Version Control Tutorial using TortoiseSVN and TortoiseGit

Christopher J. Roy, Associate Professor

Virginia Tech, cjroy@vt.edu

This tutorial can be found at:

www.aoe.vt.edu/people/webpages/cjroy/Software-Resources/Tortoise-SVN-Git-Tutorial.pdf

Software engineering is critical for accurate and reliable simulation codes in scientific computing. However, most scientists and engineers receive little or no formal training in software engineering. Configuration management is an important component of software engineering which deals with the control and management of software products. The key aspects include using version (or revision) control for source code and other software artifacts, recording and tracking issues with the software, and ensuring backups are made. This tutorial will focus on the first of these: software version control. While most software engineering practices are critical only for large software development efforts, every software project, regardless of how large or small, should use a version control system for the source code.

Version Control

Version control tracks changes to source code or any other files. A good version control system can tell you what was changed, who changed it, and when it was changed. It allows a software developer to undo any changes to the code, going back to any prior

version, release, or date. This can be particularly helpful when a researcher is trying to reproduce results from an earlier paper or report and merely requires documentation of the version number. Version control also provides a mechanism for incorporating changes from multiple developers, an essential feature for large software projects or any projects with geographically remote developers.

Some key concepts pertaining to version control are discussed below. Note that the generic descriptor "file" is used and could represent not only source code, but also user's manuals, software tests, design documents, web pages, or any other item produced during software development.

repository – a location where the current and all prior versions of the files are stored; in distributed version control systems, there is a master repository which can be copied (or "cloned") locally for development

working copy – the local copy of a file from the repository which can be modified and then checked in or "committed" to the repository

check-out – the process of creating a working copy from a repository (either the current version or an earlier version)

check-in – a check-in or commit occurs when changes made to a working copy are merged into a repository

push – the merging of changes from a local repository to the master repository (for distributed version control systems)

diff – a summary of the differences between two versions of a file, often taking the form of the two files side-by-side with differences highlighted

conflict – a conflict occurs when two or more developers attempt to make changes to the same file and the system is unable to reconcile the changes (note: conflicts generally must be resolved by either choosing one version over the other or by integrating the changes from both into the repository by hand)

update – merges recent changes to a repository into a working copy (for centralized version control systems)

pull – merges recent changes to the master repository into a local repository (for distributed version control systems)

The basic steps that one would use to get started with a centralized version control system (such as Subversion/SVN) are as follows:

1. Create a repository
2. Import a directory structure and/or files into the repository
3. Check-out the repository version as a working copy
4. Edit/modify the files in the working copy and examine the differences between the working copy and the repository (i.e., diff)
5. Check-in (or commit) the changes to the repository

The basic steps that one would use to get started with a distributed version control system (such as Git) are as follows:

1. Create an original “master” repository
2. Clone the repository to allow development
3. Add a directory structure and/or files into the local repository

4. Check-in or “commit” the directory structure and/or files to the local repository
5. Push these modifications of the local repository into the master repository
6. Edit/modify the files in the working copy and examine the differences between the modified files and those in the repository (i.e., diff)
7. Check-in or “commit” the file modifications to the local repository
8. Pull any modifications from the original repository into the local repository (in case someone has modified some files and your files are out of date)
9. Push these modifications of the local repository into the master repository

1 Recommended Version Control Systems

There is a wide array of version (or revision) control systems available to the software developer. We will focus on free, open-source version control systems. One up front choice you will have to make is whether you want a centralized repository (e.g., Subversion, CVS, Vesta) or a distributed repository (e.g., Git, Mercurial, Bazaar). A detailed list of both open source and proprietary version control software can be found at: en.wikipedia.org/wiki/List_of_revision_control_software

The most popular, freely-available, centralized version control system is Subversion (SVN), which is generally seen as the successor to CVS. Subversion is also freely-available (at subversion.tigris.org), is open-source and actively maintained, and has a very useful, free book available online (Collins-Sussman et al., 2007). Subversion has a number of easy-to-use Graphical User Interfaces (GUIs) for all major computing platforms. Eddins (2006) recommends using TortoiseSVN (tortoisesvn.tigris.org) on

Windows platforms (which integrates into Windows Explorer) and RapidSVN (rapidsvn.tigris.org) on Linux and Macintosh platforms.

