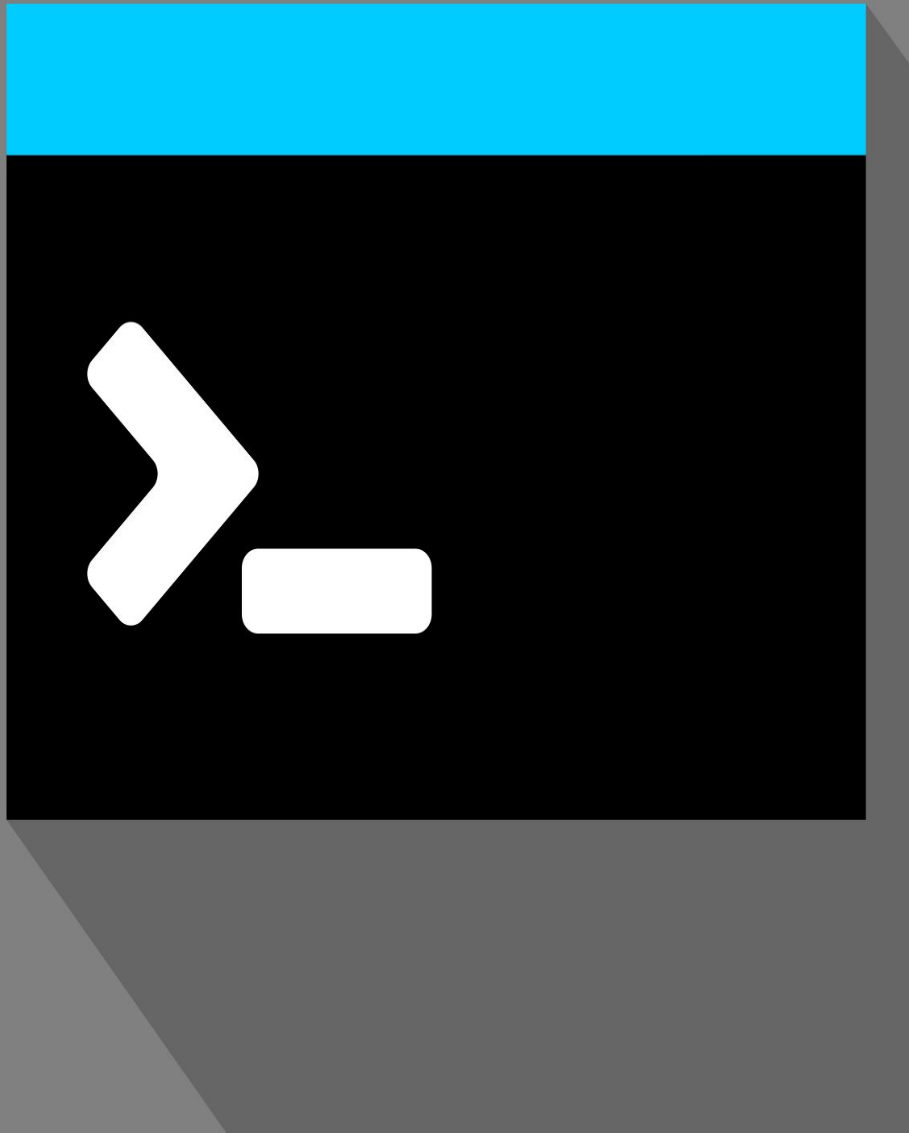
COP3503

Working With a Command Line Interface (CLI)

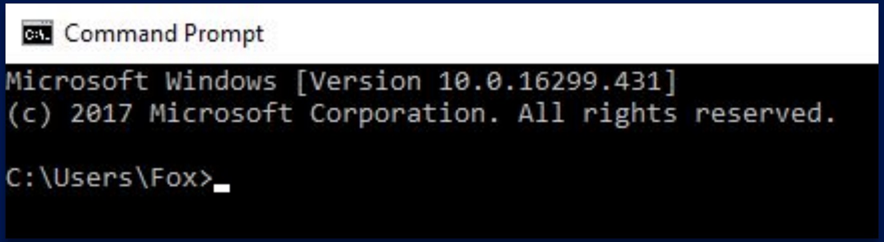
Command Line Interface (CLI)

- Text-based method of communicating with a program or your Operating System (OS)
  - You type the commands you want to execute.
- For programs (or users) that don't require a graphic user interface (GUI)
  - Typing a command is closer to "speaking computer" than clicking images on the screen.
  - Command-line interfaces were first; GUIs came later.
- Execute core operations of the OS (creating, deleting, moving files, etc)
- Execute non-OS programs (the things you write)

