SER 334 Environment Setup

In this writeup we will be installing and configuring VirtualBox, CLion, and optionally Git and GitHub integration.

IMPORTANT NOTE FOR MAC USERS

VirtualBox will not work on the new M1 Macs that run on ARM architecture. Such systems are outside the scope of this tutorial, however, there may be options you can use. ASU has a limited number of terminals available for remote development; contact the professor for more details. In addition there are other virtualization options like UTM that may work better for you. In that case, skip over the VirtualBox section and straight to setting up Xubuntu.

DEVELOPING FOR THIS CLASS

The only firm requirement for this class is that your code needs to compile and run in Ubuntu. It is not a requirement that the development must happen inside a virtual machine. You don't strictly need to have an IDE set up inside your VM, especially if you have limited system resources. It is perfectly sufficient to write code in the host OS and just run it in the guest OS, or natively if you have a Linux install. Just make sure you test it in the expected environment; we've had issues with people running assignment code in OSX and it doesn't behave properly, but in Ubuntu it works perfectly.

That said, some of the given code for the assignments requires Linux headers that may not be available in Windows. The IDE's static analysis tools will throw an error when you try to use a function out of a header file that doesn't exist, so this must either be worked around or ignored until it's back in its expected environment.

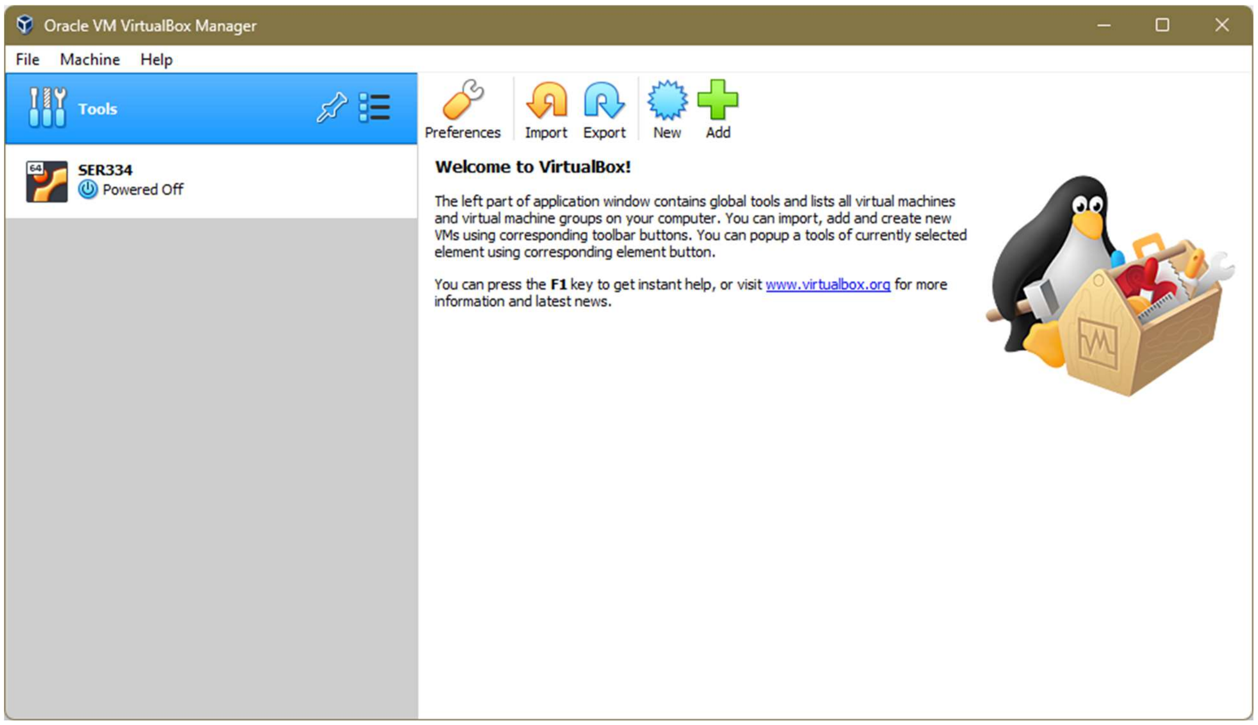
You do need a running virtual machine, even if you have a native Linux setup, for one of the assignments where you make a loadable kernel module. You'll be injecting code into a running kernel and you may not want to do this on a machine you care about. Making a snapshot of the box is a good idea when you're touching core system code.

Part I: Installing and Setting Up VirtualBox

We'll need a couple of things to get up and running.

- VirtualBox: Virtualization software uses your system resources to run an entire computer inside of a window. This allows us to run Linux inside of Windows.
- Xubuntu: A flavor of Ubuntu with lighter graphical elements. The recommended version for this course is **20.04**. Other flavors of Ubuntu may work but cannot be specifically supported by the teaching team. You'll need the .iso file, which is a disk image, to install the operating system.

With the necessary files, we can proceed. Install VirtualBox wherever you would like, then launch it to find the start screen.



Welcome to VirtualBox!

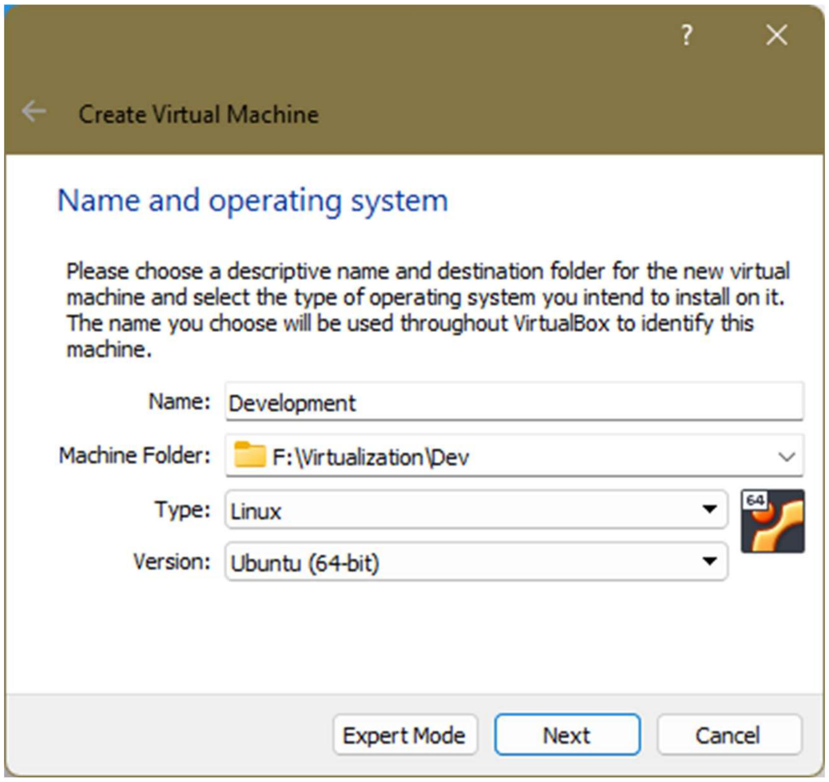
The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

I have a machine ready to go, but we'll set up a new one to see the process. Click the New button to create a new virtual machine. In the next dialog box, name your machine (it doesn't matter, it could be something to do with 334 like mine, or if you plan to save it to use with later courses it could be something general like "Development Environment."

The "machine folder" is where the machine's storage and config files will be stored. You'll want to choose a location with plenty of space if you have multiple drives.

The Type can be Linux and Version Ubuntu, but as far as I know this just changes the icon. It may also autodetect once you install your operating system.

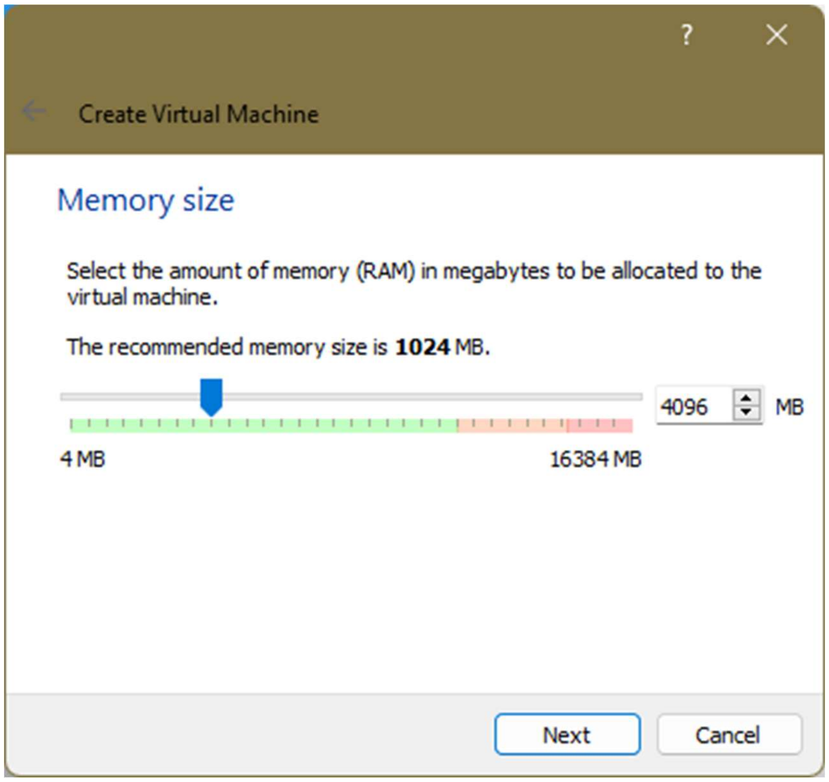

Create Virtual Machine

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: Development
Machine Folder: F:\Virtualization\Dev
Type: Linux
Version: Ubuntu (64-bit)

Allocate System RAM


Create Virtual Machine

Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

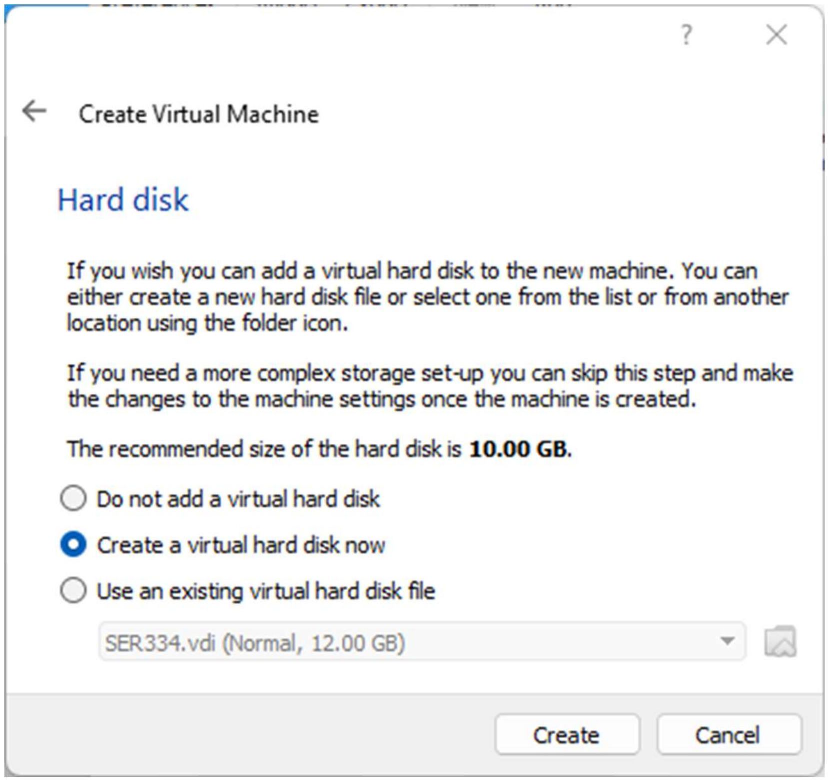
The recommended memory size is **1024** MB.

