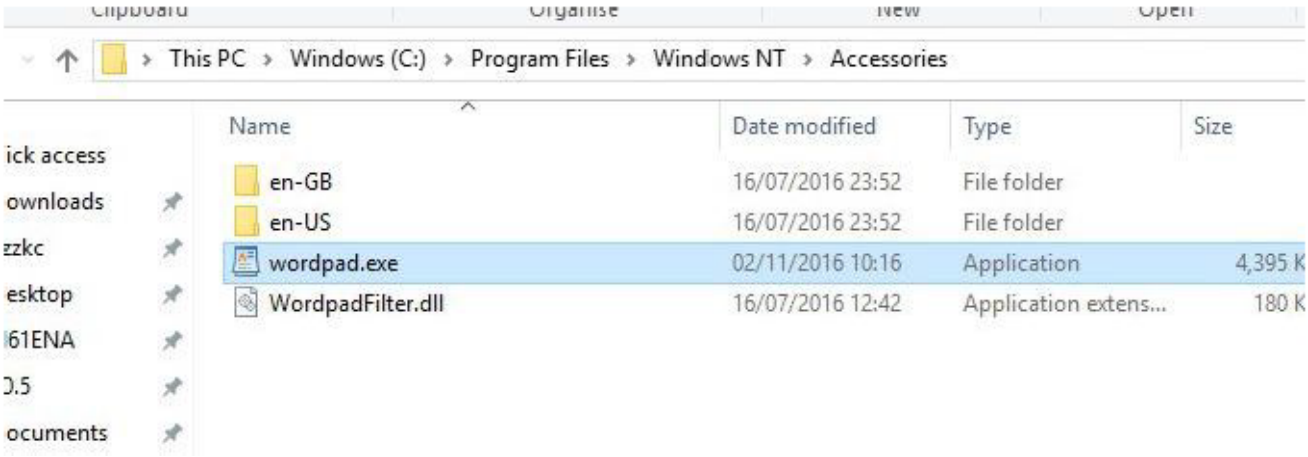
Clearly the answer is no. So which are the programmes that can be run simply by issuing their file name as a command?

To discover where a reachable programme resides on disk, you can use the Windows command `where`. Running e.g. `where notepad` reveals the file is in `C:/Windows/System32/`.



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 programmes 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:

```
C:\Users\ezzkc>echo %PATH%
C:\Program Files\Microsoft MPI\Bin\;C:\Program Files\NVIDIA GPU Computing Toolkit\
CUDA\v8.0\bin;C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\libnvvp;C:\
ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\
Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Live\
Shared;C:\Program Files\Git\cmd;C:\Program Files\Anaconda3;C:\Program Files\Anaconda3\
Scripts;C:\Program Files\Anaconda3\Library\bin;C:\Program Files\MiKTeX 2.9\miktex\
bin\x64\;C:\Program Files (x86)\Pandoc\;C:\Program Files\OpenVPN\bin;C:\Program Files (x86)
Windows Kits\8.1\Windows Performance Toolkit\;C:\Program Files\CMake\bin;C:\Program Files\
MATLAB\R2016b\runtime\win64;C:\Program Files\MATLAB\R2016b\bin;C:\Program Files (x86)\
NVIDIA Corporation\PhysX\Common;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\
Common Files\Intel\WirelessCommon\;C:\Program Files (x86)\LyX 2.2\Perl\bin;C:\Users\
ezzkc\AppData\Local\Microsoft\WindowsApps;C:\Users\ezzkc\AppData\Local\atom\bin;D:\
Users\ezzkc\Apps\ngrok-stable-windows-amd64;C:\Users\ezzkc\AppData\Local\Julia-0.5.0\
bin;D:\Users\ezzkc\Apps\gmsh-2.14.1-Windows;C:\Program Files\gs\gs9.20\bin;C:\
Program Files\Intel\WiFi\bin\;C:\Local\VTK\bin;C:\Qt\5.7\msvc2015_64\bin;C:\Program Files\
Java\jdk1.8.0_111\bin
```

This is a list of directories the system will search for file names of executables that correspond to a command you issued. If you look really hard you can find C:/Windows/System32 in this mess!