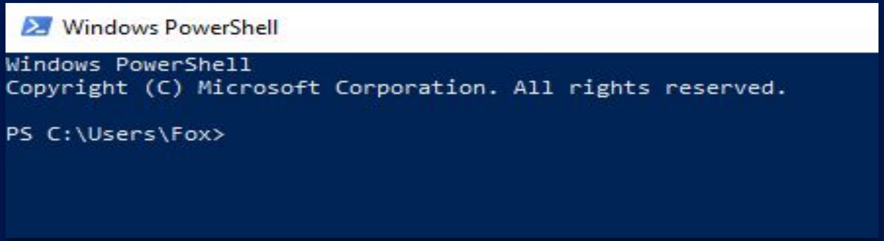


Command-Line Environments

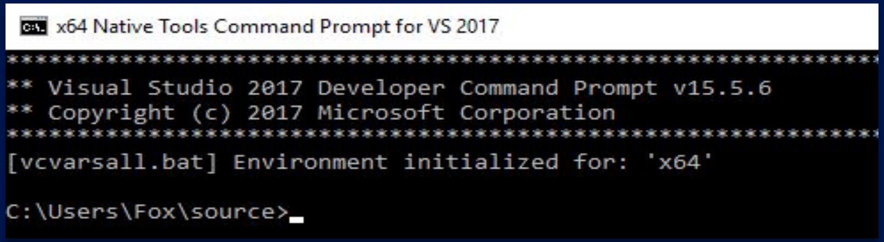
- Your operating system will have (at least) one.
- Various programs may also install their own version.
  - Command Prompt
  - PowerShell
  - Git CMD (if you have Git installed)
  - Terminal, Bash, etc…
- They all serve the same purpose:
  Let you communicate with the operating system.


Command Prompt

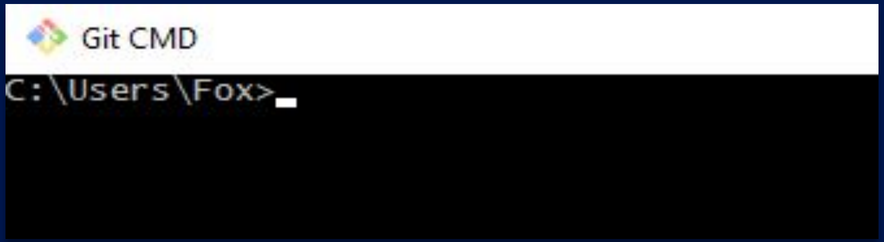
```
Microsoft Windows [Version 10.0.16299.431]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Fox>_
```


Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Fox>
```


x64 Native Tools Command Prompt for VS 2017

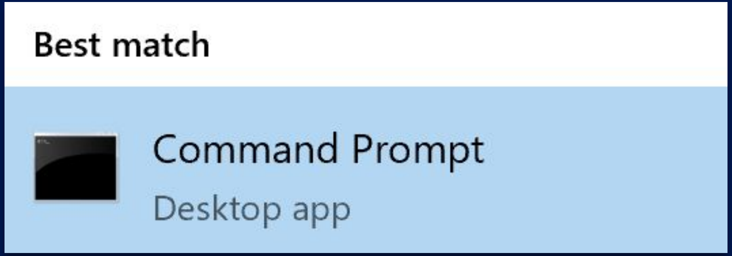
```
**********************************************************************
** Visual Studio 2017 Developer Command Prompt v15.5.6
** Copyright (c) 2017 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x64'