4096 MB

We need a decent amount of RAM because CLion is a fairly heavyweight application. If system RAM is very limited, there are other options like a more lightweight IDE or even just text editor and command line to run programs. I recommend 4 GB of RAM as shown in the snapshot, but don't go under 2 GB.

Create Hard Disk

Next we'll allocate some disk space. You don't need to go crazy with the amount of space, but I've had just general development prerequisites fill up smaller machines. I recommend between 15-25 GB. I think 12 is too small.



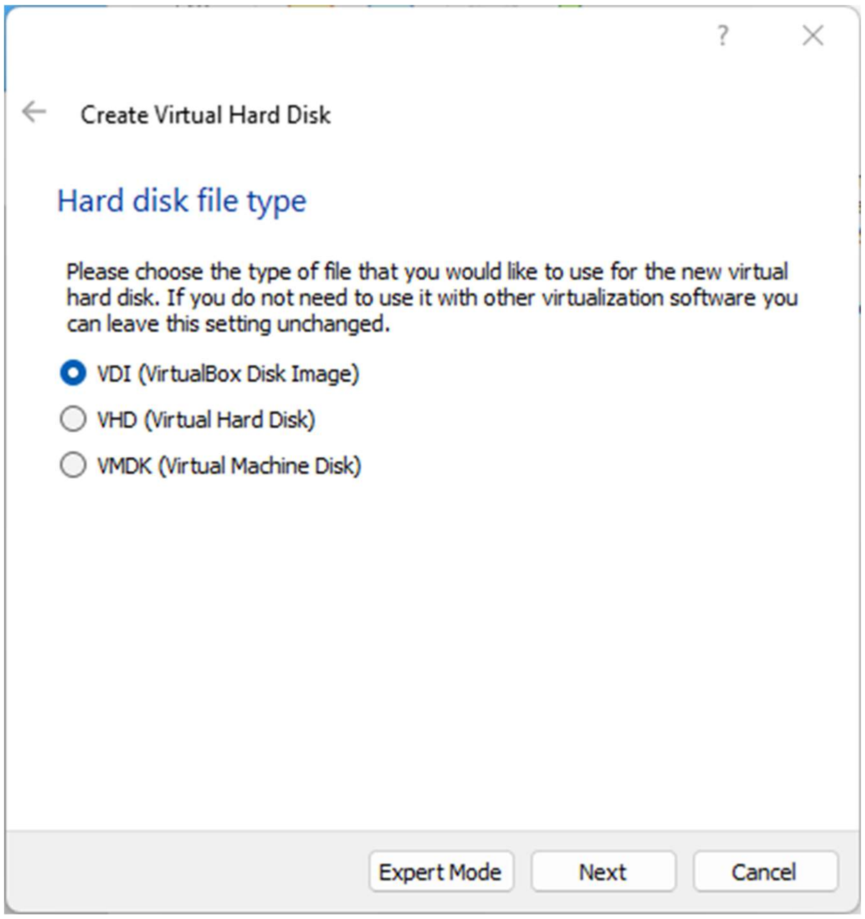
If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **10.00 GB**.

- ○ Do not add a virtual hard disk
- ● Create a virtual hard disk now
- ○ Use an existing virtual hard disk file

SER334.vdi (Normal, 12.00 GB)

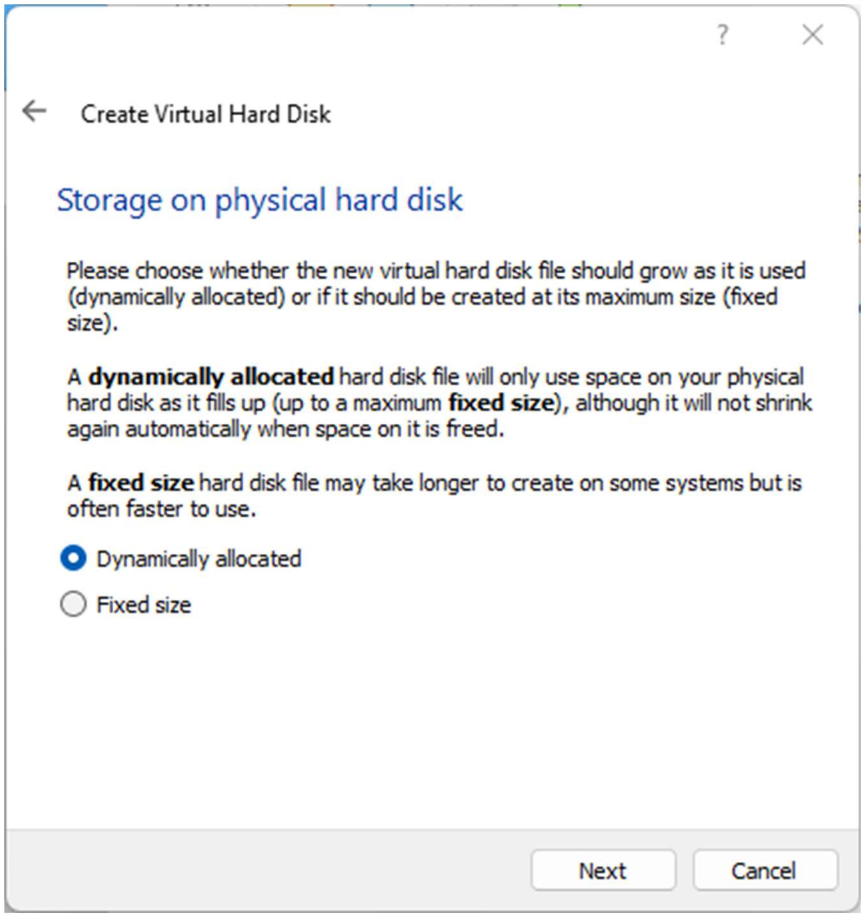

Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

[✓ VDI (VirtualBox Disk Image)]
VHD (Virtual Hard Disk)
VMDK (Virtual Machine Disk)

For the next dialog box, it's up to you and your machine. If you're strapped for space you can do dynamically allocated, if you're not worried about it or how long it will take you can do fixed size. Doesn't matter too much which one.



Create Virtual Hard Disk

Storage on physical hard disk

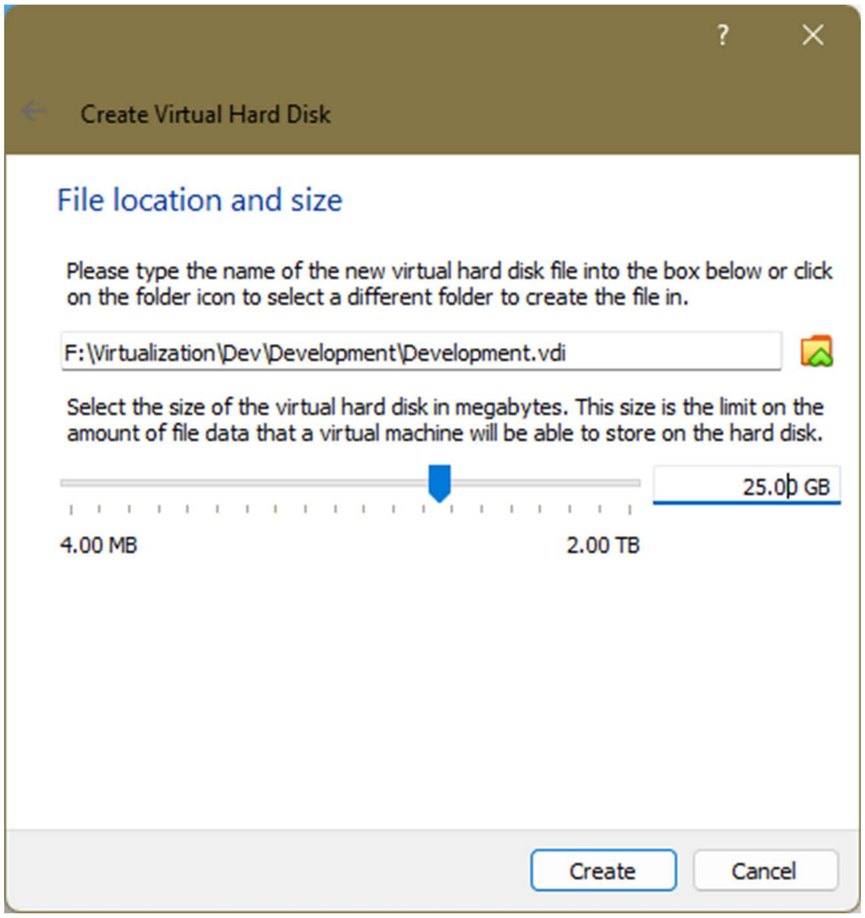
Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