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.

Linux / Mac: The separator between paths on Windows is the semi-colon. On Unix it is a colon. To print the path, you should use the command `echo $PATH` (precede the variable name with a dollar sign).

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

Environment variables

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

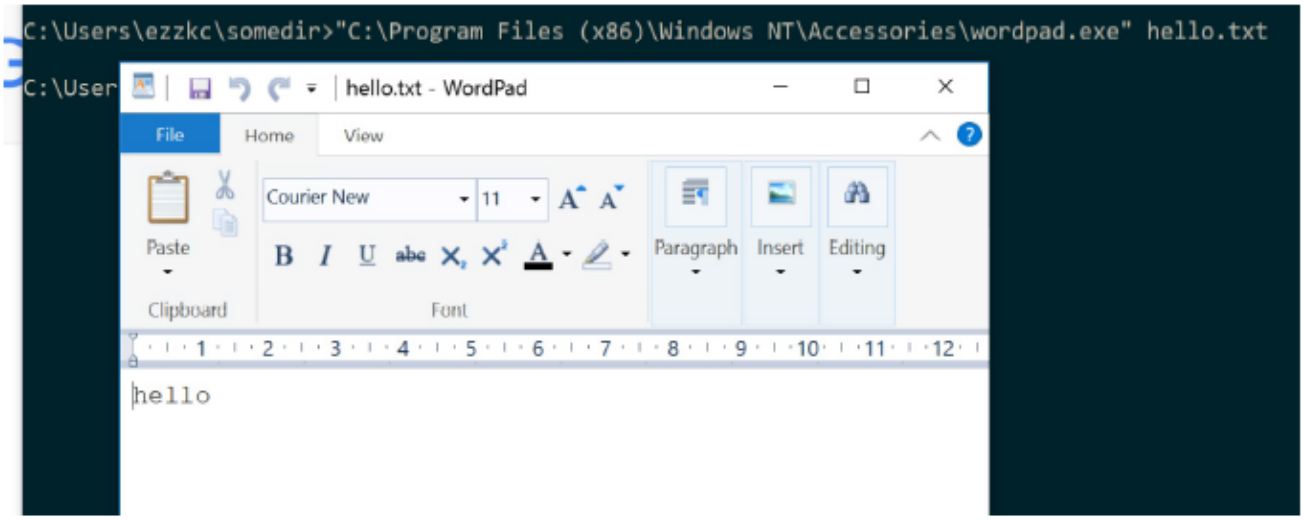
Exercise: Retrieve a list of all environment variables by issuing the set command without any arguments. Can you find the variables responsible for defining the Z-drive to be your home directory?

Linux / Mac: On bash, a list of environment variables can be retrieved by issuing the `printenv` command

Launching wordpad from the command line

- We can solve or problem by either:
  - specifying the full path to `wordpad.exe`
  - adding the location of `wordpad.exe` to the path variable

Solution 1:



Launching wordpad from the command line

Solution2:

- An environment variable can be set using… `set`

```
C:\Users\ezzkc\somedir>set BEST_MODULE_EVER=H62PEP
C:\Users\ezzkc\somedir>echo %BEST_MODULE_EVER%
H62PEP
```

