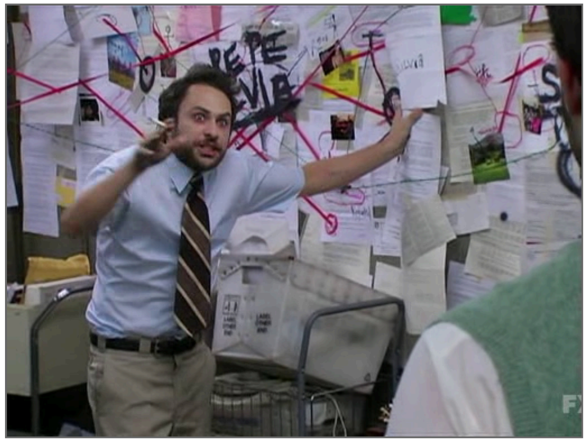
ABOUT ME

- I am a Graduate Computer Scientist
- I am an independent C++ contractor.
- I like C++
- I also like the command line.
- I am passionate about energy efficient computing.

CONTACTS

- Twitter : @bensonorina
- mastodon : @mathewbenson.hachyderm.io
- email : mathew@benson.co.ke
- Github : github.com/mathewbensoncode
- Discord : #CppAfrica → mathewbensonke

PLAN FOR THE TALK



- Rationale
- Command Line Overview
- Command Line C++
- Build Systems & Tooling

RATIONALE?

WHAT IS MY GOAL

- This is a deeper dive into the concepts I discussed in last years talk “C++ in the Developing World, Why it matters”
- An experiment at creating an environment where the C++ language can be learnt, through “Play”.
- The goal of this talk is to create a roadmap to empower any individual, anywhere to get started with learning about computing.

ANALOGY TO A MOVIE

- C++ and computing has a long history
- There is a lot of important history and concepts that are easy to overlook
- It may also be harder to enjoy when missing some plot points

WHY THE COMMAND LINE?

- Because we code in text, our compilers and other tools work with text.
- Our Integrated Development Environments try to shield us from The Text interfaces
- This may be ideal for Speed and project management in Professional Development Environments
- However, It is not ideal for Learning Environments.

PLAYGROUND

- We learn language…and other stuff by getting to play.
- Before a child starts to speak, they “play” with the sounds they hear.
- So to learn a language, you need to have a safe “playground”
- When playing, we need to slow down and allow our brains to play with concepts and ideas

COST

- It costs less to run than GUIs.
- This matters when access to power is an issue like rural areas and places where access to power is challenging

FLEXIBILITY

- The different platforms in existence today, all have their foundations in text based systems.
- Even today, “under the hood” Text is still king.

POWER

- The command line is the realm of experts.
- However, to become an expert, you need to play
- GUI based IDEs are indeed convenient, however, the convenience requires knowledge of what they do for you to make the most out of them.

WHY NOT IDES

- IDE → INTEGRATED DEVELOPMENT ENVIRONMENT.
- It is a integration of distinct components
- This integration is great for speeding up development for those already familiar with the individual components
- For learning, we need to dis-integrate in order to play with the various components

BUT...WHY C++?

- Historically, C, which is the foundation of C++ was very tightly coupled to the concepts of the unix operating system.
- C++ is the pathway to better constructs of managing software complexity while still being backward compatible with C.
- Because of the ability to link to system software.
- Many APIs that are on our systems only provide a C++ interface and others only have a C API
- Is open source really open without literacy of the language?

ABOUT THE COMMAND LINE

- Lets go through some very important terms in relation to the command line
- The first Distinction we need to make is between the “Terminal” and the “Shell”

THE TERMINAL

TELETYPE
B.SP.
ON
FREE
T E L E T Y P E

- The terminal historically was a printer and a keyboard. The printer being a “printable” surface

```
RX, address=17777170-1777717
RY, disabled
RP, Massbus adapter 0, 8 uni
RQ, address=17772150-1777215
RQB, disabled
RQC, disabled
RQD, disabled
TC, disabled
TM, address=17772520-17772533,
TS, disabled
TQ, TQ81 (10GB), address=17774
TU, disabled
TA, disabled
XQ, disabled
XQB, disabled
XU, disabled
XUB, disabled
KE, disabled
KG, disabled
CPU, 11/70, FPP, MMU, idle disabled             iguration enabled, 4088
Can not boot: real console HALT.
rsx11m_plus_1170.simh_pdp11> boot rp
Simulation stopped
sim> █
```

- This graduated to a screen and a keyboard which could be connected over long distances to a mainframe or mini-computer

- The keyboard and the screen are the most prominent and consistent components of our computer systems.
- The terminal as it exists today, is generally in programs that “Emulate” the behaviour of the traditional terminals but run in todays graphical environments.