● Dynamically allocated

○ Fixed size


Create Virtual Hard Disk

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

F:\Virtualization\Dev\Development\Development.vdi

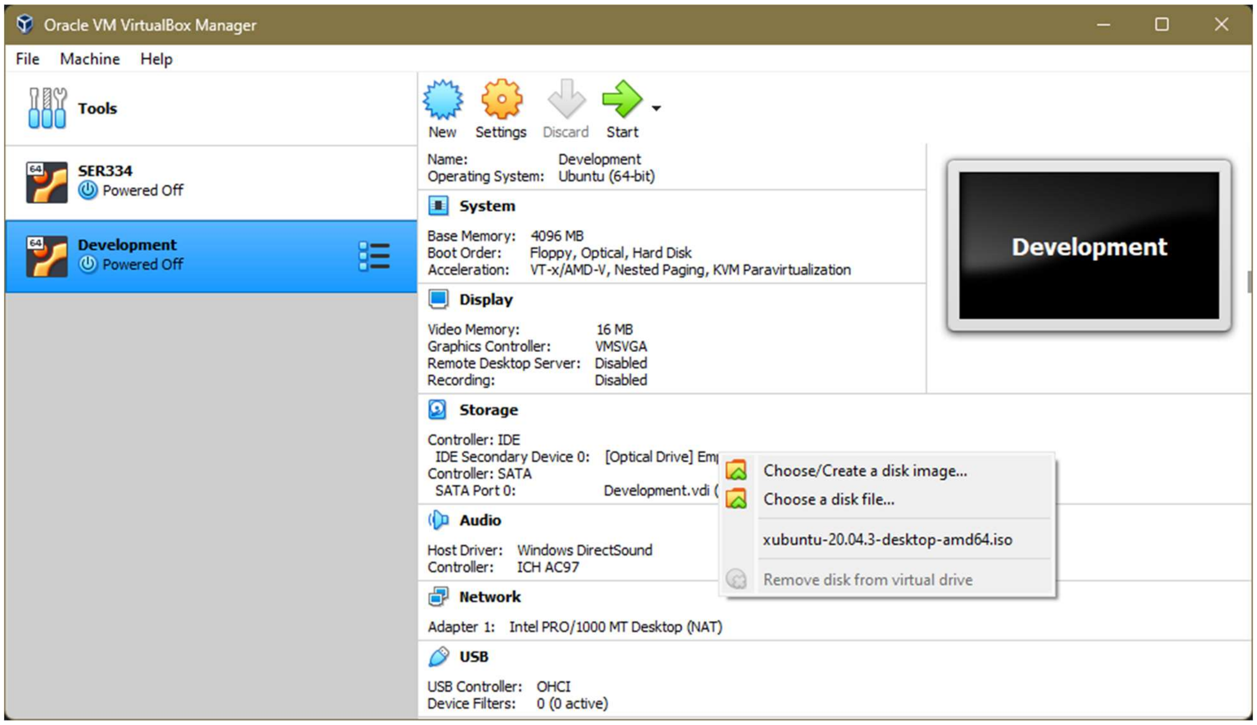
Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

25.00 GB

After this step we have a virtual machine ready to go.

Installing Xubuntu

To load the disk image into the virtual drive, make sure your machine is selected and click the text under the Storage heading that says “[Optical Drive] Empty”


Oracle VM VirtualBox Manager
File
Machine
Help
Tools
SER334
Powered Off
Development
Powered Off
New
Settings
Discard
Start
Development
Choose/Create a disk image...
Choose a disk file...
xubuntu-20.04.3-desktop-amd64.iso
Remove disk from virtual drive

Name: Development
Operating System: Ubuntu (64-bit)

System
Base Memory: 4096 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display
Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: IDE
IDE Secondary Device 0: [Optical Drive] Em
Controller: SATA
SATA Port 0: Development.vdi (

Audio
Host Driver: Windows DirectSound
Controller: ICH AC97

Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB
USB Controller: OHCI
Device Filters: 0 (0 active)

Select "Choose a Disk File," browse to where your image is stored, and select it. The text will change to show the image is loaded. Now we can start the machine.

Side Note

If you try to start the machine and it shows a Kernel Panic error like my machine does, go to Settings->System->Processors and add a virtual processor with the slider, move it from 1 to 2.

When the machine launches, it'll bring up an Install screen. Press "Install Xubuntu." The next few prompts are self explanatory, language, keyboard setup, time zone stuff. On the updates screen it may help on some systems to download updates while the OS is installing and install third-party software for graphics and wi-fi hardware.

Continue through to "Erase Disk and Install Ubuntu." Don't worry, this is the virtual disk that we created, so go ahead and press Install Now. Then you'll set up your name and your computer's name, password, etc. Then the system files will install. This will take a little while depending on your system.

Once installation has finished, don't forget to remove your installation CD from the virtual drive through Devices->Optical Drives->Remove Disk from Virtual Drive. The computer may freeze but if the installation is completed this is OK.

Just stop it or reset it through Machine->Reset. Skipping this step will cause the disk to try to install Xubuntu again which we don't need to do.

Configuring Xubuntu

When Xubuntu loads back in, login and it will prompt you to upgrade to 22.04. Click No, but follow the prompt to update your operating system through the Software Updater. If the machine is on the internet we want the latest security updates, even for virtual machines.

The first thing you'll notice is the program runs in a tiny window and you can't copy and paste things from the virtual machine to the host machine and vice versa. After the security updates finish, open a terminal by right-clicking and selecting "Open Terminal Here."

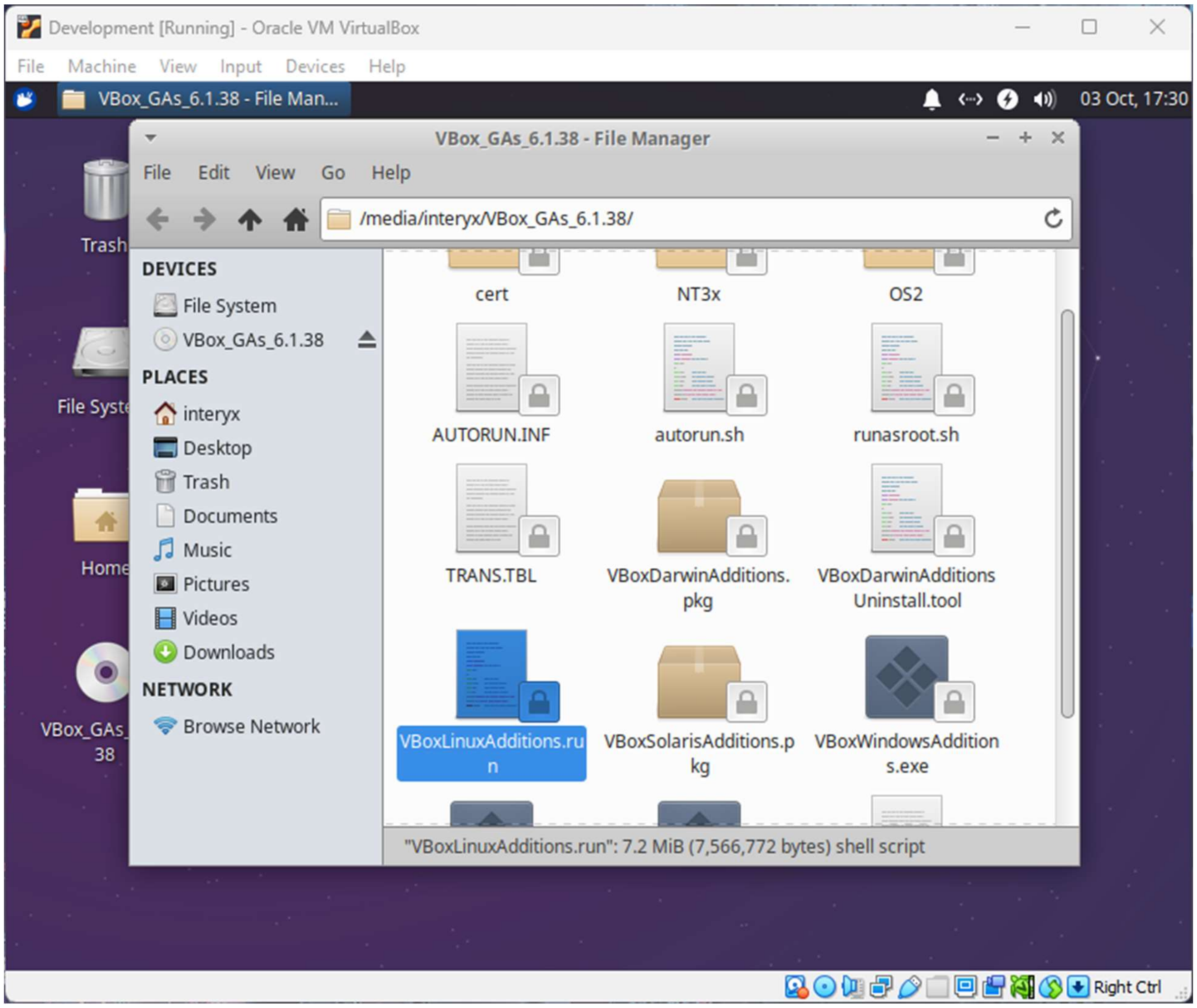
Installing Guest Additions (Optional)

This step is optional but recommended. VirtualBox can integrate with a virtual machine to make using it much easier, including automatically resizing the display to match the size of the window, allowing copy and paste, etc.

In your terminal, enter `sudo apt install build-essential`.

This will install a set of tools for compiling and debugging software including the whole c/c++ toolchain. This will be used for building the files included in the Guest Additions script.

Next, from the Devices dropdown menu select Insert Guest Additions CD Image. You should see the disc pop in on your desktop and the folder opens. If not, open the disc by clicking on it and you'll see a folder full of files. We want VboxLinuxAdditions.run.



"VBoxLinuxAdditions.run": 7.2 MiB (7,566,772 bytes) shell script

Copy the file to the desktop, then open a terminal there. The syntax to execute a file is ./<filename>. In this case it's `./VboxLinuxAdditions.run`. But this needs administrator privileges, so we need to execute this command as a superuser. Try `sudo ./VboxLinuxAdditions.run`

If the toolchain is installed correctly, this command will make the script build the code needed to make the guest additions run. Once everything is done the system needs to be restarted. The quickest way is through the terminal you're already in with the command `sudo reboot now`

Once the machine is restarted, if you resize the VirtualBox window it should also resize the desktop. If that worked, you should be clear to enable the shared clipboard and Drag and Drop in the Devices menu. The bidirectional setting makes the most sense so you can get code and files in and out of your machine easily.

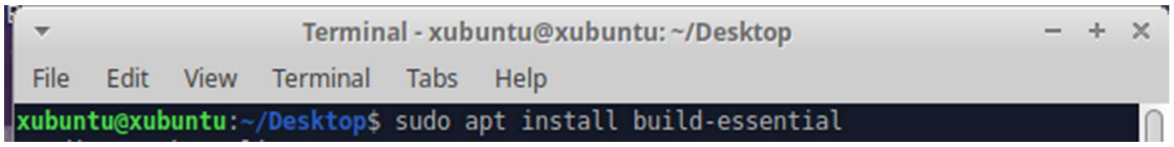
Next up is installing CLion.

Installing and Configuring CLion

Procedure

1. Install prerequisites.

If you haven't already, open a terminal and enter `sudo apt install build-essential`

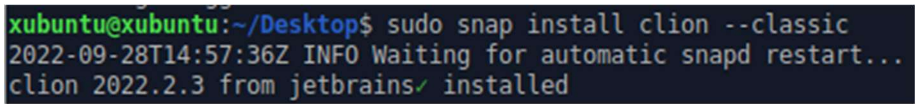


```
xubuntu@xubuntu:~/Desktop$ sudo apt install build-essential
```

This will download the toolchain needed to build and debug programs in C.