The leading distributed version control systems are Git and Mercurial; however we will focus on Git. The main difference between Subversion and Git is that while SVN provides for a single, centralized repository, Git allows a number of distributed repositories that are all tied in to a master repository. Thus Git makes it easier for a number of developers to perform their work on their own projects then integrate these changes back into the master repository once it appears to be working correctly. This helps prevent developers from "breaking" the main version of the code during development. Another advantage of the distributed repository system is that one does not need network access to the master repository in order to do development. To access Git from a Unix/Linux command line, please see the main Git web site which contains documentation, Git cheatsheets, and a 15 minute Git tutorial. A useful Windows GUI for Git is TortoiseGit, which is based on TortoiseSVN discussed earlier. The examples given in the next two sections were developed using TortoiseSVN and TortoiseGit, respectively.

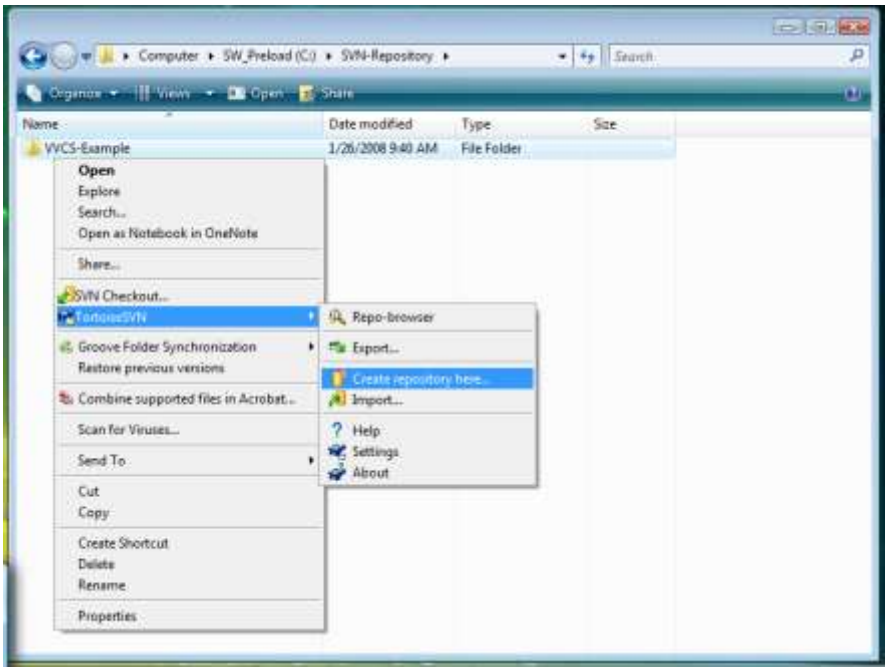
2 TortoiseSVN Tutorial

The following tutorial on the Subversion version control system was created using TortoiseSVN version 1.4.5 on a computer running Microsoft Windows.

2.1 Creating a Repository

Determine a location for the repository, ideally on a server which is automatically backed up. Create a folder with the name of the repository; in this example the repository is called "VVCS-Example." Right click on the folder name, choose "TortoiseSVN"

