CST0006 – Computer Programming Foundations

INTRODUCTION to WEEK02

In this lab section we are going to learn how to use two different IDEs, and two different Git GUI clients. Make sure you follow the steps EXACTLY as you see them, otherwise you're going to get so lost, even Indiana Jones won't find you.

IDEs – Integrated Development Environments

An Integrated Development Environment is a collection of software development tools, integrated into one application that programmers can use to aid them in developing software.

We are going to test some of the main features of Thonny and Code::Blocks. Features like syntax highlighting, compiling, linking, building and running the code.

Before we begin though, we are going to use Git to track the changes made to our software projects, so we need to create two separate projects on Gitlab, then we will download the repositories locally.

Login to Gitlab, create two projects:

- One is for the C programming Project: 2018W_CST0006_LAB02_C
- One is for the Python Project: 2018W_CST0006_LAB02_Python

Once you've created those two projects, open Git Bash and follow the steps that we did last time to download the repository locally.

Remember to make a new Lab 02 folder to hold both of these projects:

<home>/git/2018W/2018W_CST0006_LAB02

You will need to type following:

Open Git Bash and Change to the working term directory

```
cd git/2018W
```

Make a new lab directory for the current week

```
mkdir 2018W_CST0006_LAB02
```

Change into Directory for the current week

```
cd 2018W_CST0006_LAB02
```

Download the Gitlab repositories Locally

Now you're ready to clone your two git repositories. Follow the steps given on Gitlab, under the "Create a new repository" command line instructions.

WARNING! Don't forget to change back to the main lab folder after you've cloned the first directory, otherwise you will be cloning the second project into the first.

Here are the two possible ways to change back to the main lab folder:

Move up one directory

```
cd ..
```

...or just type the full path name to get back to were you need to be

```
cd ~/git/2018W/2018W_CST0006_LAB02
```

Seriously, if you don't do this step, the rest of the lab won't work and you'll get all kinds of errors.

Code::Blocks

In this section we are going to create a Code::Blocks project. We are starting with Code::Blocks because they have the concept of a project, that most IDEs don't have. This can be confusing to a beginner, so you will need to pay attention to what is going on. Once you get used to this way of building a project, you'll wonder why other IDEs don't do it this way.

1. Open Code::Blocks
    - Once you open Code::Blocks, if you're using Windows, you will be shown a default compilers screen. If GNU GCC Compiler is not the current default compiler, highlight it and click the "Set as default" button. Then click the "OK" button.
    - The next option screen that will pop up is the File Associations window. If you don't have any other IDE that creates C programming files, it's OK to choose "Yes, associate Code::Blocks with C/C++ file types". Click "OK"
2. Click "Create a new project"
    - Once you click "Create a new project" a project template window will open. You will want to choose "Console application", click the "Go" button
    - Leave the "Skip this page next time" box **unchecked** click the "Next" button.
    - Choose "C", click the "Next" button.
    - Choose "Folder to create project in": <home>/git/2018W/2018_CST0006_LAB02
    - For the Project title: 2018W_CST0006_LAB02_C

WARNING! Double check that you've spelled the project title correctly, the Project title should be the exact same name (case sensitive) as your git repository that exists within the lab folder. Otherwise the rest of this lab won't work the way I've laid it out.

- Click the "Next" button if you're sure that these two names are the same.
- Leave this next page set to the defaults and click the "Finish" button.

At this point you should be able to enter these two commands in bash

```
cd ~/git/2018W/2018W_CST0006_LAB02/2018W_CST0006_LAB02_C
ls -l
```

So that you get this output...

```
-rw-rw-r--. 1 cory cory 1102 Feb  2 01:43 2018W_CST0006_LAB02_C.cbp
-rw-r--r--. 1 cory cory   99 Feb  2 01:43 main.c
-rw-rw-r--. 1 cory cory    0 Feb  2 01:40 README.md
```

3. If you see anything other than this output, you did it wrong and you will need to do it over again. Otherwise, Congratulations! You will now have a new Code::Blocks project, if you look to the left panel, you will see a Sources folder. Within that folder, you will see a file called "main.c" that Code::Blocks automatically generated.
   - Double click the "main.c" file to open it.
4. You will now have a real C program that you can play with. We are going to compile this program and run it. We will do it separately first, then do it in one step.
   - Click the yellow "cog" button in the menu bar. This will build your project. This step is what is the compiling and linking stages in one convenient button. In short, it makes your source code into an executable program. We will learn more about this later in the course.
   - Next, click the green "play" button.
   - Congratulations! You should see a pop up window with your print out.
5. Editing the source code.
   - Go back to your C program, and within the quotation marks, directly after the \n (no space), type My name is <*your name*>. Note that you need to have everything you type inside the quotation marks.
6. Building and Running
   - Next, you will build and run by clicking the button with the yellow and green symbols.
   - This should build and run your program. If you get any errors, read the error messages and fix whatever the issue was.

Take a screenshot of this running program, you will be submitting it for marks.

Thonny

In this section, we are going to create a Python project. You will see a huge difference with creating a project in Thonny than what happened in Code::Blocks. There is no concept of a project within Thonny, so we have to manually create one. Which mainly consists of creating a project folder and saving any of our source files within that "project" folder. Since we already have a local git repository, we will use that as our project folder.