2. Install CLion

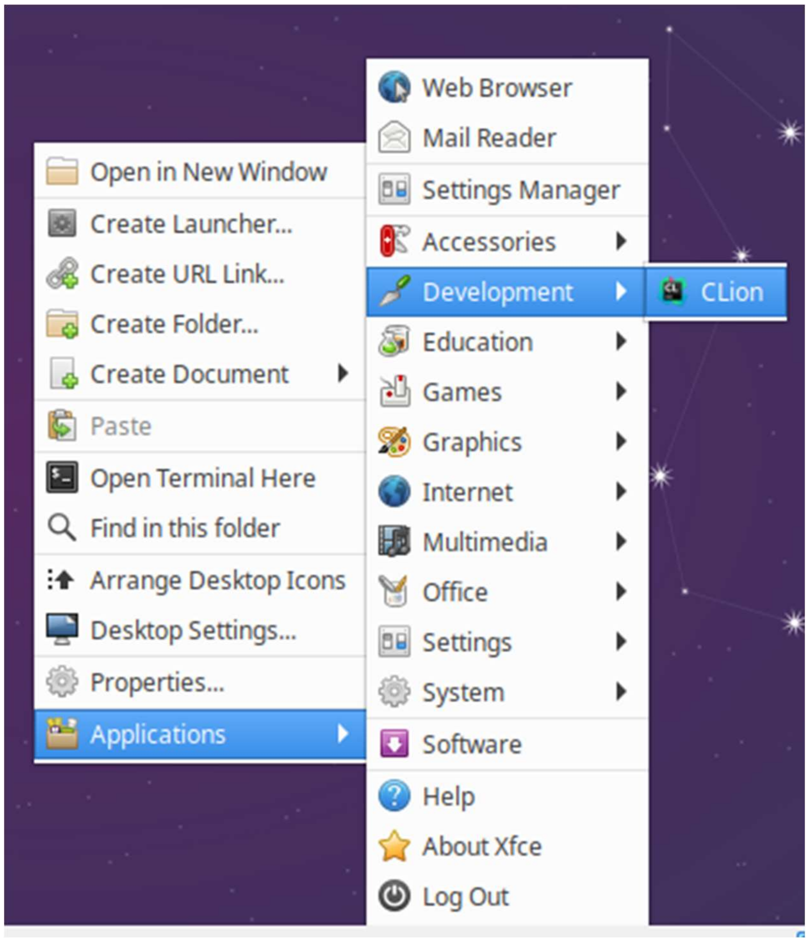
Open a terminal and enter `sudo snap install clion -classic`



```
xubuntu@xubuntu:~/Desktop$ sudo snap install clion --classic
2022-09-28T14:57:36Z INFO Waiting for automatic snapd restart...
clion 2022.2.3 from jetbrains✓ installed
```

This command uses the snap store (preinstalled with Ubuntu 16.01 and above) to install an app with all its required dependencies. By default snap applications are fairly self-contained with limited access to the outside system; because we need to open and close files and such, the –classic flag tells the system to install it as an application with full access to the system.

3. Run CLion for the first time.



The application is accessible through either the main menu (located in the upper left corner) or the right-click menu, under Applications->Development->CLion.

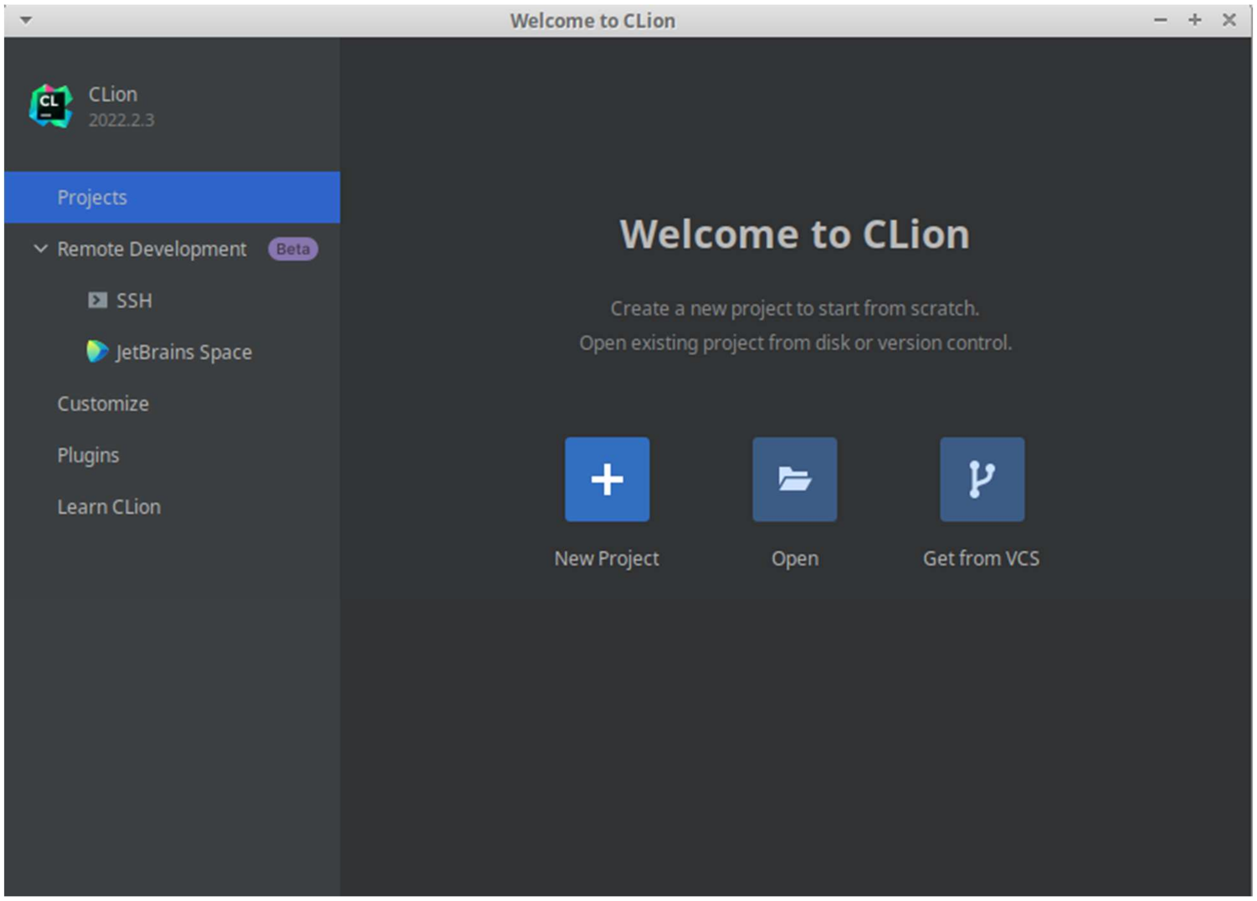
On first start, it will prompt you to login. CLion is paid software, but don't worry, because they offer free access to students. Create an account and visit the JetBrains Educational Discount page. Under the "For Students and Teachers" tab, hit the big blue "Apply Now" button.

On the next page it will ask for your information. However, entering your email address is not sufficient evidence that you are a student as ASU allows non-students to keep their email addresses. To properly verify student status, visit the ASU Unofficial Transcript page. This will generate a report showing enrollment in classes that JetBrains accepts as evidence you are a student. Make sure pop-ups are turned on when you generate a transcript. Back on the JetBrains education application site, fill out the University Email Address form, and on the next page it will prompt for additional proof. Upload the enrollment verification PDF and wait.

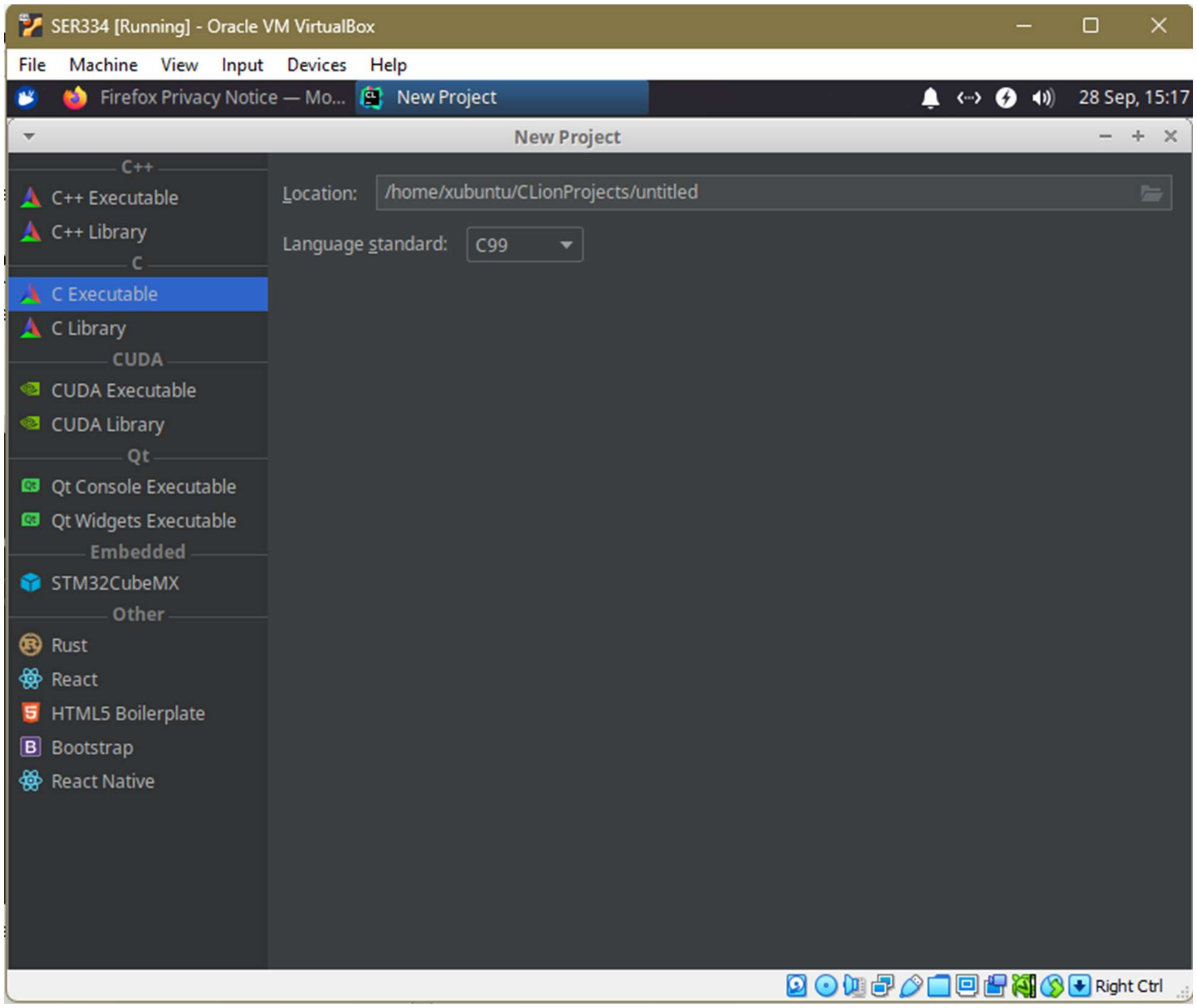
While your enrollment is being verified you can start a free two-week trial with your new JetBrains account. Once verification is complete, you can simply login to your account to provide the license to the application.