C:\Users\Fox\source>_
```

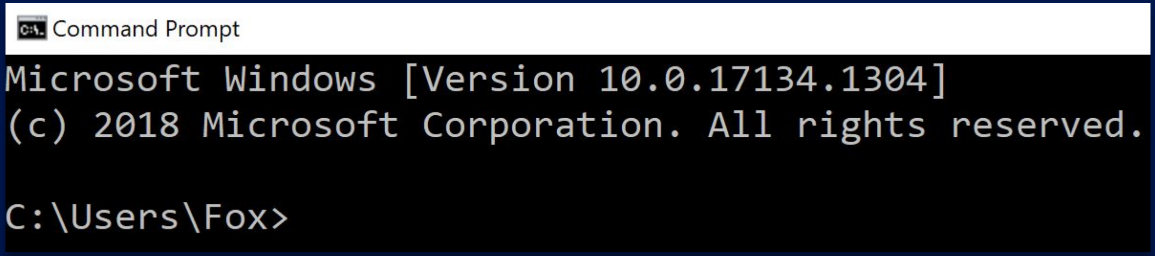

Git CMD

```
C:\Users\Fox>_
```

Command Line Basics

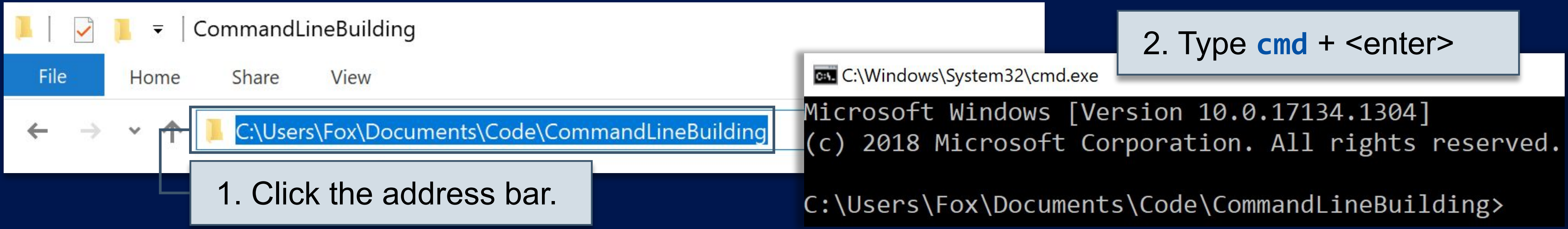
- In Windows, from the Start menu you can search for “command” or “cmd” to open the Command Prompt.





```
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Fox>
```

- MacOS will be called Terminal, on another OS it may be called something else.


CommandLineBuilding
File
Home
Share
View
C:\Users\Fox\Documents\Code\CommandLineBuilding
C:\Windows\System32\cmd.exe

2. Type `cmd` + <enter>

1. Click the address bar.

```
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Fox\Documents\Code\CommandLineBuilding>
```

- Either way, a default folder is opened, and the system waits for you.

Basic Command Line Commands

- **cd** Change directory. Everyone should know this.

```
> cd c:/
> cd c:/Users/Fox/Documents
> cd c:/Users/Fox
> cd Documents
> cd..
```

Change to a specific directory anywhere on a drive

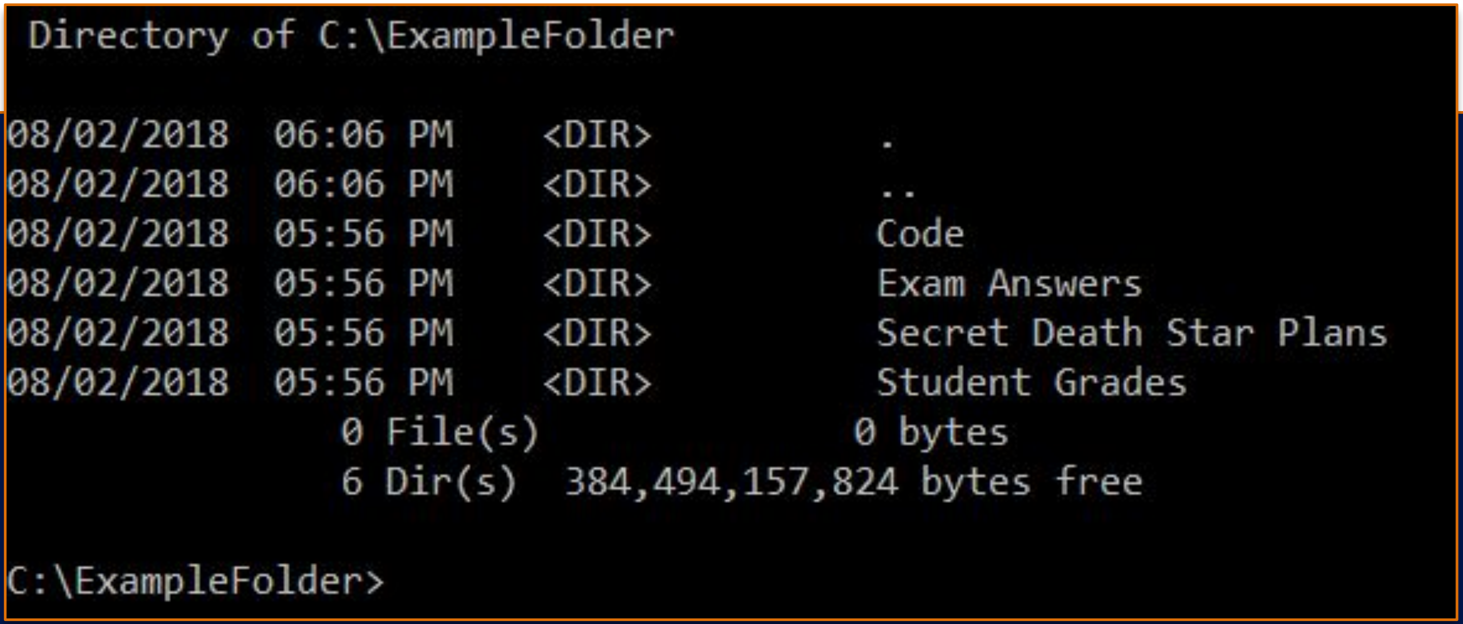
Change to a **local directory** within the current one (users\Fox in this case)

Move **up** one directory
From "users/Fox/Documents", back to just "users/Fox"

Basic Command Line Commands

- `dir` Show the contents of the current directory (on Windows)
- `ls` Same thing, but for Unix-based systems