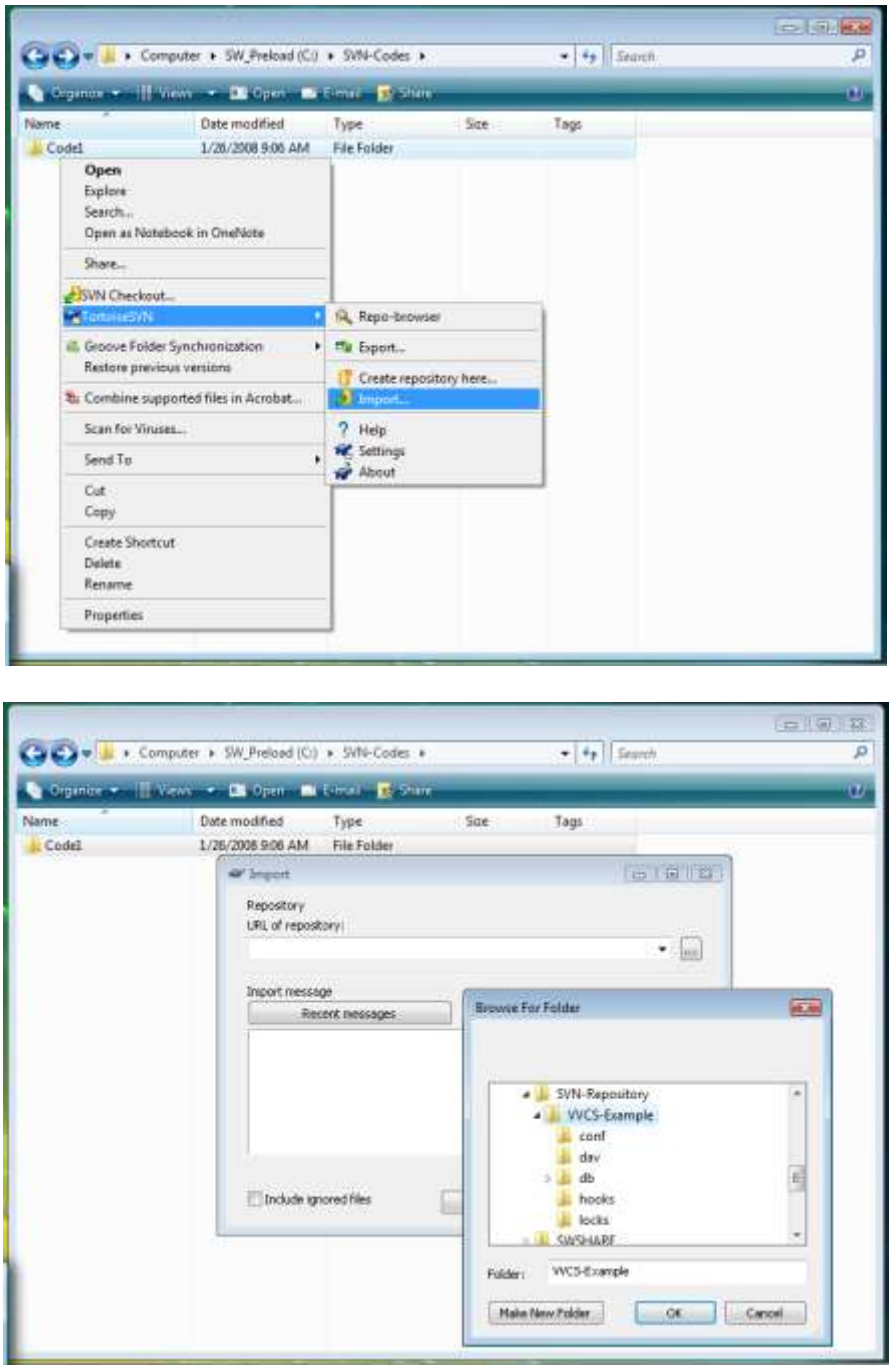
(which is integrated into the Microsoft Windows Explorer menu), then "Create Repository Here." Choose the Native Filesystem, then you should see the message "Repository Successfully Created."



Creating a repository

2.2 Importing a File into the Repository

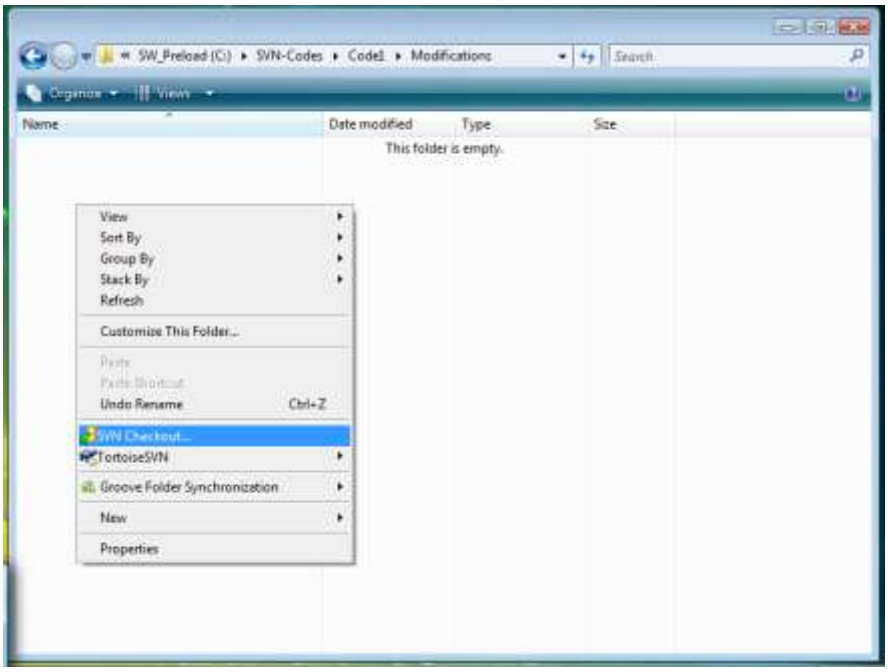
Right click on the directory containing the file(s) and/or directory structure you wish to import to the repository (note, the directory that you click on will not be imported). Here we will simply be importing the file "code1.f" from directory "Code1." This code creates a 17×17 two-dimensional Cartesian grid for x and y between 0 and 1. Browse until you find the location of the repository "VVCS-Example" and select that directory name. This version of the code will be Revision 1.