COMMAND LINE/SHELL

- This is a program that is used to input keystrokes as text and output text to the terminal
- Text Based user interface where Commands are typed in, interpreted by the “operating system”, processes created, leading to Joy or agony
- The command line is a text based way interface to accessing the computer.
- It was the primary way of accessing the computer resources, like programs, files actions, etc before the advent of the graphical usre interface

HOW IT WORKS

- You generally start with a prompt, which is the systems way of showing you that its waiting for input
  ```
  $
  ```
- The first word you enter is the name of a program “known” by the operating system
- Any words after the first word are “Arguments” to the program
- The output from the program is displayed after running the program
- The prompt is displayed again after the previous program is done

GENERAL SHELL TYPES

- I will try to classify the “Shell” types available on different platforms

UNIX BASED(POSIX-COMPATIBLE)

- These are available on Unix like systems, which includes macs

DOS BASED(CMD)

- This is generally only available on windows and was its primary shell

POWERSHELL

- The powershell is a newer type of Shell, primarily available on windows, but also available on other platforms
- This one is kind of different from the other types, it may use text, but the text refers to “DotNet” like constructs

HOW TO GET HELP

- For Unix(POSIX) based shells, the convention for most programs is to use the Argument “-h” or “—help” after the program name
- For DOS based shells, the convention is the “/?” Argument after the program name
- Powershell based shells, the convention is the “Help” cmdlet followed by the name of the cmdlet

TEXT

BUT WHAT IS TEXT?

- Computers, are basically just big calculators, Calculate being a synonym for Compute.
- Its all just ONEs and ZEROs
- We create groups of these ones and zeros(bits) into groups, the most common being a byte which is 8 bits

ASCII

- A Standard called ASCII(American Standard Code for Information Interchange), was developed to symbolize how to represent letters, numbers and characters.
- This standard was actually initially designed for Transmiting telegrams over large distances.
- ASCII, consists of 128 values for representing a set of visible characters and other values that are used for communicating different values in numerical form.
- ASCII, represents each character as a 7 bit value

UNICODE

- As ASCII was developed in “English” speaking nations, its character set was sufficient for that language
- However, in order to accomodate other writing systems, character types, different approaches were used
- These all did retain the idea of ASCII
- It got really messy…but that isn’t the focus of this talk
- In the internet world of today we use “UTF-8” for representing text in our terminal emulators and many other applications

TEXT IN C++?

- According to cppreference.com, A C++ program is a sequence of text files(typically header and source files)that contain declarations
- These undergo, translation to become and executable program which is executed when the C++ implementation calls its main function.
- The smallest data type in C and C++ is the char, which is an 8 bit value also known as a byte.
- The char is able to handle the ASCII character codes and is quite ubiquitous in alot of “programs” in the wild.

WHAT IS A PROGRAM?

- Our computers, don’t really run on text, the CPUs understand a fixed set of instructions, represented in Ones and Zeros, that control and co-ordinate these instructions many times a second, to produce the wonderful computing experience we know and love.
- When we write our code in text, in the C++ language(there are other languages, but we don’t talk about those in a C++ conference), it gets stored in a file(which is just a sequence of ones and zeroes stored on disk).
- This file is “passed” through the command line to another program called the compiler, which does the translation into the kind of instructions understood by the computer system.

COMPILED VS TRANSLATED

- There are many languages in existence, but they generally fall into two broad categories, Compiled and Translated.
- C++ is Compiled language, it has to be full translated and put together before being run on the computer.
- There exist languages that do this translation in “real” time. These are called translated languages. This translation happens each time the program is run.