First CLion Project

Past the login screen you'll see this:



We'll start a new project by clicking New Project.

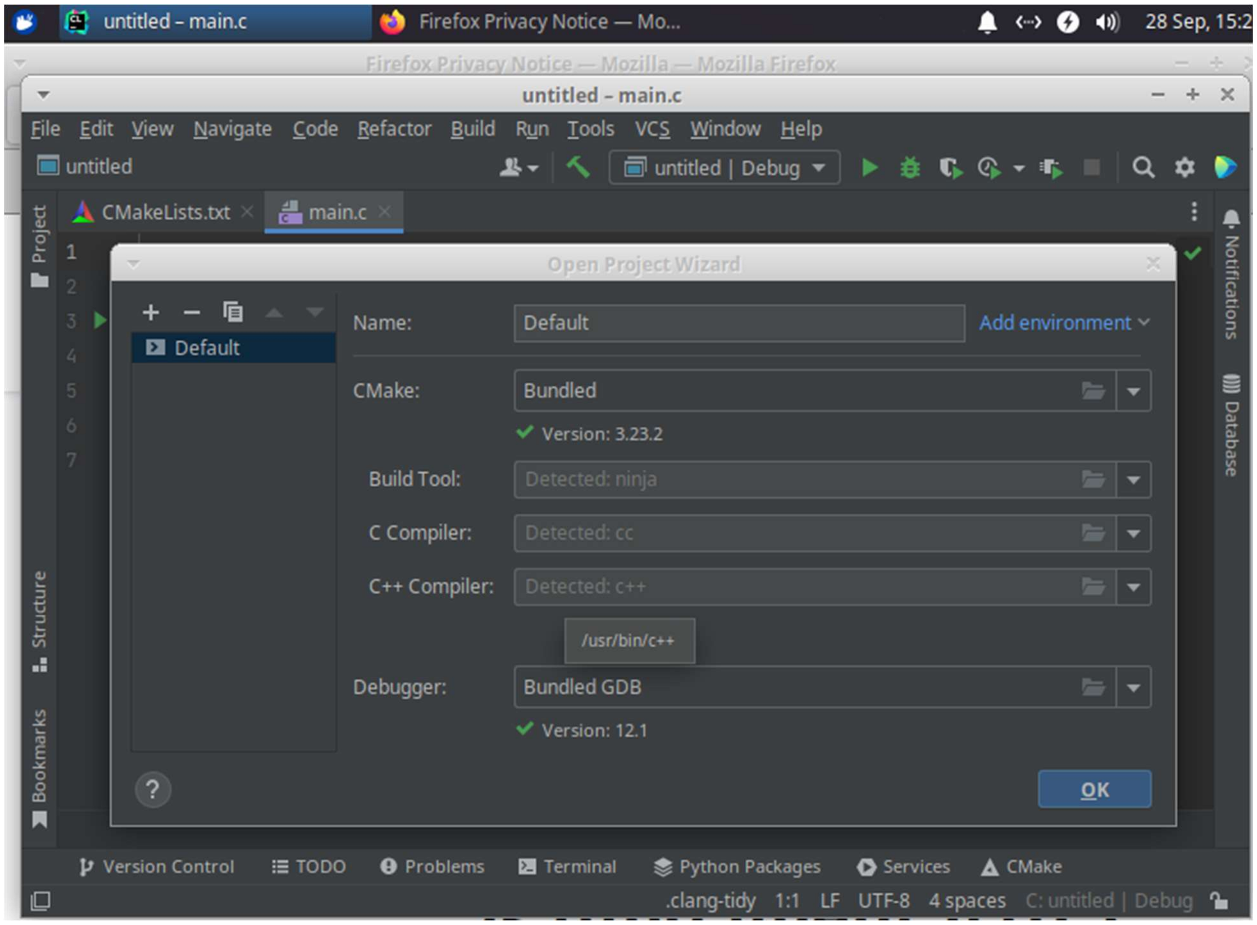


Location: /home/xubuntu/CLionProjects/untitled

Language standard: C99

For this class we'll be making C Executables when we need to write new code. Mostly we'll be importing existing code and we'll cover that later.

Note that it has a default location for new projects, which is ~/CLionProjects. (For those unfamiliar with Linux syntax, the tilde ~ is a shortcut for /User/Home). This can be changed if you like. Personally I like to keep everything in ~/dev but that's all personal preference. Make sure you don't accidentally save it to /dev because that's a top-level folder with system files in it. The C99 standard is fine as well for our purposes. Once that's done press Enter (there might also be a button if you have higher resolution than my little demo window). CLion will get the project name from whatever you save the folder as. Next you'll be presented with this:


Name: Default

CMake: Bundled
Version: 3.23.2

Build Tool: Detected: ninja

C Compiler: Detected: cc

C++ Compiler: Detected: c++

/usr/bin/c++

Debugger: Bundled GDB
Version: 12.1

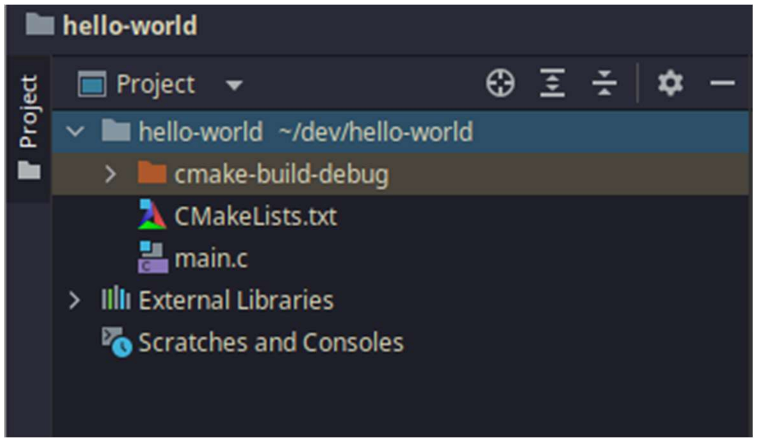
If this screen looks different, it's most likely because you're missing the build tools provided by `build-essential.` You'll need to go back Step 1 of this tutorial and run the command. Then CLion should be able to find and import your toolchain. Press OK and you'll see that there's an autogenerated Hello World program.

(At this point my demo VM froze, which is why following screenshots will look different)

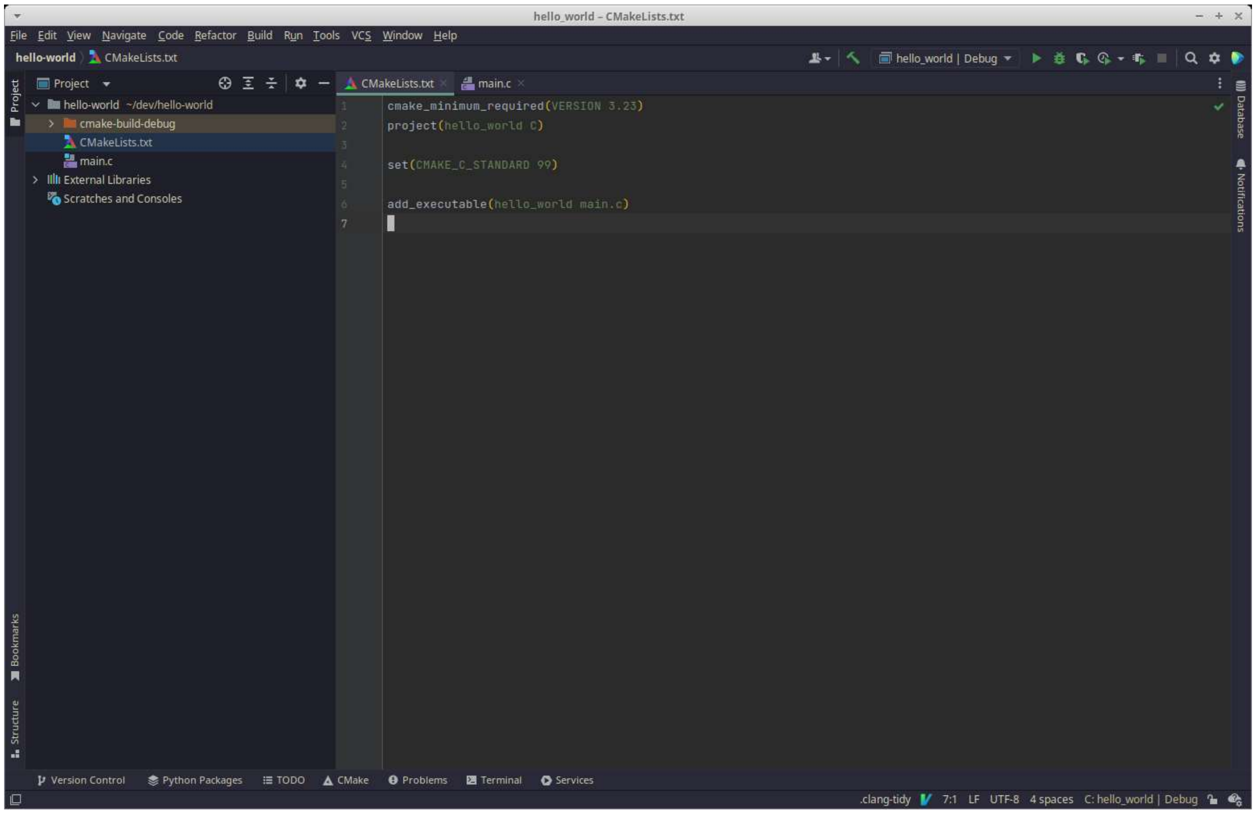
Tour of CLion

To get started we'll take a look at some essential functions of CLion and how it works editing and running your programs.

To get the file structure, CLion looks at an entire project folder. This is accessible through the Project tab on the left. Here I have a project called "hello-world" with the associated files.

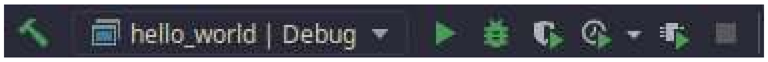


Main.c is pretty self-explanatory, that's where the main file code is stored. CMakeLists.txt is where the project settings are stored. Let's take a look.