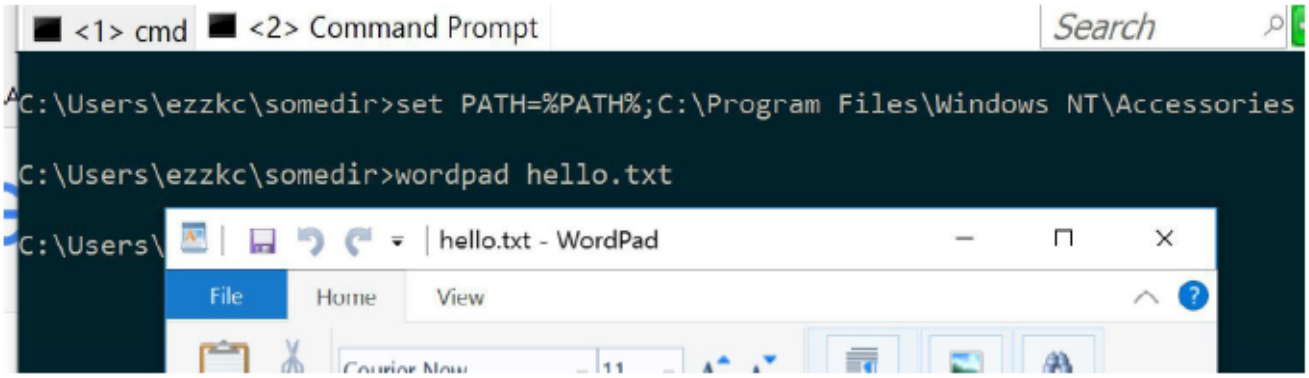
To add something to an existing variable we can use the following syntax:

```
C:\Users\ezzkc\somedir>set BEST_MODULE_EVER=%BEST_MODULE_EVER%:computing
C:\Users\ezzkc\somedir>echo %BEST_MODULE_EVER%
H62PEP:Computing
```

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`:

Launching wordpad from the command line

- We want to **add** the location of `wordpad` to the list stored in `PATH` (see figure)



Why `PATH` is evil

At some point you will see a weird bug appear that can be traced back to the path variable: Mark my words!

Say we deployed a virus called `notepad.exe` in `C:\Users\myname\somedir`

```
C:\Users\ezzkc>set PATH=C:\Users\ezzkc\somedir;%PATH%
C:\Users\ezzkc>notepad
portal to hell opened!
C:\Users\ezzkc>
```

We just hijacked `notepad` because the `PATH` is searched from left to right. As a result when issuing the command notepad it isn't the real `notepad.exe` that is launched but instead the version we injected in the system. This scenario is not as unlikely as it seems.

Windows: on Windows the PATH has a double role. In addition to providing the set of paths that is search for exe files it also provides the set of paths that is searched for shared or dynamic libraries (dlls). You will learn about dll files next project week. In essence they contain parts of the code of your programme that you want to keep separate because it is shared by multiple programmes, it needs updating independently, or it might even originate from a different developer. The programme will dynamically link to the first shared library it find on the PATH. There is no guarantee that the first version encountered has the correct version, is compiled for the correct platform, or was produced with a compatible compiler. This caused a lot of headache in the early days of the Windows OS and originated the expression dll hell.

MacOS and Linux do not suffer from this because: the path variable is set more conservatively, library versions are part of the filename, these platforms have a clear preferred compiler, and executables and shared libraries are stored in different locations.

Exercise (Windows)

- Download the `atom` editor: https://atom.io/
- Install, find out where `atom.exe` lives
- modify the path so you can edit files by simply typing `atom filename.txt`
- create in your user directory a subdir `secretlocation`
- Inside that dir, create using notepad a file `atom.bat` containing

```
@echo I just hijacked atom.exe
@echo Corrupted_data > %USERPROFILE%\exterminate.txt
@echo ..and injected a virus in your homedir!
```