COMMAND LINE C++

THE C++ COMPILER

- The Compiler is a text based program that reads in “plain text files” with C++ in them
- It creates output files which are runnable programs
- If unsuccessful in compilation it gives output explaining what happened

GETTING TO KNOW YOUR COMPILER

- The compiler has many options that control the process of taking your text program to machine code
- Here is a start to exploring what is possible

GCC AND/OR CLANG

```
$ c++ --help
$ info g++
$ clang++ --help
```

MSVC(CL.EXE)

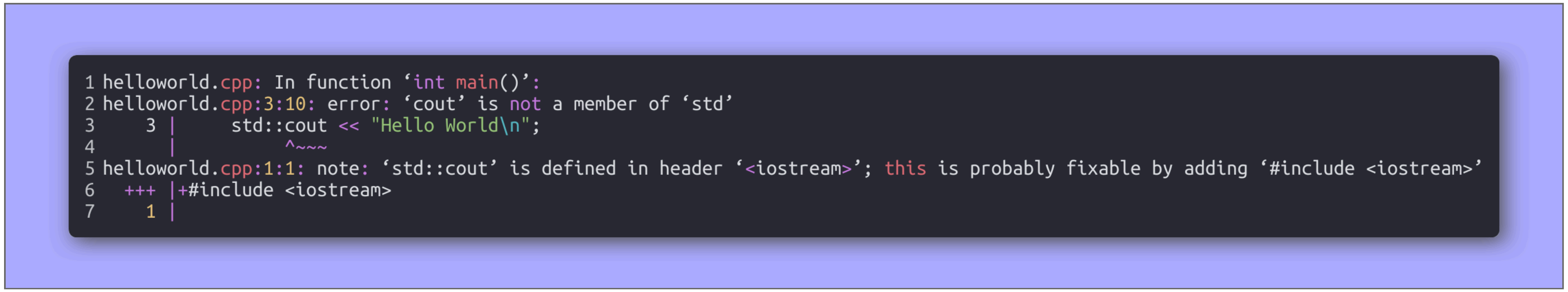
```
$ cl.exe /?
```

YOUR FIRST C++ PROGRAM

HELLO WORLD USING GCC

```cpp
   // File ⇒ helloworld.cpp
int main(){
    std::cout<<"Hello World!\n";
}
```

```
$ c++ helloworld.cpp
```



```
helloworld.cpp: In function ‘int main()’:
helloworld.cpp:3:10: error: ‘cout’ is not a member of ‘std’
    3 |     std::cout << "Hello World\n";
      |          ^~~~
helloworld.cpp:1:1: note: ‘std::cout’ is defined in header ‘<iostream>’; this is probably fixable by adding ‘#include <iostream>’
  +++ |+#include <iostream>
    1 |
```

We Get a Compiler Error

- The compiler will give an explanation as to why it didn’t succeed in producing a valid program
- This is in text form…(which is important for later)
- Also Note the File name and Position information of the error
- IDEs and other tools use this information to aid in development

HELLO AGAIN WORLD

```cpp
  // File => helloworld.cpp
#include <iostream>

int main(){
    std::cout<<"Hello World!\n";
}
```

```
$ c++ helloworld.cpp
```

- The program now compiles.
- Because we didn't specify the output name, a default name "a.out" is given to the resulting executable

WHAT ABOUT WINDOWS?

- Microsoft has its own implementation of the C++ compiler.
- To achieve the same as above we use the the “cl.exe” program

HELLO WORLD ON WINDOWS

```cpp
 // File ⇒ helloworld.cpp
#include <iostream>

int main(){
    std::cout<<"Hello World!\n";
}
```

```
$ cl.exe helloworld.cpp
```

- The MSVC compiler cl.exe does not follow the a.out convention
- It creates an executable with the same name as the source file