```
> cd c:/ExampleFolder
> dir
```



```
 Directory of C:\ExampleFolder

08/02/2018  06:06 PM    <DIR>          .
08/02/2018  06:06 PM    <DIR>          ..
08/02/2018  05:56 PM    <DIR>          Code
08/02/2018  05:56 PM    <DIR>          Exam Answers
08/02/2018  05:56 PM    <DIR>          Secret Death Star Plans
08/02/2018  05:56 PM    <DIR>          Student Grades
               0 File(s)              0 bytes
               6 Dir(s)  384,494,157,824 bytes free

C:\ExampleFolder>
```

Basic Command Line Commands

`mkdir`	Create a directory
`rmdir`	Remove a directory
`copy x, y`	Copy file **x** to directory **y**
`del`	Delete a file
`exit`	Exit the command prompt
`cls/clear`	Clear the CLI window

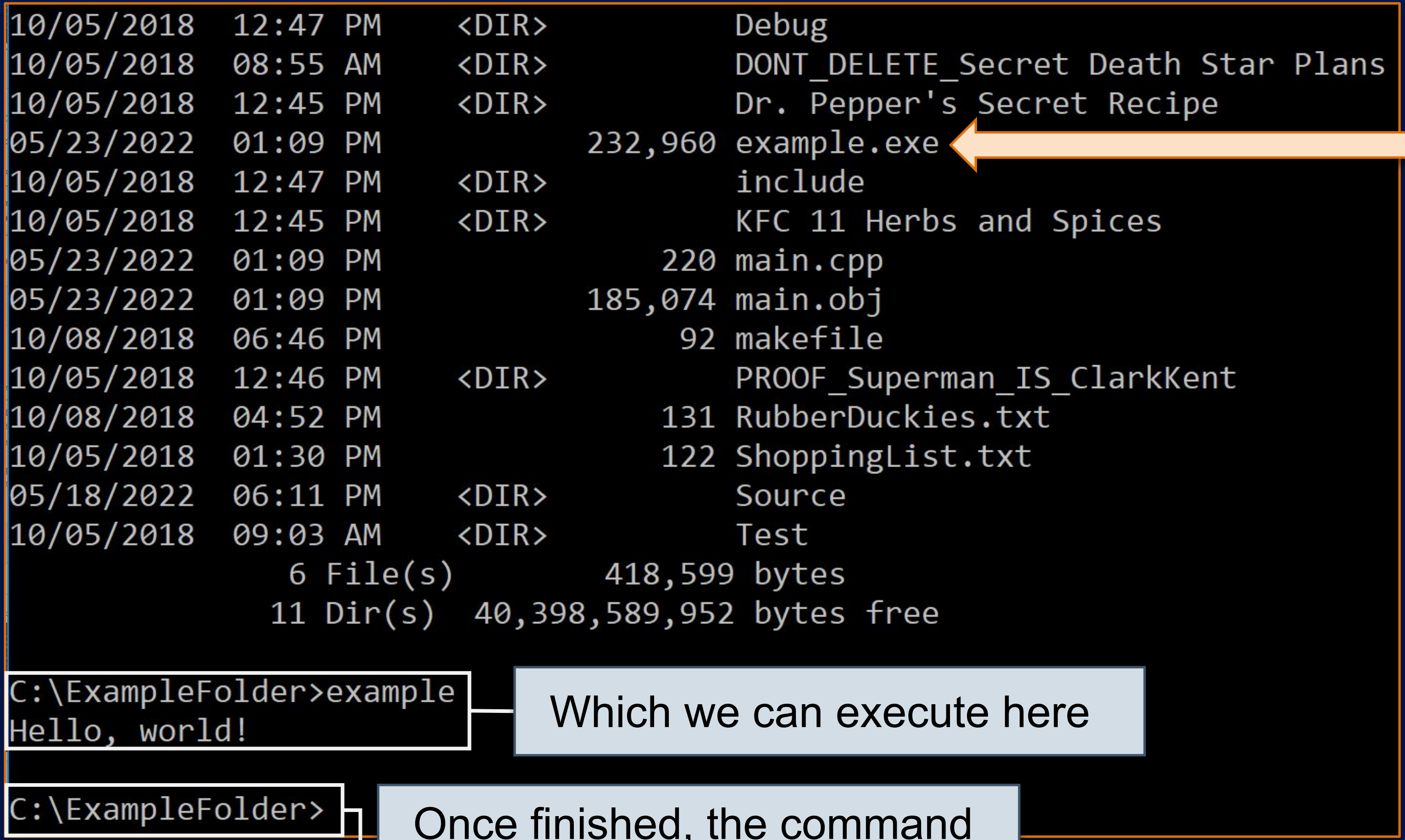
- And so on… every OS has a list of commands, many unique to that OS.
- Everyone should know `cd` and `dir/ls` – basic navigation.

You don’t have to be a master of the CLI and memorize all this stuff.

Being aware of it, and knowing how to use it on a basic level can be very valuable.

Running Programs from CLI

- Just type the name of the executable



