Working Remotely Suggestions

Here is a collection of additional suggestions that you can try. There is no single way of working on your code and testing it on the server, so we cannot tell you exactly what to do. Instead, we'll provide a list of suggestions and you can choose one of them. All of them should allow you to work on this course.

Option: Use a local IDE/editor to work with files directly in the server

1. The `GettingStarted.pdf` (available on Git and Learn) contains instructions on how to install SSHFS. It allows you to see the files on the school server as a folder on your local computer.

   The "SSHFS-Instructions" PDF provides useful step by step instructions CSCF for installing the Secure SHell File System, SSHFS, on Windows 10.

   (Note that SSHFS for Linux won't work on the Windows Subsystem for Linux because that's not a "real" Linux. To install on Windows, you will need to follow the instructions from the link above.)

2. If you're on Windows, you can also try Samba instead of SSHFS. Samba should be much easier to set up. However, it requires the use of a VPN and we've heard that the VPN is not too stable nowadays with so many people needing to use it. If you want to try Samba, check these instructions at `https://cs.uwaterloo.ca/twiki/view/ISG/SambaWindows` (replace the course id with your own userid).

3. Visual Studio Code has a remote development plugin, which allows you to open files remotely via SSH without needing SSHFS installed in your system. It's a neat feature and it is simpler to install than SSHFS.

   NOTE: CS246 and CSCF do **NOT** support VSCode and you must have a backup option at hand in case there are interruptions with VSCode.

   Instructions: See Remote Development using SSH, `https://code.visualstudio.com/docs/remote/ssh`.

   You can set the default end-of-line (EOL) in your workspace settings if you set up a `.code-workspace` file in your home directory and open that workspace with the Remote-SSH extension. In particular, you'll want to change your end-of-line character to line feeds (LF i.e. "`\n`"), since Windows uses carriage returns followed by line feeds (CRLF "`\r\n`") and these don't work well with Linux-like operating systems and will likely cause errors on Marmoset.

Option: Edit the files locally and upload them to the server to compile and test

Any of the options above require a stable connection. None of those systems will behave well if the connection breaks while you have a remote file opened on an editor. If you're physically far from the University, it may be hard to control the stability of the connection. Even if everything is good on your end and the University's end, there may be another node in between that can break the connection at times. If you notice that your connection is not stable and it's hard to edit files directly on the server, another option is to edit the files locally and upload them to the server to compile and test. It may slow your work a bit because you always need to upload files before running them, but it should work better overall if the connection is not stable. Over the years, we've used this kind of strategy in many situations and once you get used to it, it's not as bad as it may seem initially.

4. Write all your files locally. Every time you want to test them, upload them to the server using a SFTP or SCP client (e.g. Filezilla, WinSCP, or just the `scp` command, which you can save in a script or alias so you don't need to retype every time). Once the files are uploaded, then just connect to the server to compile and run them.

5. Create a private Git repository for yourself. You can do this using UW's Gitlab (`https://git.uwaterloo.ca/`) server (or Github if you prefer). **But please ensure that your repository is private!** If you make it public, other students may access your code and copy from you, so you may end up in an academic integrity case.

   Then, you can checkout a copy of the repository on your local computer and a copy on the school server. Now, you can edit all your files locally and push them to the Git repository when you want to test them. Then, just connect to the school server and do a `git pull` to download the updated files from Git.

Option: Learn to use vim or emacs efficiently

6. Many people only know enough about vim or emacs to use the basic features. Some people prefer to use graphical editors. However, vim and emacs were created with the goal of enabling people to write code efficiently using them. There are lots of people in the world that use them and write code as effectively as people using graphical editors. So, if you take some time to learn vim or emacs well enough (there are plenty of tutorials and documentation online), you should be able to do all your coding using one of them in the server directly, without needing to use a local graphical editor.

   A word of warning, however. IDEs (integrated development environments) are popular with new coders due to the auto-complete feature among other features.

   That's not bad when it comes to suggesting variables/methods from context; *however*, you should make sure you really pay attention and understand the choices suggested. Otherwise, you're missing out on some of the learning/internalization that using a simpler editor forces you to learn. And in many circumstances, such as exams, you won't have either the tools or the time to play around with an IDE.

   tl;dr Yes, you can use an IDE; but, make sure you pay attention and understand why it's making those suggestions; otherwise, maybe save it for A4 and up.

Option: Work locally and only use the school server for the final tests

If the connection is so unstable that all of the options above don't seem to work, you may need to work locally. That's why it's the last option. If all else fails, it may be the best option for you.

7. If you really need to work locally, it's better to work on Ubuntu.

   - Ubuntu: The school servers use Ubuntu, so if you also use it, it's more likely that your environment will be similar to the school's environment.
   - Windows: You can install Ubuntu natively in your computer, use the Windows Subsystem for Linux (if you're on Windows), or create a virtual machine (using, for example, VirtualBox) and install Ubuntu on it. Ensure that you're setting g++ on your Ubuntu to use C++14 (as we'll tell you to do in the school server, so just do the same locally). Now, you will be able to edit your files locally and also compile and test them locally.
   - Mac: while it runs a flavour of Linux, the C/C++ compiler they use nowadays is `clang`, not GCC. While `clang` is *mostly* compatible with `gcc`/`g++`, it's not 100% compatible, so be careful about using a Mac to do all of your work locally. Note that it is possible to install a virtual operating system on a Mac, though, so if you really want to, you can set up a virtual machine on your Mac that runs Ubuntu.

   After you've finished your work but before you submit it to Marmoset, upload it to the school server (see options 4 and 5 above for ways to do it). Now, test your program extensively in the school server to ensure that it's working there. If you create good automated tests like we'll tell you to do, it will be easy because you can just run your test script on the server. After you're sure that your program works well in the school server, then you're ready to submit it.