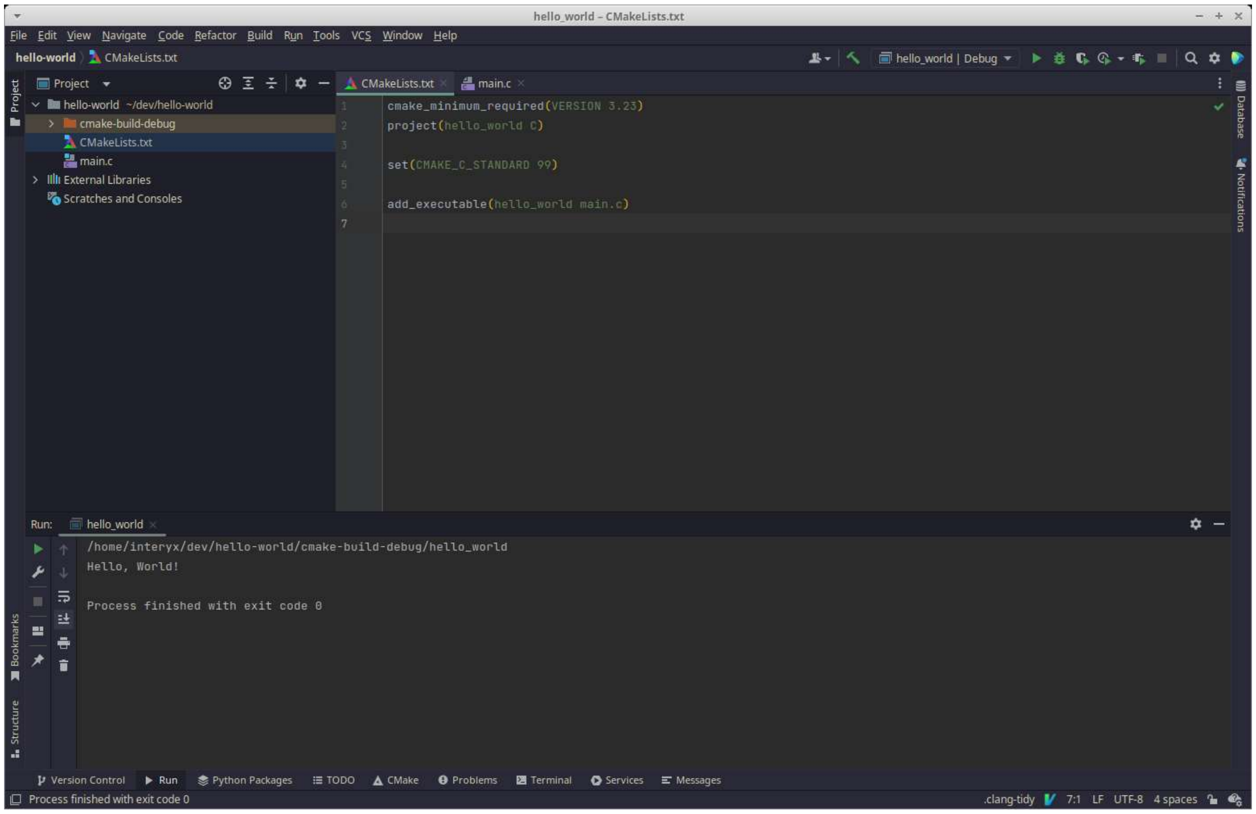

hello_world – CMakeLists.txt
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
hello-world CMakeLists.txt
hello_world | Debug
Project
hello-world ~/dev/hello-world
cmake-build-debug
CMakeLists.txt
main.c
External Libraries
Scratches and Consoles
CMakeLists.txt
main.c
Version Control
Python Packages
TODO
CMake
Problems
Terminal
Services
.clang-tidy 7:1 LF UTF-8 4 spaces C: hello_world | Debug

```
cmake_minimum_required(VERSION 3.23)
project(hello_world C)

set(CMAKE_C_STANDARD 99)

add_executable(hello_world main.c)
```

The project is defined in the Project property shown in the title bar. The "add_executable" property tells CLion where to look for the file to run when you press the Run button in the toolbar.



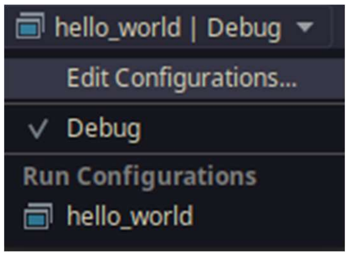
These are the run controls. Pressing Run, which is the play icon, will compile the program and execute it in the integrated terminal. Try it now.



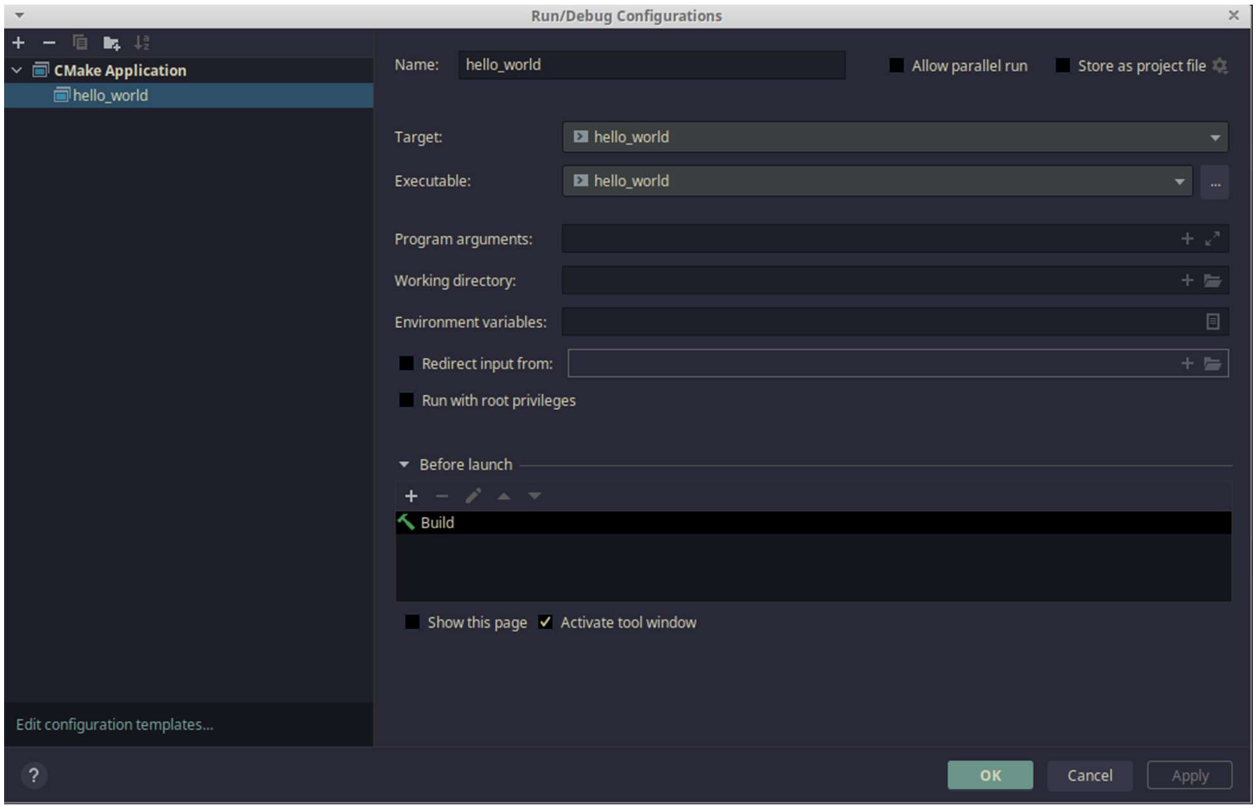
```
cmake_minimum_required(VERSION 3.23)
project(hello_world C)

set(CMAKE_C_STANDARD 99)

add_executable(hello_world main.c)
```

```
/home/interyx/dev/hello-world/cmake-build-debug/hello_world
Hello, World!

Process finished with exit code 0
```

Now we can see the program has finished executing and the output is displayed underneath, just like it would in the regular Linux terminal. However, if we need access to external files (as you will in programming assignment 2) we need to do a bit more work.



Click on the executable to bring up the Run Configurations screen, then click Edit Configurations…



Name: hello_world
Allow parallel run
Store as project file
Target: hello_world
Executable: hello_world
Program arguments:
Working directory:
Environment variables:
Redirect input from:
Run with root privileges
Before launch
Build
Show this page
[✓ Activate tool window]

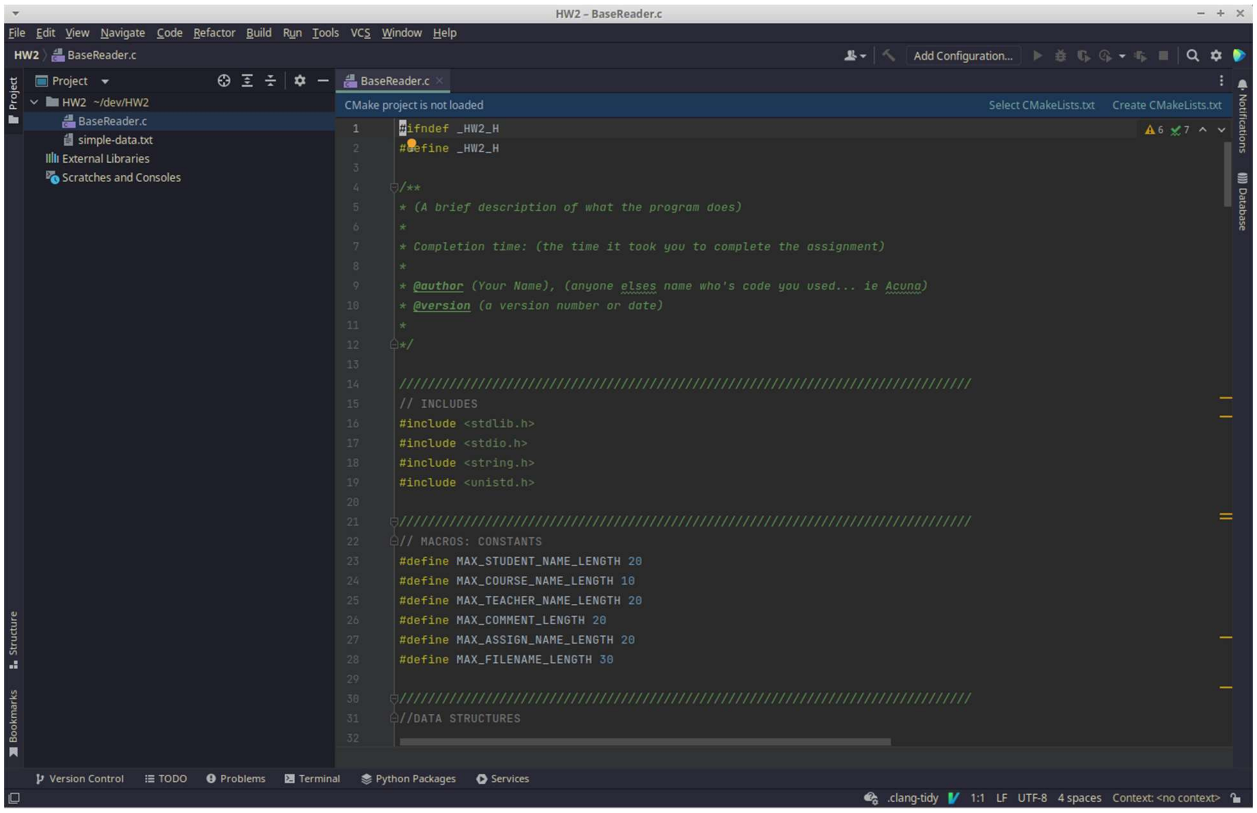
This tells CLion how to run the program. The most important pieces we'll look at are Program Arguments and Working Directory.

