setup

cmput415

Sep 03, 2020

CONTENTS

These instructions are for setting up your development environment for CMPUT 415. First, follow “First Steps”. Afterwards, follow the instructions for your particular OS.

CHAPTER

ONE

FIRST STEPS

1.1 JetBrains License

You need a valid JetBrains account to use CLion. Luckily, you're a student and that gets you free things. If you already have an account with a student license, you can skip this.

1. Fill out this form to start getting your student license. Select `I'm a student` then fill out the rest of the form appropriately. You **must use your UAlberta email address** in order to be approved.
2. Go to your email inbox (the automation process is typically quite quick), find the email with the subject `JetBrains Educational Pack Confirmation`. Open it and click the link `Confirm Request`.
3. You should see a new page with the header `Congrats! You've been #approved!`.
4. There should be a new email in your inbox with the subject `JetBrains Student License Confirmation`. Open it and click the link `Activate Educational License`.
5. Fill out the form to create your account. Enter your first name, last name, and username. Choose an appropriate password and accept the account agreement. It's your choice to consent to the use of your data, but it's not necessary.
6. Your account should be usable with CLion now.

CHAPTER

TWO

UBUNTU

This section details how to setup the Ubuntu development environment.

2.1 Installing build-essential

On anything except a fresh install you've almost certainly installed this package in the course of your regular development. In case you're working from a fresh install, you'll need this first:

```
$ sudo apt-get update
$ sudo apt-get install build-essential
```

2.2 Installing libUUID

LibUUID is a library required for many applications so it's possible that you've already installed this while installing something else. Best to try and install it anyways:

```
$ sudo apt-get update
$ sudo apt-get install uuid-dev
```

2.3 Installing OpenJDK JRE 8

The Java runtime environment (JRE) is required to run the ANTLR generator. OpenJDK's JRE is easier to install than Oracle's, so we'll use that.

```
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jre
```

2.4 Installing Git

Git should be installed by default in Ubuntu. If you've removed it or it is otherwise unavailable then you can install it using this command:

```
$ sudo apt-get update
$ sudo apt-get install git
```

2.5 Installing CMake

Installing CMake from the package manager is easy too:

```
$ sudo apt-get update
$ sudo apt-get install cmake
```

2.6 ANTLR 4 C++ Runtime

This section details how to install the ANTLR 4 C++ runtime on Ubuntu assuming your default shell is bash. If you've changed your shell from bash it's assumed that you are familiar enough with your environment that you can modify these steps appropriately.

1. To make things easy, we are going to do everything inside a new directory in your home directory.

   ```
   $ mkdir $HOME/antlr
   ```

   We'll refer to this directory (`$HOME/antlr`) as `ANTLR_PARENT`.

2. Next we need to clone the runtime source from GitHub:

   ```
   $ cd <ANTLR_PARENT>
   $ git clone https://github.com/antlr/antlr4.git
   ```

   This should create a new folder called `antlr4` in `ANTLR_PARENT`. We'll refer to this new directory (`<ANTLR_PARENT>/antlr4`) as `SRC_DIR`.

3. We will be using ANTLR 4.8 so we need to change to the git tag for version 4.8.

   ```
   $ cd <SRC_DIR>
   $ git checkout 4.8
   ```

   This will give you a warning about being in a "detached head state". Since we won't be changing anything in ANTLR there is no need to create a branch. No extra work is needed here.

4. Now we need a place to build the runtime. CMake suggests making your build directory inside your source directory.

   ```
   $ cd <SRC_DIR>
   $ mkdir antlr4-build
   ```

   We'll refer to this new directory (`<SRC_DIR>/antlr4-build`) as `BUILD_DIR`.

5. We need to have an install directory prepared before building since it's referenced in the build step. This directory will have the headers and compiled ANTLR libraries put into it. To make the actual directory:

```
$ cd <ANTLR_PARENT>
$ mkdir antlr4-install
```

We'll refer to this new directory (`<ANTLR_PARENT>/antlr4-install`) as `INSTALL_DIR`.