```
10/05/2018  12:47 PM    <DIR>          Debug
10/05/2018  08:55 AM    <DIR>          DONT_DELETE_Secret Death Star Plans
10/05/2018  12:45 PM    <DIR>          Dr. Pepper's Secret Recipe
05/23/2022  01:09 PM           232,960 example.exe
10/05/2018  12:47 PM    <DIR>          include
10/05/2018  12:45 PM    <DIR>          KFC 11 Herbs and Spices
05/23/2022  01:09 PM               220 main.cpp
05/23/2022  01:09 PM           185,074 main.obj
10/08/2018  06:46 PM                92 makefile
10/05/2018  12:46 PM    <DIR>          PROOF_Superman_IS_ClarkKent
10/08/2018  04:52 PM               131 RubberDuckies.txt
10/05/2018  01:30 PM               122 ShoppingList.txt
05/18/2022  06:11 PM    <DIR>          Source
10/05/2018  09:03 AM    <DIR>          Test
               6 File(s)        418,599 bytes
              11 Dir(s)  40,398,589,952 bytes free

C:\ExampleFolder>example
Hello, world!

C:\ExampleFolder>
```

Which we can execute here

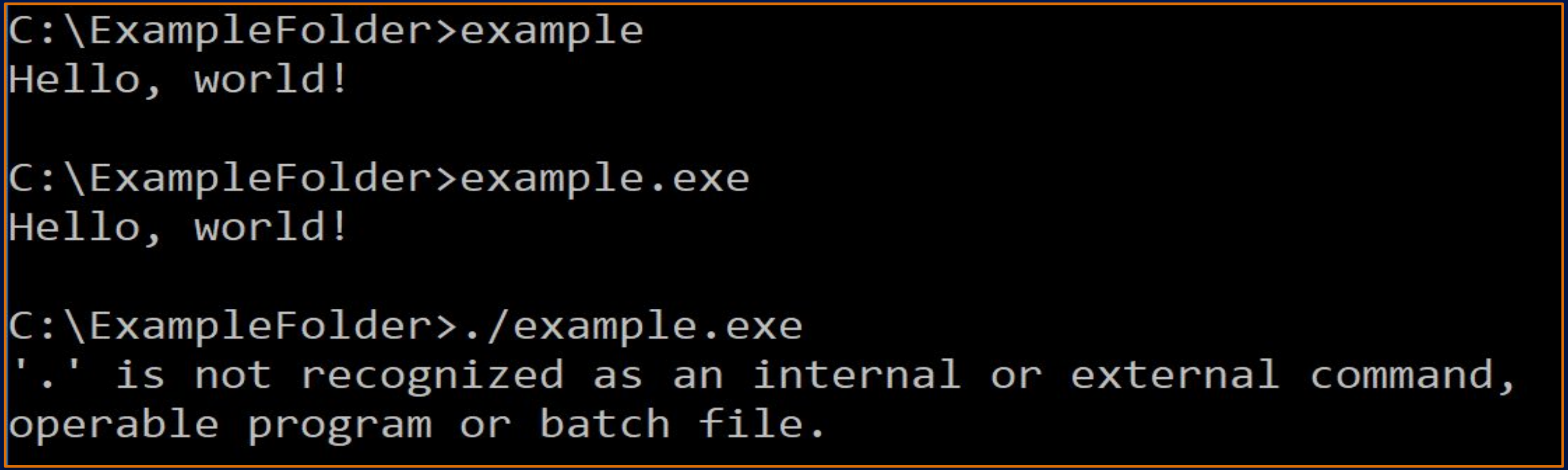
Once finished, the command prompt waits for more input.

This code turns into an executable.