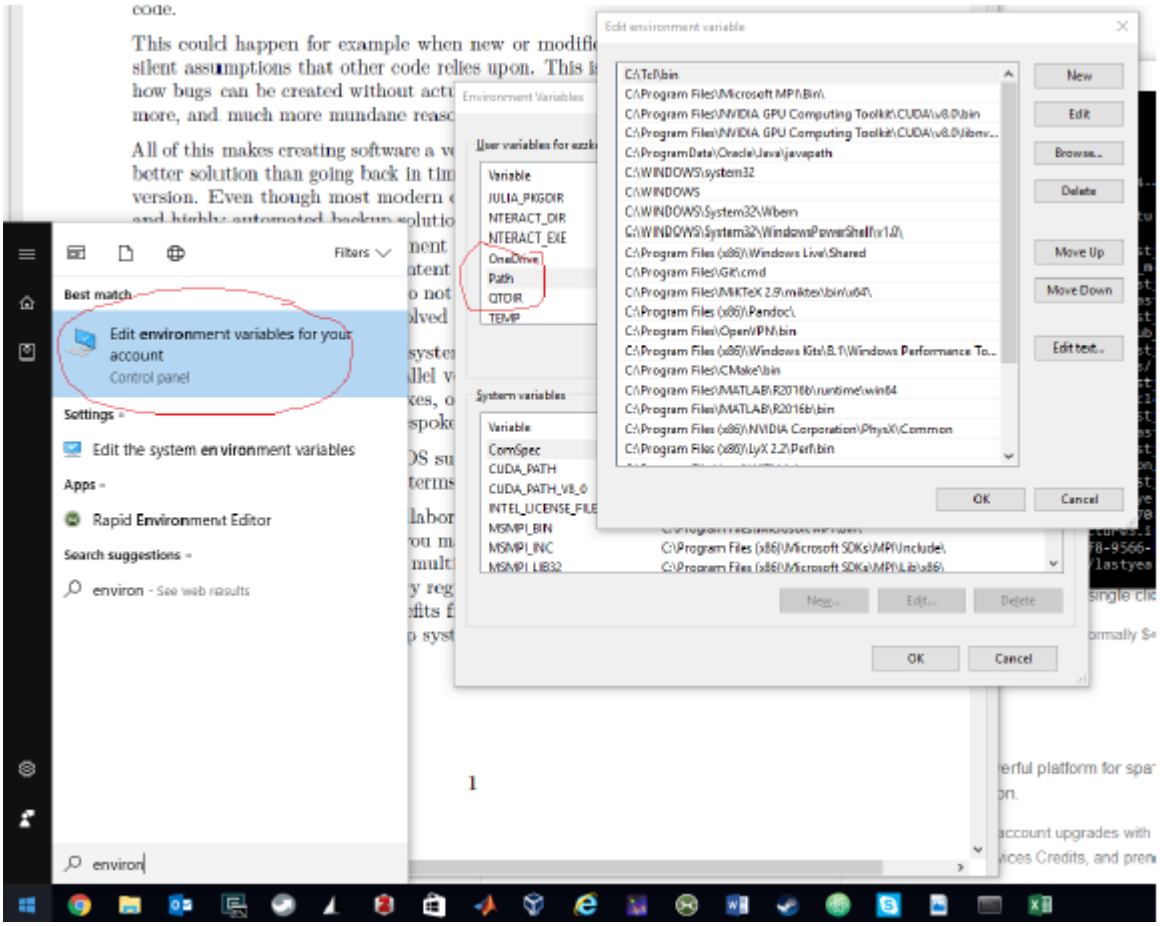
- modify the path so `atom.bat` **shadow** shadows the real `atom.exe`
- verify that a file named `exterminate.txt` was created in your homedir

Exercise (Other OSs)

- Make sure you have a programming text editor available on your computer that you know how to launch and use.
- Make sure you know how write a short shell script (equivalent of a batch file) that can:
  - Set / modify environment variables such as the path
  - print messages to the console and into text files (as the Windows batch file above does)
  - execute commands

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 the figure. Note that it is possible you need to log out and in again for this changes to kick in.



Linux: When working in bash under Linux you need to edit a file called .bashrc. This file contains a list of bash commands that are executed every time a bash window is opened. To add an entry to the path, add at the bottom of .bashrc a line like:

‘`export PATH=$PATH:/home/myname/Apps/bin/`

The order in which entries appear in the path is important! You might not see .bashrc in you Linux file explorer. This is because it is Linux convention that files starting with a dot are hidden and not displayed by default. On Windows files are hidden by setting a certain attribute.

Batch files

The file `atom.bat` that you created in the previous exercise does not contain a series of machine instructions (like an `exe` file) but rather a listing of command line instructions. These files are called scripts or batch files and are treated as programmes by the command shell, just like `.exe` files.

Further Reading

- Windows Command Prompt in 15 Minutes - http://www.cs.princeton.edu/courses/archive/spr05/cos126/cmd-prompt.html
- Jansen, guide to Windows batch scripting - http://steve-jansen.github.io/guides/windows-batch-scripting/index.html