Before continuing, if you're following this guide exactly, confirm your directory structure looks like this:

```
$HOME
+-- antlr/
    +-- antlr4/
    |   +-- antlr4-build/
    +-- antlr4-install/
```

6. Finally, we're ready to start the actual build process. Let's begin by doing the generate and configure CMake step for the runtime. We need to do this while inside the build directory. As well, we need to tell it that we want a release build and to install it to a certain directory.

```
$ cd <BUILD_DIR>
$ cmake <SRC_DIR>/runtime/Cpp/ \
    -DCMAKE_BUILD_TYPE=RELEASE \
    -DCMAKE_INSTALL_PREFIX="<INSTALL_DIR>"
```

You will be presented with some CMake warnings but they're safe to ignore.

7. We can finally run `make` to build the library and install it. You can make the process significantly faster by running with multiple threads using the `-j` option and specifying a thread count. Using the option without a count will use unlimited threads. Be careful when using unlimited threads, the build has failed in the past due to limited resources. This isn't a big issue for the build because you can always just try again with a limited number of threads but your computer may appear to hang due to being over capacity.

```
$ make install -j<number of threads>
```

8. Now we can add the install to your bashrc. Pick your favorite text editor, open `~/.bashrc`, and add the following lines to the end, substituting appropriately:

```
# C415 ANTLR install
export ANTLR_INS="<INSTALL_DIR>"
```

Make sure there is no trailing /. Close and reopen your terminal for things to take effect.

2.7 Installing CLion

1. Go to the download page and download *CLion* for Linux.
2. Assuming you've downloaded the tarball to your `~/Downloads` folder, you can extract it to `/opt/` using the following command:

```
$ sudo tar -xzf ~/Downloads/CLion-<version>.tar.gz -C /opt/
```

If you are confident about your ability to setup your own install you can put it elsewhere but you will be on your own.

3. Execute the installer:

```
$ /opt/CLion-<version>/bin/clion.sh
```

4. Perform the initial set up of CLion.
    1. Select `Do not import settings` and click `OK`.
    2. Scroll to the bottom of the license agreement then hit `Accept`.
    3. Choose if you want to share usage statistics.
    4. You should be presented with a prompt for your license. Select `Activate`, `JetBrains Account`, enter your UAlberta email address and JetBrains password. Click the `Activate` button.
    5. Pick your favorite UI. Then click `Next: Toolchains`.
    6. CLion bundles a version of CMake with it. If you'd prefer to use the one we've just installed change `Bundled` to `/usr/bin/cmake`. The info text beneath should update with a checkmark and the version of your installed cmake. Click `Next: Default Plugins`.
    7. You might consider disabling all but the git plugin, and even then, using it is up to you. It can be useful to see the color coded files for differences at a glance or track changes in a file. You should consider disabling all of the web development plugins. Disabling other tools is up to you as well. Now select `Next: Feature Plugins`
    8. Again, the choices here are yours. If you like vim, then maybe the vim plugin is up your alley. The markdown plugin can be useful as well. You do not need the TeamCity Integration, the Lua integration, nor the Swift integration. Select `Start using CLion`

2.8 Installing the ANTLR Plugin for CLion

ANTLR has a CLion integration that gives syntax highlighting as well as tool for visualising the parse tree for a grammar rule and an input.

1. Launch CLion by going to the application launcher (tap the super/Windows button) and typing `clion`. This should launch CLion.
2. Open the settings window `CLion` → `Preferences...`
3. Select `Plugins` from the menu on the left.
4. Click `Browse Repositories...` below the plugin list.
5. In the new window, type `antlr` into the search bar at the top.
6. From the list select `ANTLR v4 grammar plugin`.
7. Click `Install` in the right pane and accept the notice.
8. After the install bar ends click the `Restart CLion` button that should have replaced the `Install` button.