To pass command line arguments into your program, in the terminal you would just type them in while running it. But CLion just has a Run button, so the arguments go here. Type them as usual, like `-d simple-data.txt` But it also needs to know where to look for that file. The terminal holds the current directory as the working directory, but CLion does not make that assumption. To use a relative path you need to set the working directory to your project directory by clicking on the folder icon to browse to your project directory.

Once those two are set you should be able to run a program repeatedly with the same test data with a single click. A more advanced version of this: if you have multiple test data files, you can add the same executable multiple times in your CMakeLists and set the run configurations differently. That's a good way to challenge yourself to understand CLion better when you get to weeks 2 and 4. It will definitely come in handy.

Importing Projects

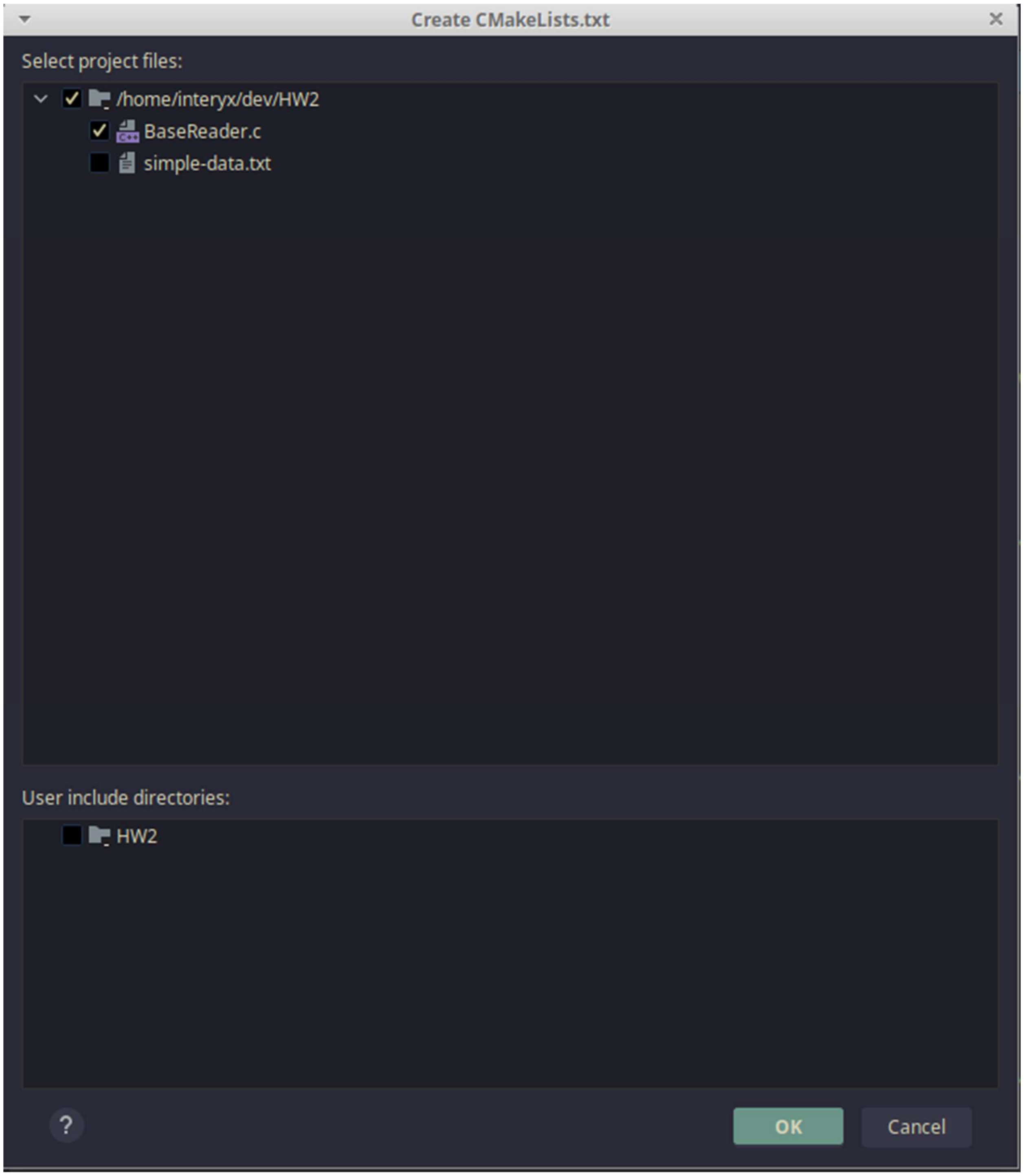
That’s good for making a new project, but how do you import given code? I’ll use Assignment 2 as a demonstration.



```
#ifndef _HW2_H
#define _HW2_H

/**
* (A brief description of what the program does)
*
* Completion time: (the time it took you to complete the assignment)
*
* @author (Your Name), (anyone elses name who's code you used... ie Acuna)
* @version (a version number or date)
*
*/

////////////////////////////////////////////////////////////////////////////////
// INCLUDES
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

////////////////////////////////////////////////////////////////////////////////
// MACROS: CONSTANTS
#define MAX_STUDENT_NAME_LENGTH 20
#define MAX_COURSE_NAME_LENGTH 10
#define MAX_TEACHER_NAME_LENGTH 20
#define MAX_COMMENT_LENGTH 20
#define MAX_ASSIGN_NAME_LENGTH 20
#define MAX_FILENAME_LENGTH 30

////////////////////////////////////////////////////////////////////////////////
//DATA STRUCTURES
```

Here I have opened “HW2” holding the assignment files for Homework 2, BaseReader.c and the data file simple-data.txt. As you can see, there is no run configuration and CLion is prompting us to either select or create a CmakeLists.txt.

We don’t have one, so we’ll select Create.



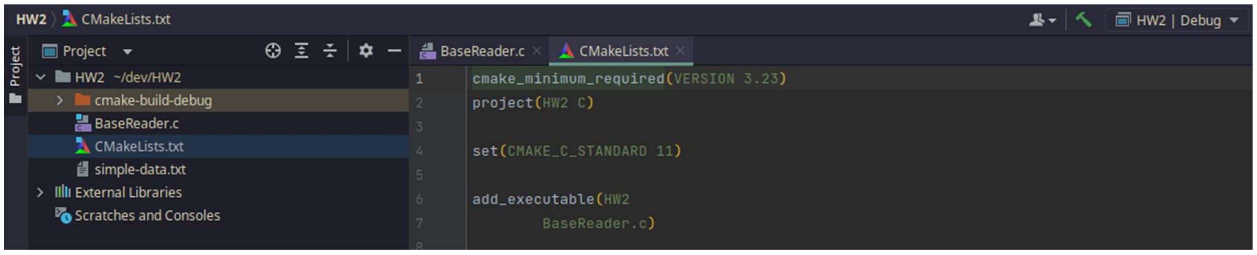
Select project files:

- [✓] /home/interyx/dev/HW2
  - [✓] BaseReader.c
  - [ ] simple-data.txt

User include directories:

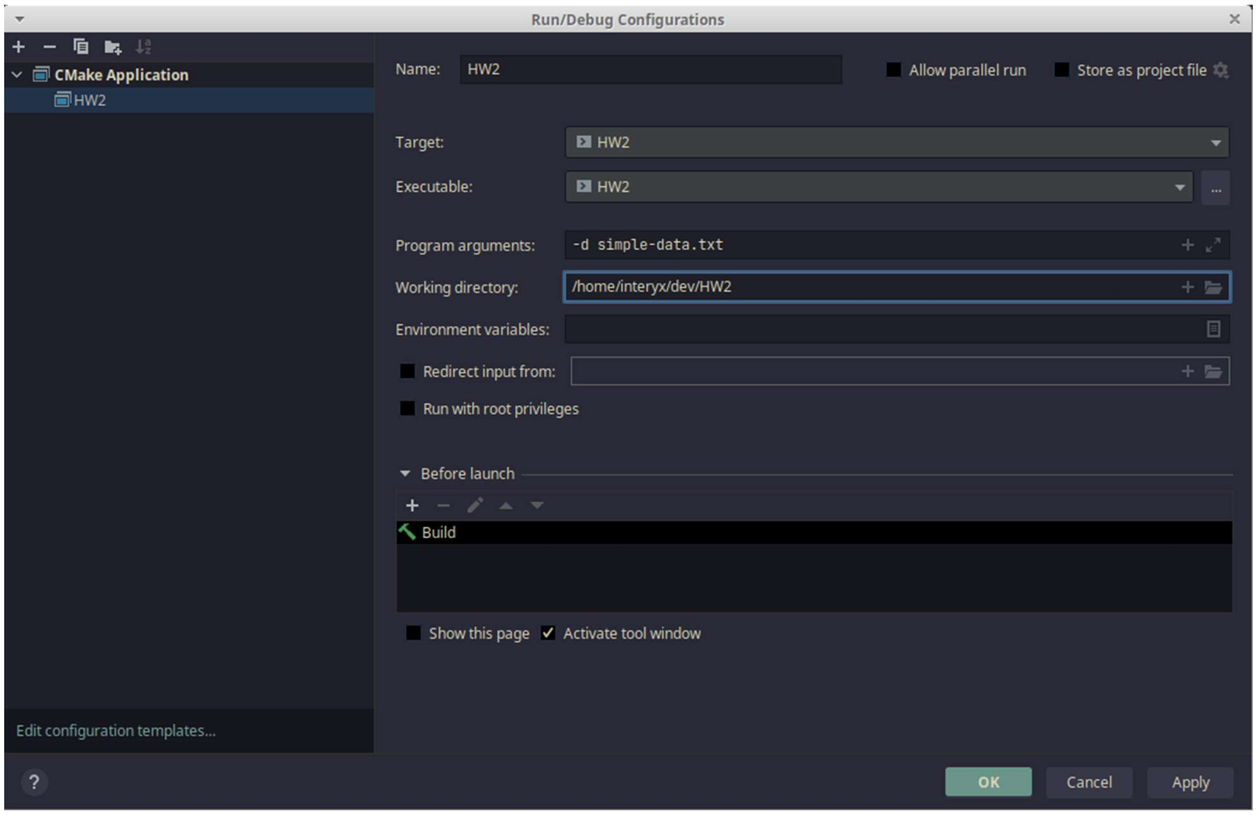
- [ ] HW2

It's detected the .c files automatically. Here you can fine-tune what files you want in your project and what CLion will recognize. We only need this one, so let’s continue.



```
cmake_minimum_required(VERSION 3.23)
project(HW2 C)

set(CMAKE_C_STANDARD 11)

add_executable(HW2
        BaseReader.c)
```

We can see that the autogenerated CMakeLists has filled in the required fields. Since "HW2" is the project name pulled from the project directory, it's been made the name of the run configuration and we can now run the program. Of course, this program needs to be run with the arguments -d simple-data.txt, and we need to find the simple-data.txt file in the same directory. As earlier, we can do this by editing the run configuration.