1. Open Thonny. If you have any other files open, close them so we can start clean.
2. Create a new file and then save it as test.py in your 2018W_CST0006_LAB02_Python folder
3. Add the following text to this file:

```
print("Hello, world")
```

4. Save this file again and run it so that the output of the source code runs in the Shell.
5. Now copy the code from #3 into the shell and press enter.
6. You should see that the shell acts as both an interpreter and as an interactive shell.
7. Using the command line (Git Bash) stage and commit the changes to this repository, then push the changes you made to Gitlab.

Take a screenshot of your successful push to Gitlab, you'll be submitting it for marks

GIT GUI Clients

We're going to be using two different Git GUIs so that we can see a visual representation of what our git repositories look like.

GitAhead

We are going to download and install GitAhead, so you can see what a simple and clean Git GUI looks like. We will contrast this with GitKraken

Download GitAhead here: https://scitools.com/gitahead/download

When installing just choose all the defaults,

After installing, launch GitAhead and choose your preferred theme.

Open an existing repository

Choose the 2018W_CST0006_LAB02_C project

Leave this window open, and go to the next step

GitKraken

Download and install GitAhead so you can see the contrast between two Git GUIs looks like. GitKracken has a lot more features but isn't as simple as GitAhead

Download GitKraken here: https://www.gitkraken.com/download

Choose all the defaults when installing

After launching GitKraken create an account or login with a Github account if you created one.

Open a respository

Choose the 2018W_CST0006_LAB02_C project

Next steps

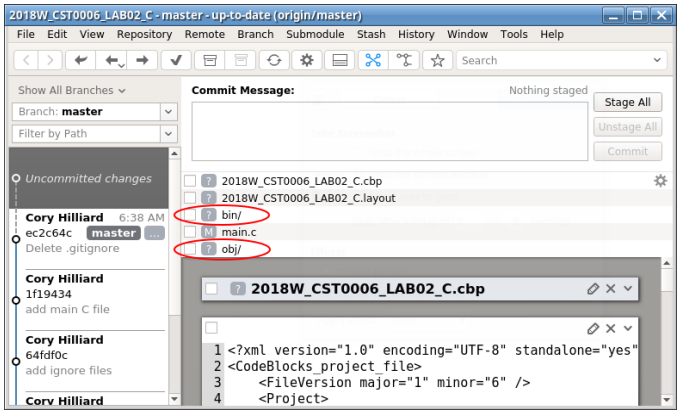
You should now have both GitAhead and GitKraken managing the same repository.

Within this repository you will see 4 untracked items (2 folders and 2 files).

Since the two folders are used for building programs, we don't need to upload or track the contents of them. We only really want to manage the changes to the code, so these to folders should be ignored.

Git has some ignore functionality built into it. So in GitAhead, right-click these two folders and choose ignore.

Here is an example of the two folders you need to ignore in GitAhead:



You should see a new file now called .gitignore, and you will see that .gitignore now has two new lines entered on it with the exact names of the folders that we told it to ignore.

Next, check the .gitignore checkbox, this is how you stage a file. Write a commit message that is descriptive to what you what to commit. Then click the commit button.

Once you've committed, Push your changes to Gitlab.

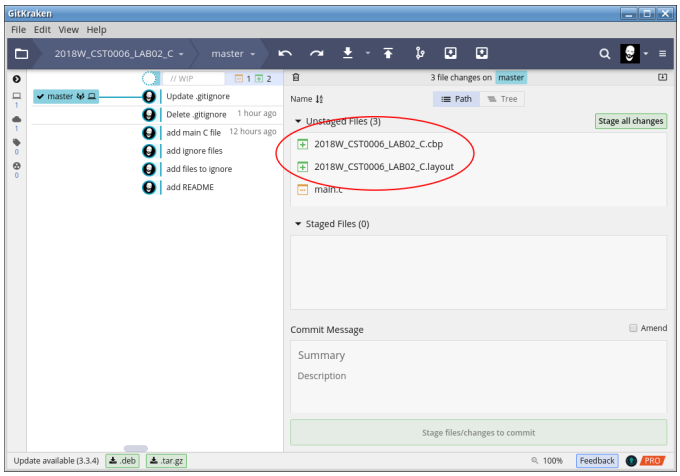
Notice the visual changes, you can see your commit on the left now.

We are now going to do something similar using GitKraken

Since **2018W_CST0006_LAB02_C.cbp** and **2018W_CST0006_LAB02_C.layout** (if you have it) are both Code::Blocks configuration files, that is set specifically for your computer and settings, you don't want to upload these files to everyone. They will end up changing everyone's settings to the way you have yours set. This could potentially cause you to get called mean names by a disgruntled team mate. So you'll want to ignore these files too. If you don't have the layout file, don't worry about it.

On the right, under Unstaged Files, right-click and ignore the configuration file.

Here is an example of the two files you need to ignore in GitKraken:



Then stage .gitignore, write a commit message and commit the changes.

Push your changes to Gitlab.

Finally, using whichever GUI you want, commit the main.c file and push to GitLab.

Once you are done, **take a screenshot of GitAhead, GitKraken, and GitLab Activity** showing all four of the commits.

Submit all 5 screenshots to blackboard for marking.

When you're done, you are allowed to help others… but make sure you're helping and not doing. You can also try opening your other repositories from last week to see what branching looks like in a GUI.

Remember, the more you use these tools, the more familiar you will become with them. You can even try other Git GUIs to see which one you like best.