2.9 Installing ANTLR Generator

If you'd like to manually generate a listener or visitor you need to have the ANTLR generator. Follow these steps into install it:

1. Make the destination directory. I would suggest putting this in `<INSTALL_DIR>/bin` since the assignments will already automatically download a copy there and duplicating this seems wasteful. If you want to put it elsewhere though, you can.

```
$ mkdir <INSTALL_DIR>/bin
```

We'll refer to this new directory (e.g. `<INSTALL_DIR>/bin`) as `ANTLR_BIN`.

2. Next, download the tool.

```
$ curl https://www.antlr.org/download/antlr-4.8-complete.jar \
    -o <ANTLR_BIN>/antlr-4.8-complete.jar
```

3. Now we can make it easy to use. Add the following lines to your `~/.bashrc`:

```
# C415 ANTLR generator.
export CLASSPATH="<ANTLR_BIN>/antlr-4.8-complete.jar:$CLASSPATH"
alias antlr4="java -Xmx500M org.antlr.v4.Tool"
alias grun='java org.antlr.v4.gui.TestRig'
```

4. Close and reopen your terminal for things to take effect. Now these commands should produce useful help outputs:

```
$ antlr4
$ grun
```

2.10 Installing the Tester

This is the tool you'll be using for testing your solutions locally. You'll be building it yourself so that any changes later are easily obtainable.

If you encounter issues, please log them on the GitHub issue tracker or, if you want to, submit a pull request and we'll review it!

1. We'll build the tool in your home directory.

```
$ cd $HOME
$ git clone https://github.com/cmput415/Tester.git
```

2. Next we'll make the build directory.

```
$ cd Tester
$ mkdir build
```

3. Now, the configure and generate step.

```
$ cd build
$ cmake ..
```

4. Finally, build the project.

```
$ make
```

5. We could refer directly to the executable every time, but it's probably easier to just have it on our path. Add these lines to the end of `~/.bashrc`.

```
# C415 testing utility.
export PATH="$HOME/Tester/bin/:$PATH"
```

6. Close and reopen your terminal to have changes take effect. Test the command to make sure it works.

```
$ tester --help
```

For more info about organising your tests and creating a configuration (though templates will be provided with your assignments) you can check the Tester README.

2.11 Testing Your Environment

Everything should be setup! Let's just make sure.

1. Download this tarball.
2. Extract it via

```
$ tar -xzf demo.tar.gz
```

3. Change into the extracted directory.

```
$ cd demo
```

4. Make the project.

```
$ make
```

5. The project should compile with no warnings or errors. If there's a problem, you may have set something up incorrectly. Otherwise, congrats!
6. If you'd like to start playing with the tools this is a good opportunity! Here are a few challenges you can attempt with the files provided:
    1. The tool is asking for an input file. Examine the grammar and C++ source and figure out how to construct an appropriate input where ANTLR doesn't complain about extra tokens.
    2. Add floats.
        - Be careful of lexer rule ordering.
        - Be careful that things like `6|5` or `6a5` are not recognised as floats.

2.12 Creating a Personal Project

We're providing two ways for you to play with ANTLR and C++. The first way uses the Makefile from the demo you've just done, and the other uses CMake to set up a project using the CMake modules that are also used by your assgnments.

2.12.1 Makefile

First, download the Makefile from the link and put it in your folder. Alternatively you can download straight to your directory:

```
$ curl https://cmput415.github.io/415-docs/setup/_static/Makefile -o Makefile
```

This Makefile is both rather complex and simple. The internals are the complicated part. If you'd like to understand how the Makefile works then everything is well documented. However, that complexity makes using it simple! So if you'd prefer to just use the Makefile then we can keep everything simple.

First things first, your grammars. All grammars need to be in the same directory as the Makefile. If they aren't, then they won't be detected, generated, built, or linked.

Next, your source files (`.cpp` or `.h(pp)`) must also be in the same directory as the Makefile. Again, if they aren't, they won't be detected, built, or linked.