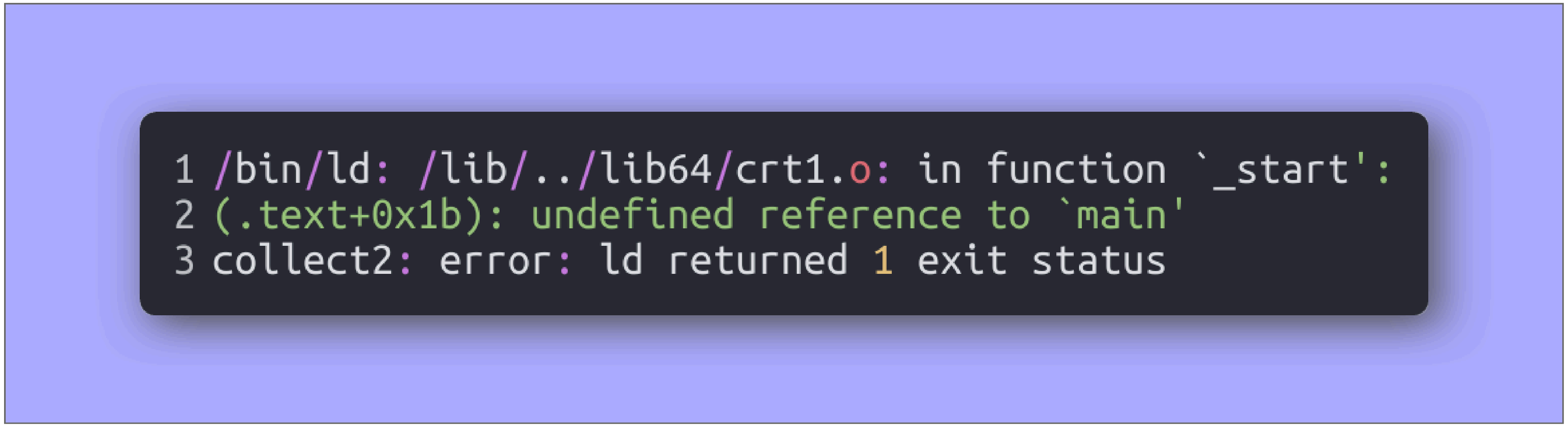
LIBRARIES

- Lets Refactor our hello world program, we'll move the functionality into a function called greet and try compiling

```cpp
// File ⇒ libgreet.cpp
#include <iostream>

void greet(){
    std::cout<<"Hello World From Greet!\n"
}
```

```
$ c++ libgreet.cpp
$ cl.exe libgreet.cpp
```



```
/bin/ld: /lib/../lib64/crt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
```

No main Function

- Runnable programs require a main function
- When we try to compile without any arguments, the compiler tries to create an executable

STAGES OF COMPILATION

The Compiler is actually does its work in a few different stages

Pre-Process

Assemble

Compile

Link

CONTROLLING THE COMPILER PROCESS

- Until now, we have been passing the name of a text file with the extension .cpp to the compiler
- The compiler assumes this to be an executable and tries to do the full process
- With Some compiler flags it is possible to state where you want the process to stop

PRE-PROCESS

- This is usually(by convention) done with the ‘E’ argument
```
$ c++ -E libgreet.cpp
$ cl.exe /E libgreet.cpp
```
- It produces alot of text output to the console.
- This is because it copies any “included” file combines it with our source file to create the actual file to be compiled
- The #include <iostream> file is really big
- There should be flags to pass this into a file

ASSEMBLE

- This takes the pre-processed file and converts it into Assembly Code, which is a human readable form of machine code
  ```
  $ c++ -S libgreet.cpp
  ```
- This produces a file with the .s suffix and the same name as the source file which contains the assembly code of the translation unit
- I don’t know if there is a way to stop at this stage with the MSVC compiler.

COMPILE

- This translates the translation unit into machine code
- By convention this uses the /c flag
- The output of this stage is an “Object” file by convention with the suffix .o

  ```
  $ c++ -c libgreet.cpp
  $ cl.exe /c libgreet.cpp
  ```
- Note that the libgreet.cpp file compiles, i.e. there is no output after compilation

LINK

- In this stage, the “Compiler” Front-End links together several object files to create the actual runnable program.
- The output of this stage may also be an “LIBRARY” archive containing the combination of several object files which can be linked to in the link process of another executable.
- This is one of the “Super Powers” of C++. The ability to combine with “Other” peoples object files and Libraries

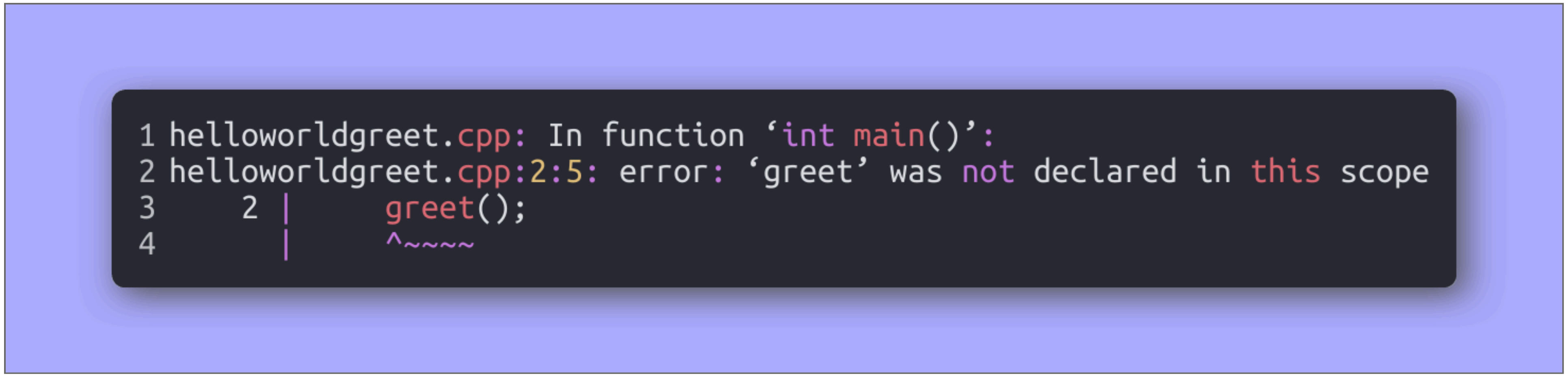
BACK TO REFACTORING THE GREETING PROGRAM