Name: HW2

Target: HW2

Executable: HW2

Program arguments: -d simple-data.txt

Working directory: /home/interyx/dev/HW2

Environment variables:

With this change this program will now run properly. Or at least it will when you write the code to make it understand how to process the data.

Optional Tools

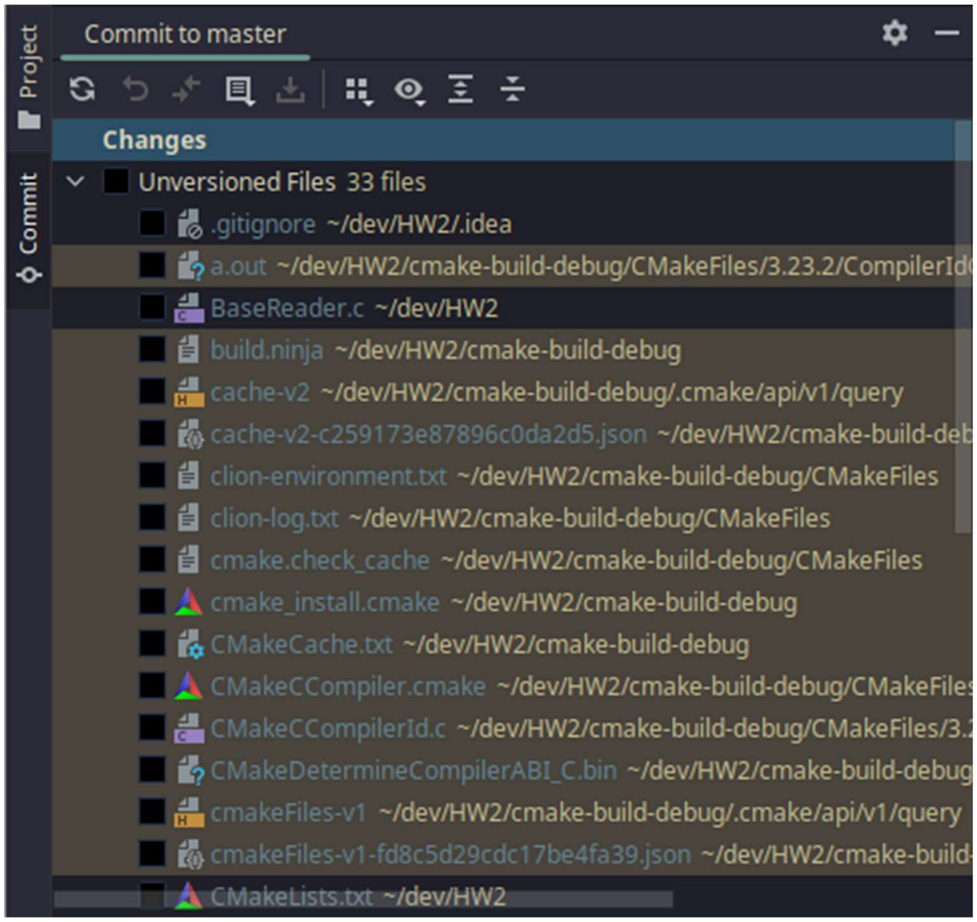
This section is optional as everything you need to run CLion has been covered already. If you want to integrate version control and GitHub integration, read on.

First, Ubuntu does not come with git preinstalled. Git is a version control system where you can commit specific states of your code at different times and move back and forth between them. It's very useful to be able to save a working version of your code and work on a new feature; if you find that you've completely broken your code (which has happened to me) it's easy to save, roll back to a previous version, and see what's different and how to fix it.

To install git, it's `sudo apt install git`. Pretty easy. CLion has version control built into it; to use it, go to VCS->Enable Version Control Integration.

A prompt will appear asking which version control system to use. We'll use Git.

Immediately all your files will turn blue in the project view. This means that they have not been added to your project in git. The Commit tab will appear in the sidebar underneath the Project tab. If we got in there, we'll see… something of a mess.



Unversioned Files 33 files
.gitignore ~/dev/HW2/.idea
a.out ~/dev/HW2/cmake-build-debug/CMakeFiles/3.23.2/CompilerId
BaseReader.c ~/dev/HW2
build.ninja ~/dev/HW2/cmake-build-debug
cache-v2 ~/dev/HW2/cmake-build-debug/.cmake/api/v1/query
cache-v2-c259173e87896c0da2d5.json ~/dev/HW2/cmake-build-deb
clion-environment.txt ~/dev/HW2/cmake-build-debug/CMakeFiles
clion-log.txt ~/dev/HW2/cmake-build-debug/CMakeFiles
cmake.check_cache ~/dev/HW2/cmake-build-debug/CMakeFiles
cmake_install.cmake ~/dev/HW2/cmake-build-debug
CMakeCache.txt ~/dev/HW2/cmake-build-debug
CMakeCCompiler.cmake ~/dev/HW2/cmake-build-debug/CMakeFiles
CMakeCCompilerId.c ~/dev/HW2/cmake-build-debug/CMakeFiles/3.2
CMakeDetermineCompilerABI_C.bin ~/dev/HW2/cmake-build-debug
cmakeFiles-v1 ~/dev/HW2/cmake-build-debug/.cmake/api/v1/query
cmakeFiles-v1-fd8c5d29cdc17be4fa39.json ~/dev/HW2/cmake-build
CMakeLists.txt ~/dev/HW2

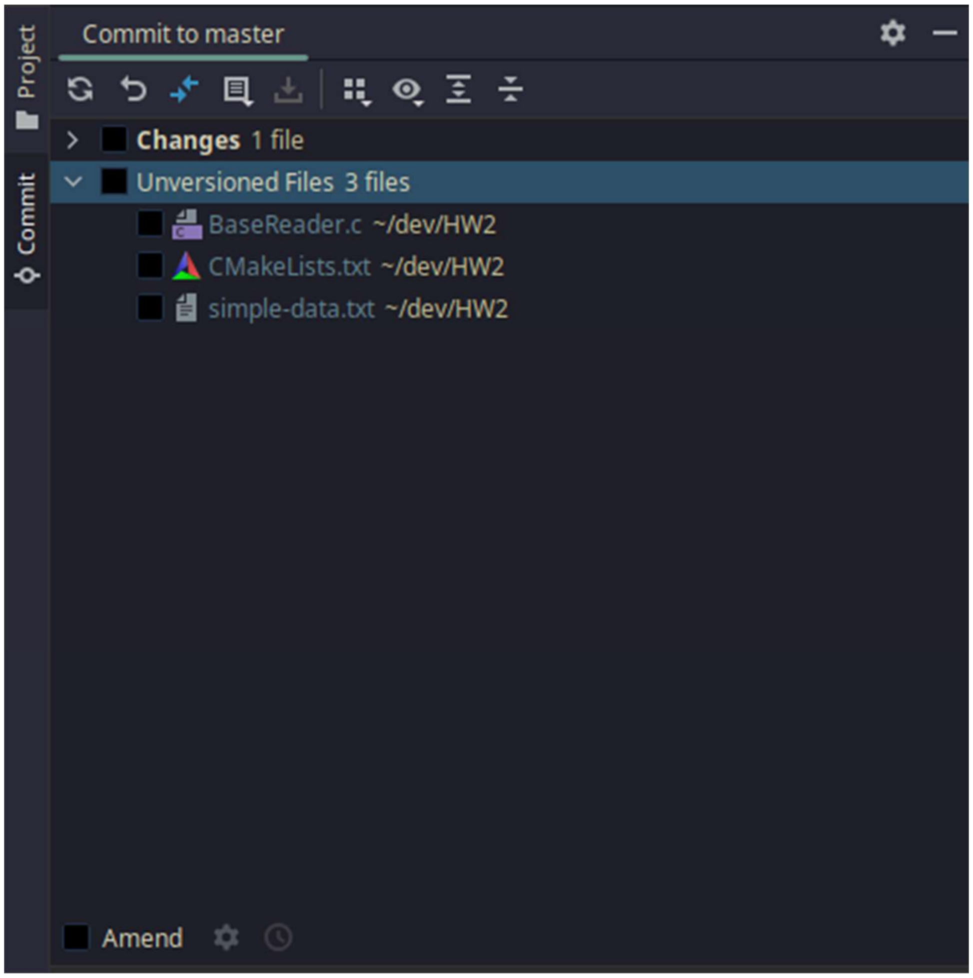
We don’t want all this mess in here. This is all local development stuff that CLion needs to run, but absolutely does not need to be in your repository. A repo is strictly for files that someone else needs to build and run the software, not project configuration files.

Create a file called .gitignore in your HW2 folder, and in it, add the following:

/cmake-build-debug/

/.idea/

These two folders contain IntelliJ project data and debug data for the CMake tools. Now in Commit, you should see in Unversioned Files the following:



Much nicer. Now we can add these files to our repository by selecting them; when files are selected the Commit Message prompt pops up along with the option to Commit and Commit and Push.

The commit message is one that describes the changes in the commit, which is a file state. It's useful if you want to go back to a specific version of your code, so make it a useful one. This one can be "initial file commit," which we can go ahead and commit. The Commit and Push button will send it to a remote repository located on, for instance, GitHub. To use that feature you'll have to have a GitHub account and login through the IDE, then you can push your code changes straight to the internet. This really helps if your virtual machine crashes or becomes unrecoverable, then you still have your latest commit to work from instead of… nothing.

If you'd prefer to do this in the terminal, you can either use CLion's integrated terminal or open a terminal window. Then the command is (assuming the git ignore file is set up`)` `git add [filenames]`, or if you just want to add all files it's `git add *`. Then to commit them it's `git commit -m <commit message>`

Note that if you forget the -m switch to add a message, git will bring up a window in vim to write the changes, and vim has something of a learning cliff. To get out of it, press the **i** key, write your commit message, then hit **Esc** and then **:wq**. A full vim tutorial is outside the scope of this writeup but this is how to get out of it the easiest.

This is very handy but you have to keep in mind: **If you use a remote repository, it must be private!** If someone finds and uses your code that's an academic integrity violation, which we take very seriously.

Using git locally has no such restrictions, though, and still can be used for local version backups. If you want to publish the project to GitHub, go to Git->GitHub->Share Project on GitHub. This will prompt you to login and create a remote repository attached to your account.

That's all for this CLion setup and tutorial. Thanks for reading, and as always don't hesitate to reach out if you have any questions!