```cpp
int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

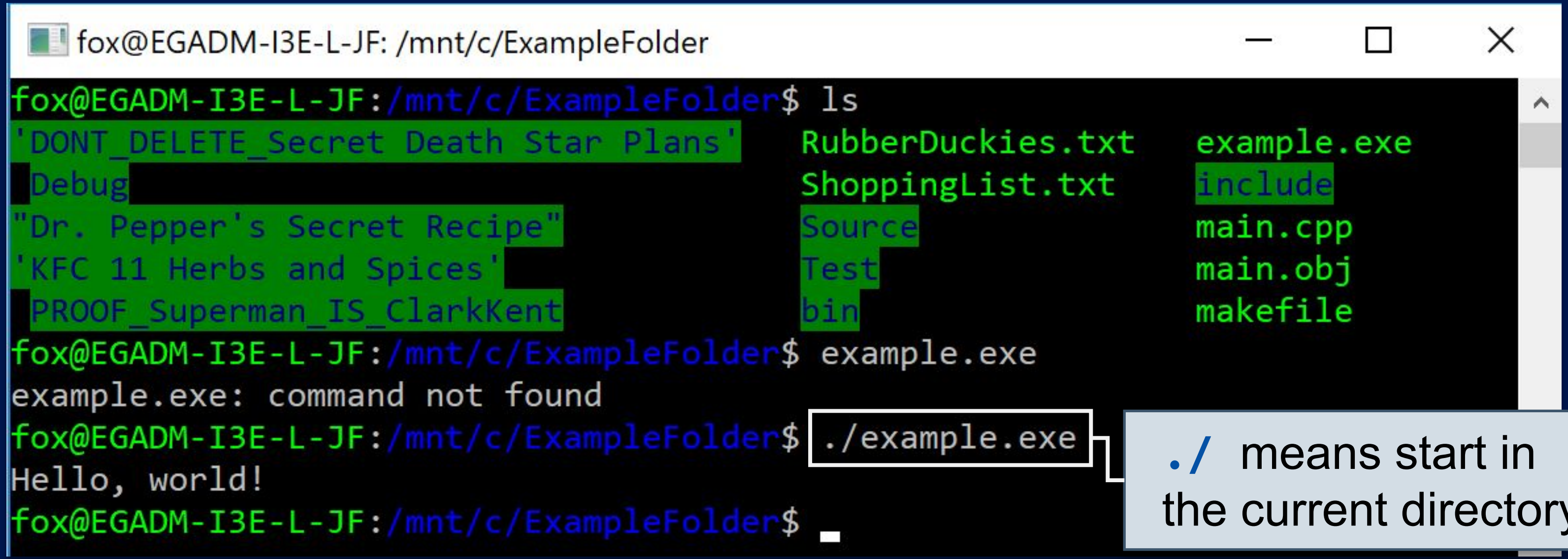
Operating Systems May Differ Slightly

Windows command prompt



```
C:\ExampleFolder>example
Hello, world!

C:\ExampleFolder>example.exe
Hello, world!

C:\ExampleFolder>./example.exe
'.' is not recognized as an internal or external command,
operable program or batch file.
```

Windows Subsystem for Linux (WSL)



fox@EGADM-I3E-L-JF: /mnt/c/ExampleFolder

```
fox@EGADM-I3E-L-JF:/mnt/c/ExampleFolder$ ls
'DONT_DELETE_Secret Death Star Plans'    RubberDuckies.txt    example.exe
 Debug                                   ShoppingList.txt     include
"Dr. Pepper's Secret Recipe"             Source               main.cpp
'KFC 11 Herbs and Spices'                Test                 main.obj
 PROOF_Superman_IS_ClarkKent             bin                  makefile
fox@EGADM-I3E-L-JF:/mnt/c/ExampleFolder$ example.exe
example.exe: command not found
fox@EGADM-I3E-L-JF:/mnt/c/ExampleFolder$ ./example.exe
Hello, world!
fox@EGADM-I3E-L-JF:/mnt/c/ExampleFolder$ _
```

`./` means start in the current directory.