As you can see, this isn't the most scalable of directory structures but it is functional for playing with ANTLR and C++. To test that it's working, create your grammar file with:

```
grammar <file_name>;
<top_rule>: ANYTHING*? EOF;
ANYTHING: .;
```

And the file that has your main in it:

```
#include "<grammar_name>Parser.h"
int main() { return 0; }
```

You should be able to make it and run the tool (it won't produce any output):

```
$ make
$ ./tool
```

We've also enabled you to use the ANTLR GUI through the Makefile. First, make an input file. Then, pass it to the Makefile 'gui' rule:

```
$ echo "this is a test" > test.txt
$ make gui grammar=<grammar_name> rule=<top_rule> file=test.txt
```

Any grammar in the same directory as the make file can be used in this fashion (including the `.g4` extension is optional). The `rule` can be any rule in the grammar, but usually it makes sense to test your "top level" rule. If the `file` option is not included then the GUI will take input from stdin to parse (type into your terminal). Terminate your input with EOF (ctrl+d on linux generally).

You're ready to start modifying the grammar and C++ source. Don't be afraid to add new source files and header files: style will eventually be part of your mark so starting here is a good idea! Feel free to cannibalise anything you'd like from the demo files.

2.12.2 CMake

Todo: WIP

A CMake setup is possible for a better scaling setup but hasn't been prepared for individual project consumption outside of assignments.

CHAPTER

THREE

MAC OS

This section details how to setup the Mac OS development environment.

3.1 Installing Developer tools

It's likely that you've already done this since you're take a high level CS class, but in the event that you have a fresh install, run the following command to install Mac OS developer tools:

```
$ xcode-select --install
```

3.2 Installing Homebrew

Homebrew is a package manager for Mac OS. If you don't have it yet, you can read about it here. Otherwise, install it via the command on their front page:

```
$ /usr/bin/ruby -e "$(curl -fsSL \
    https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

You can check that it succeeded by checking its version:

```
$ brew --version
```

Homebrew itself is not a requirement, just an easy suggestion, but a package manager is. Using another package manager (like Nix) is fine, as long as you understand how to install packages.

3.3 Installing Oracle Java JDK

Installing from the Oracle download page requires a bunch of extra set up when instead you could just use our good friend Homebrew to install it (unfortunately you get the JDK not just the JRE).

```
$ brew cask install java
```

3.4 Installing Git

The Apple managed version of git should have been installed with your developer tools, you can test this by checking the version.

```
$ git --version
```

(OPTIONAL) If you want a more recent version, you can install one through brew (or your favorite package manager). At the time of writing this, the versions only differ by two minor versions, so the difference is not significant.

```
$ brew install git
```

3.5 Installing CMake

Brew (or otherwise) makes this easy:

```
$ brew install cmake
```

3.6 ANTLR 4 C++ Runtime

This section details how to install the ANTLR 4 C++ runtime on Mac OS assuming your default shell is bash. If you've changed your shell from bash it's assumed that you are familiar enough with your environment that you can modify these steps appropriately.

1. To make things easy, we are going to do everything inside a new directory in your home directory.

   ```
   $ mkdir $HOME/antlr
   ```

   We'll refer to this directory (`$HOME/antlr`) as `ANTLR_PARENT`.

2. Next we need to clone the runtime source from GitHub:

   ```
   $ cd <ANTLR_PARENT>
   $ git clone https://github.com/antlr/antlr4.git
   ```

   This should create a new folder called `antlr4` in `ANTLR_PARENT`. We'll refer to this new directory (`<ANTLR_PARENT>/antlr4`) as `SRC_DIR`.

3. We will be using ANTLR 4.8 so we need to change to the git tag for version 4.8.

   ```
   $ cd <SRC_DIR>
   $ git checkout 4.8
   ```

   This will give you a warning about being in a "detached head state". Since we won't be changing anything in ANTLR there is no need to create a branch. No extra work is needed here.

4. Now we need a place to build the runtime. CMake suggests making your build directory inside your source directory.

   ```
   $ cd <SRC_DIR>
   $ mkdir antlr4-build
   ```

   We'll refer to this new directory (`<SRC_DIR>/antlr4-build`) as `BUILD_DIR`.

5. We need to have an install directory prepared before building since it's referenced in the build step. This directory will have the headers and compiled ANTLR libraries put into it. To make the actual directory:

```
$ cd <ANTLR_PARENT>
$ mkdir antlr4-install
```

   We'll refer to this new directory (`<ANTLR_PARENT>/antlr4-install`) as `INSTALL_DIR`.

   Before continuing, if you're following this guide exactly, confirm your directory structure looks like this:

```
$HOME
+-- antlr/
    +-- antlr4/
    |   +-- antlr4-build/
    +-- antlr4-install/
```

6. Finally, we're ready to start the actual build process. Let's begin by doing the generate and configure CMake step for the runtime. We need to do this while inside the build directory. As well, we need to tell it that we want a release build and to install it to a certain directory.

```
$ cd <BUILD_DIR>
$ cmake <SRC_DIR>/runtime/Cpp/ \
    -DCMAKE_BUILD_TYPE=RELEASE \
    -DCMAKE_INSTALL_PREFIX="<INSTALL_DIR>"
```

   You will be presented with some CMake warnings but they're safe to ignore.

7. We can finally run `make` to build the library and install it. You can make the process significantly faster by running with multiple threads using the `-j` option and specifying a thread count. Using the option without a count will use unlimited threads. Be careful when using unlimited threads, the build has failed in the past due to limited resources. This isn't a big issue for the build because you can always just try again with a limited number of threads but your computer may appear to hang due to being over capacity.

```
$ make install -j<number of threads>
```

8. Now we can add the install to your bash profile. Pick your favorite text editor, open `~/.bash_profile`, and add the following lines to the end, substituting appropriately:

```
# C415 ANTLR install
export ANTLR_INS="<INSTALL_DIR>"
```

   Make sure there is no trailing /. Close and reopen your terminal for things to take effect.

3.7 Installing CLion

1. Use Homebrew to install CLion:

```
$ brew cask install clion
```

2. Open CLion (via spotlight: command+space → type `CLion`).
3. Perform the initial set up of CLion.
   1. Select `Do not import settings` and click `OK`.
   2. Scroll to the bottom of the license agreement then hit `Accept`.
   3. Choose if you want to share usage statistics.

4. You should be presented with a prompt for your license. Select `Activate`, `JetBrains Account`, enter your UAlberta email address and JetBrains password. Click the `Activate` button.
5. Pick your favorite UI. Then click `Next: Toolchains`.
6. CLion bundles a version of CMake with it. If you'd prefer to use the one we've just installed change `Bundled` to `/usr/local/bin/cmake`. The info text beneath should update with a checkmark and the version of your installed cmake. Click `Next: Default Plugins`.
7. You might consider disabling all but the git plugin, and even then, using it is up to you. It can be useful to see the color coded files for differences at a glance or track changes in a file. You should consider disabling all of the web development plugins. Disabling other tools is up to you as well. Now select `Next: Feature Plugins`
8. Again, the choices here are yours. If you like vim, then maybe the vim plugin is up your alley. The markdown plugin can be useful as well. You do not need the TeamCity Integration, the Lua integration, nor the Swift integration. Select `Start using CLion`

3.8 Installing the ANTLR Plugin for CLion

ANTLR has a CLion integration that gives syntax highlighting as well as tool for visualising the parse tree for a grammar rule and an input.

1. Launch CLion by going to the application launcher (finder) and typing `clion`. This should launch CLion.
2. Open the settings window `CLion` → `Preferences...`
3. Select `Plugins` from the menu on the left.
4. Click `Browse Repositories...` below the plugin list.
5. In the new window, type `antlr` into the search bar at the top.
6. From the list select `ANTLR v4 grammar plugin`.
7. Click `Install` in the right pane and accept the notice.
8. After the install bar ends click the `Restart CLion` button that should have replaced the `Install` button.

3.9 Installing ANTLR Generator

If you'd like to manually generate a listener or visitor you need to have the ANTLR generator. Follow these steps into install it:

1. Make the destination directory. I would suggest putting this in `<INSTALL_DIR>/bin` since the assignments will already automatically download a copy there and duplicating this seems wasteful. If you want to put it elsewhere though, you can.

   ```
   $ mkdir <INSTALL_DIR>/bin
   ```

   We'll refer to this new directory (e.g. `<INSTALL_DIR>/bin`) as `ANTLR_BIN`.

2. Next, download the tool.

   ```
   $ curl https://www.antlr.org/download/antlr-4.8-complete.jar \
       -o <ANTLR_BIN>/antlr-4.8-complete.jar
   ```

3. Now we can make it easy to use. Add the following lines to your `~/.bash_profile`:

```
# C415 ANTLR generator.
export CLASSPATH="<ANTLR_BIN>/antlr-4.8-complete.jar:$CLASSPATH"
alias antlr4="java -Xmx500M org.antlr.v4.Tool"
alias grun='java org.antlr.v4.gui.TestRig'
```

4. Close and reopen your terminal for things to take effect. Now these commands should produce useful help outputs:

```
$ antlr4
$ grun
```

3.10 Installing the Tester

This is the tool you'll be using for testing your solutions locally. You'll be building it yourself so that any changes later are easily obtainable.

If you encounter issues, please log them on the GitHub issue tracker or, if you want to, submit a pull request and we'll review it!

1. The testing tool uses C++17 features (that have been experimental since C++11) that the default clang installation doesn't ship with by default. While it *is* possible to build with Clang, the process and invocation is much more involved. Why stress ourselves when GCC can save us the trouble?

```
$ brew install gcc@10
```

2. We'll build the tool in your home directory.

```
$ cd $HOME
$ git clone https://github.com/cmput415/Tester.git
```

3. Next we'll make the build directory.

```
$ cd Tester
$ mkdir build
```

4. Now, the configure and generate step.

```
# cd build
# cmake .. -DCMAKE_CXX_COMPILER="g++-10" -DCMAKE_C_COMPILER="gcc-10"
```

   The flags on the end ensure we're using GCC to compile this.

5. Finally, build the project.

```
$ make
```

6. We could refer directly to the executable every time, but it's probably easier to just have it on our path. Add these lines to the end of `~/.bash_profile`.

```
# C415 testing utility.
export PATH="$HOME/Tester/bin/:$PATH"
```

7. Close and reopen your terminal to have changes take effect. Test the command to make sure it works.

```
$ tester --help
```

For more info about organising your tests and creating a configuration (though templates will be provided with your assignments) you can check the Tester README.

3.11 Testing Your Environment

Everything should be setup! Let's just make sure.

1. Download this tarball.
2. Extract it via

```
$ tar -xzf demo.tar.gz
```

3. Change into the extracted directory.

```
$ cd demo
```

4. Make the project.

```
$ make
```

5. The project should compile with no warnings or errors. If there's a problem, you may have set something up incorrectly. Otherwise, congrats!
6. If you'd like to start playing with the tools this is a good opportunity! Here are a few challenges you can attempt with the files provided:
    1. The tool is asking for an input file. Examine the grammar and C++ source and figure out how to construct an appropriate input where ANTLR doesn't complain about extra tokens.
    2. Add floats.
        - Be careful of lexer rule ordering.
        - Be careful that things like `6|5` or `6a5` are not recognised as floats.

3.12 Creating a Personal Project

We're providing two ways for you to play with ANTLR and C++. The first way uses the Makefile from the demo you've just done, and the other uses CMake to set up a project using the CMake modules that are also used by your assgnments.

3.12.1 Makefile

First, download the Makefile from the link and put it in your folder. Alternatively you can download straight to your directory:

```
$ curl https://cmput415.github.io/415-docs/setup/_static/Makefile -o Makefile
```

This Makefile is both rather complex and simple. The internals are the complicated part. If you'd like to understand how the Makefile works then everything is well documented. However, that complexity makes using it simple! So if you'd prefer to just use the Makefile then we can keep everything simple.

First things first, your grammars. All grammars need to be in the same directory as the Makefile. If they aren't, then they won't be detected, generated, built, or linked.

Next, your source files (`.cpp` or `.h(pp)`) must also be in the same directory as the Makefile. Again, if they aren't, they won't be detected, built, or linked.

As you can see, this isn't the most scalable of directory structures but it is functional for playing with ANTLR and C++. To test that it's working, create your grammar file with:

```
grammar <file_name>;
<top_rule>: ANYTHING*? EOF;
ANYTHING: .;
```

And the file that has your main in it:

```
#include "<grammar_name>Parser.h"
int main() { return 0; }
```

You should be able to make it and run the tool (it won't produce any output):

```
$ make
$ ./tool
```

We've also enabled you to use the ANTLR GUI through the Makefile. First, make an input file. Then, pass it to the Makefile 'gui' rule:

```
$ echo "this is a test" > test.txt
$ make gui grammar=<grammar_name> rule=<top_rule> file=test.txt
```

Any grammar in the same directory as the make file can be used in this fashion (including the `.g4` extension is optional). The `rule` can be any rule in the grammar, but usually it makes sense to test your "top level" rule. If the `file` option is not included then the GUI will take input from stdin to parse (type into your terminal). Terminate your input with EOF (ctrl+d on linux generally).

You're ready to start modifying the grammar and C++ source. Don't be afraid to add new source files and header files: style will eventually be part of your mark so starting here is a good idea! Feel free to cannibalise anything you'd like from the demo files.

3.12.2 CMake

Todo: WIP

A CMake setup is possible for a better scaling setup but hasn't been prepared for individual project consumption outside of assignments.

CHAPTER

FOUR

WINDOWS

Todo: WIP

Windows is generally much more difficult to use than either Mac OS X or Linux when it comes to tooling for this course and therefore hasn't been used in the past. If great demand occurs then this section may be expanded.

CHAPTER

FIVE

CS COMPUTERS

This section details how to setup your environment to use the resources already present on the CSC lab computers.

5.1 Using Prebuilt Resources

Set up on the CSC machines is a lot simpler because all of the resources are managed for you. Therefore, all you need to do is to add the provided definitions to your `~/.bashrc`.

```
# C415 Predefinitions
source "/cshome/c415/415-resources/415env.sh"
```

This should enable you to build manually using the command line.

5.2 Installing CLion

1. Go to the download page and download *CLion* for Linux.
2. Assuming you've downloaded the tarball to your `~/Downloads` folder, you can extract it to your home directory using the following command:

   ```
   $ sudo tar -xzf ~/Downloads/CLion-<version>.tar.gz -C ~
   ```

   If you are confident about your ability to setup your own install you can put it elsewhere but you will be on your own.
3. From now on, you can start *CLion* by using the following command:

   ```
   $ ~/CLion-<version>/bin/clion.sh
   ```

4. Perform the initial set up of CLion.
   1. Select `Do not import settings` and click `OK`.
   2. Scroll to the bottom of the license agreement then hit `Accept`.
   3. Choose if you want to share usage statistics.
   4. You should be presented with a prompt for your license. Select `Activate`, `JetBrains Account`, enter your UAlberta email address and JetBrains password. Click the `Activate` button.
   5. Pick your favorite UI. Then click `Next: Toolchains`.
   6. Click `Next: Default Plugins`.

7. You might consider disabling all but the git plugin, and even then, using it is up to you. It can be useful to see the color coded files for differences at a glance or track changes in a file. You should consider disabling all of the web development plugins. Disabling other tools is up to you as well. Now select `Next: Feature Plugins`
8. Again, the choices here are yours. If you like vim, then maybe the vim plugin is up your alley. The markdown plugin can be useful as well. You do not need the TeamCity Integration, the Lua integration, nor the Swift integration. Select `Start using CLion`