HELLOGREET.CPP

```cpp
// file ⇒ hellogreet.cpp
int main(){
    greet();
}
```

```
$ c++ hellogreet.cpp librarygreet.cpp
```

- We can compile more than one translation unit at a time and the “Compiler” infrastructure should link them together



```
helloworldgreet.cpp: In function ‘int main()’:
helloworldgreet.cpp:2:5: error: ‘greet’ was not declared in this scope
    2 |     greet();
      |     ^~~~~
```

We get a compiler error!

- This is because the hellogreet.cpp has no idea about the greet function.
- .cpp files are Translation units and are “translated” independently

HEADER FILES

- Lets Create a Header file, with the extension .hpp that contains a “Declaration” of the function that is shared between the two translation units

```cpp
// file ⇒ librarygreet.hpp

void greet();
```

- We will include the line #include "librarygreet.hpp" into both the librarygreet.hpp and the helloworldgreet.cpp files

```cpp
//file ⇒ librarygreet.cpp
#include <iostream>
#include "librarygreet.hpp"
  ...

```

```cpp
//file ⇒ helloworldgreet.cpp
#include "librarygreet.hpp"
  ...
```

```
$ c++ helloworldgreet.cpp librarygreet.cpp
```

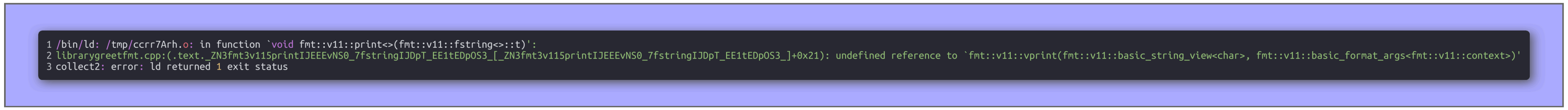
EXTERNAL LIBRARY

- Lets try and use the fmt library to Format our greeting

```cpp
// file ⇒ librarygreetfmt.cpp
#include <fmt/format.h>
#include "librarygreet.hpp"
void greet(){
 fmt::print("Hello Format World\n");
}
```

- To do this, we #include <fmt/format.h>, which is one of the Headers for the fmtlib API

- Lets now compile the greet application as before, but this time with the librarygreetfmt.cpp file
  $ c++ librarygreetfmt.cpp helloworldgreet.cpp



```
/bin/ld: /tmp/ccrr7Arh.o: in function `void fmt::v11::print<>(fmt::v11::fstring<>::t)':
librarygreetfmt.cpp:(.text._ZN3fmt3v115printIJEEEvNS0_7fstringIJDpT_EE1tEDpOS3_[_ZN3fmt3v115printIJEEEvNS0_7fstringIJDpT_EE1tEDpOS3_]+0x21): undefined reference to `fmt::v11::vprint(fmt::v11::basic_string_view<char>, fmt::v11::basic_format_args<fmt::v11::context>)'
collect2: error: ld returned 1 exit status
```

Linker Error

- To resolve this, we need to pass into the the compiler an argument that tells us about the Library Binary to include in the linking process to enable us to get the object files for the library to be joined with our object files

  ```
  $ c++ -lfmt librarygreetfmt.cpp helloworldgreet.cpp
  ```

NEXT STEPS AND FURTHER RESOURCES

- C++Weekly - list of videos for learning
- CMake
- New windows Shell
- Neovim for C++

CONCLUSION

- The slides above were really just an example of how to play on the command line using command line tools instead of IDEs.
- There are many more areas to explore, This was just a roadmap of how to get to play with some of the core flags of the command line
- The command line is an accessible way to learn to program with C++. Take advantage of it

Speaker notes