Case Sensitivity

- Windows is NOT case sensitive for things like filenames, paths, and CLI commands.
  - `dir == DIR == DiR == diR`, etc…
  - `DATAFILE.txt == datafile.txt == DaTaFILE.txt`, etc
- Unix-based environments **are** case-sensitive:
  - `ls != LS != Ls`
  - `myprogram != MyProgram`
  - `folder/subfolder != folder/SubFolder`

Command Line Arguments

- Executing commands via command line can be aided by passing arguments to the executable.
- Just like arguments to a function—and programs are essentially just functions.

```
// Normal main
int main()
{
    return 0;
}
```

```
// main() that supports command-line arguments
int main(int argc, const char** argv)
{
    return 0;
}
```

Command Line Arguments

- `argc` How many arguments are there. There will always be at least one—**the name of the program itself**.
- `argv` An array of `char*` (strings), each of the arguments passed to the program

```
int main(int argc, const char** argv)
{
    return 0;
}
```

```
// Alternately...
int main(int argc, const char* argv[])
{
    return 0;
}
```

`char**` vs `char*[]`, same thing. An array of character pointers, an array of character arrays—an array of strings.

Using Arguments in Your Program

```
int main(int argc, const char** argv)
{
    if (argc > 1) // if there is more than just the executable name
    {
        // Print out all the arguments
        for (int i = 0; i < argc; i++)
            cout << "Argument #" << i << argv[i] << endl;
    }

    if (argc == 2)
        DoSomethingWithArgument(argv[1]); // Use the "first" argument
    else
    {
        cout << "Invalid arg count! Usage is: program <argument1>" << endl;
        return -1;
    }

    return 0;
}
```

We might start at 1, if we don't need the name of the executable.

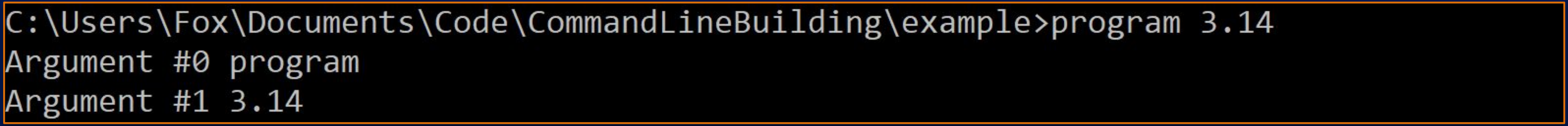
You may check for specific numbers of arguments, depending on your program.

A program might require arguments—no arguments, no working program.

The arguments themselves are just strings—once they get into your program, do whatever you want with them.

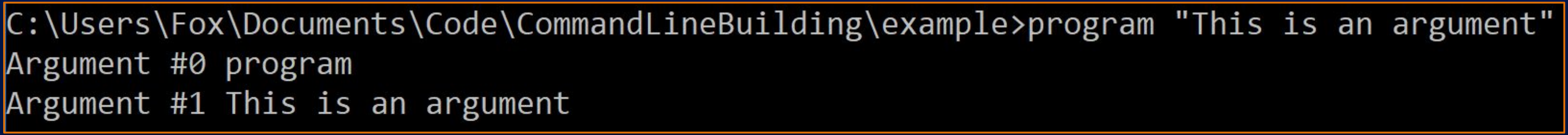
Example

- Passing a single argument to a program



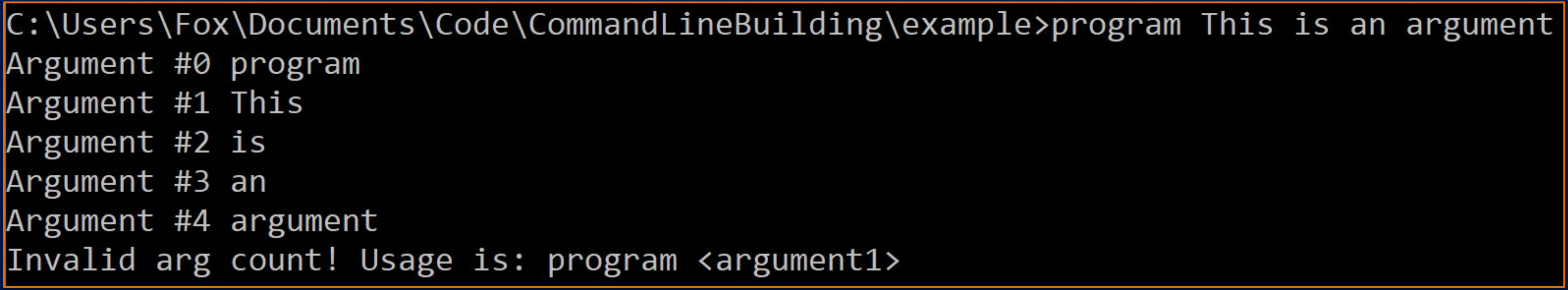
```
C:\Users\Fox\Documents\Code\CommandLineBuilding\example>program 3.14
Argument #0 program
Argument #1 3.14
```

- Passing a string as an argument requires double quotes “”.



```
C:\Users\Fox\Documents\Code\CommandLineBuilding\example>program "This is an argument"
Argument #0 program
Argument #1 This is an argument
```

- No double quotes? One argument becomes many!

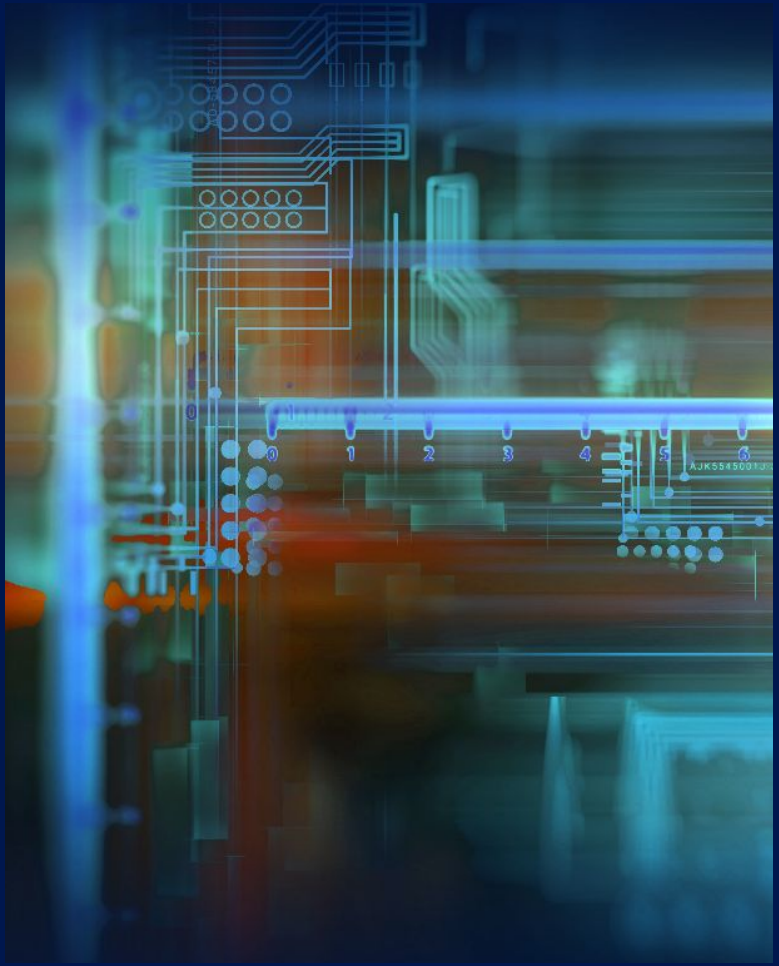


```
C:\Users\Fox\Documents\Code\CommandLineBuilding\example>program This is an argument
Argument #0 program
Argument #1 This
Argument #2 is
Argument #3 an
Argument #4 argument
Invalid arg count! Usage is: program <argument1>
```

Recap

- A command-line interface lets you interact with an operating system (OS) with **text-based commands.**
- A graphic-user interface (GUI) is a “middle man” between you and the OS.
- It can (sometimes) be faster / more convenient to work directly with the OS instead of using the GUI.
- You don’t have to learn all the commands, but you should get familiar with some of the basics (like `ls`/`dir` and `cd`).
- Data can be passed to a program with **command-line arguments**.
- We have to modify main() to support command-line arguments in our own programs.
- You may not need (or want) to master a CLI, but it’s **important to understand it**.

Conclusion



Placeholder for the instructor’s welcome message. Video team, please insert the instructor’s video here.

Thank you for watching.