Importing a file into the repository: selecting Import in the TortoiseSVN menu (top) and selecting the repository directory name (bottom)

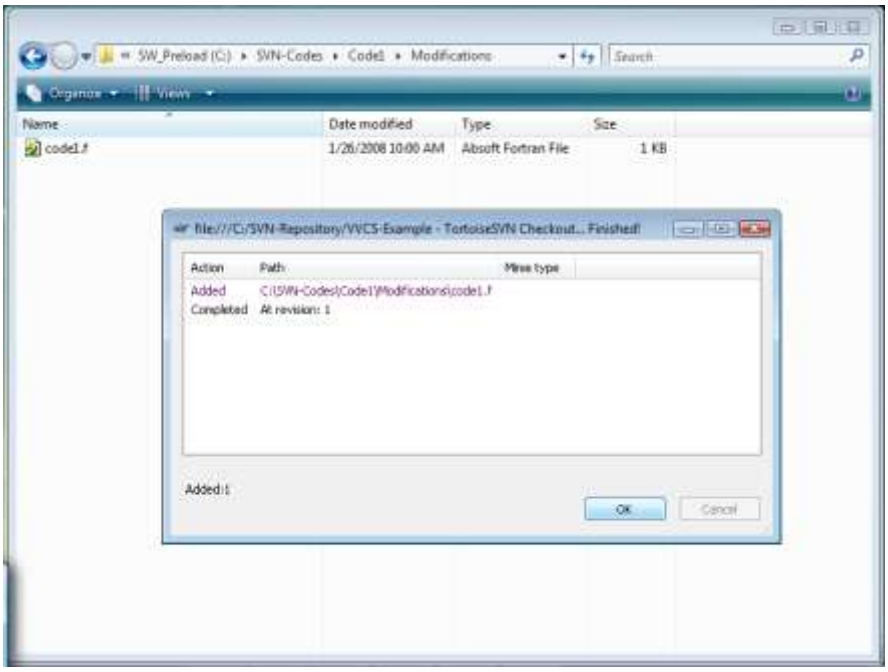
2.3 Checking the Code out from the Repository

You now have the code “code1.f” safely placed in the repository. To modify this code and create a new revision, you will need to check out a working copy of the code. Go to the directory where you will be modifying the code, in this example, the directory “Modifications.” Right click in Windows Explorer, and select “SVN Checkout…” Select the name of the repository you just created, then click “OK.” You will now get a window telling you that you are at Revision 1. Notice the green check mark on the “code1.f” icon. This indicates that this working copy is up to date with the version in the repository.



a) SVN Checkout

b) Selecting the proper repository

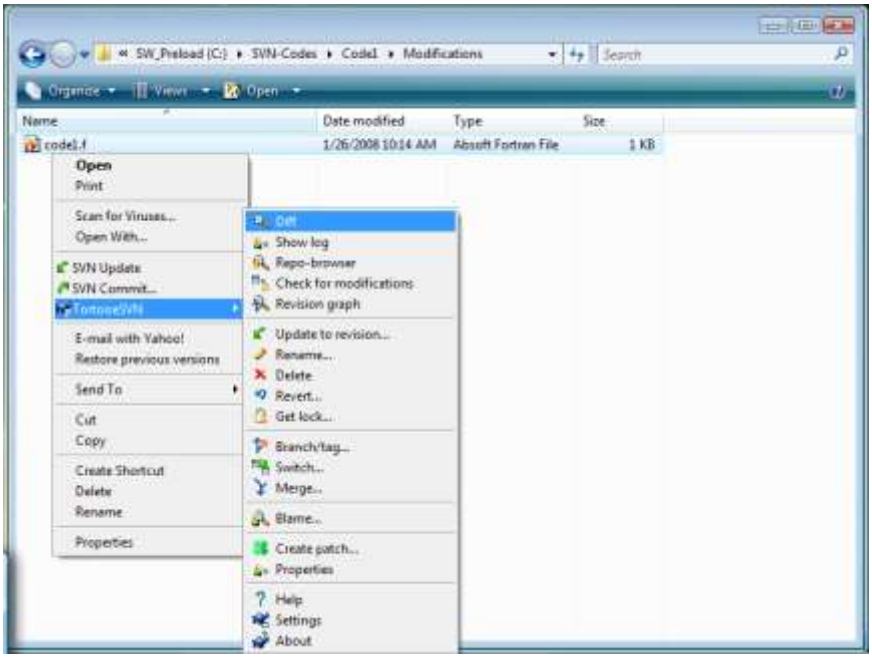


c) Successful checkout of Revision 1

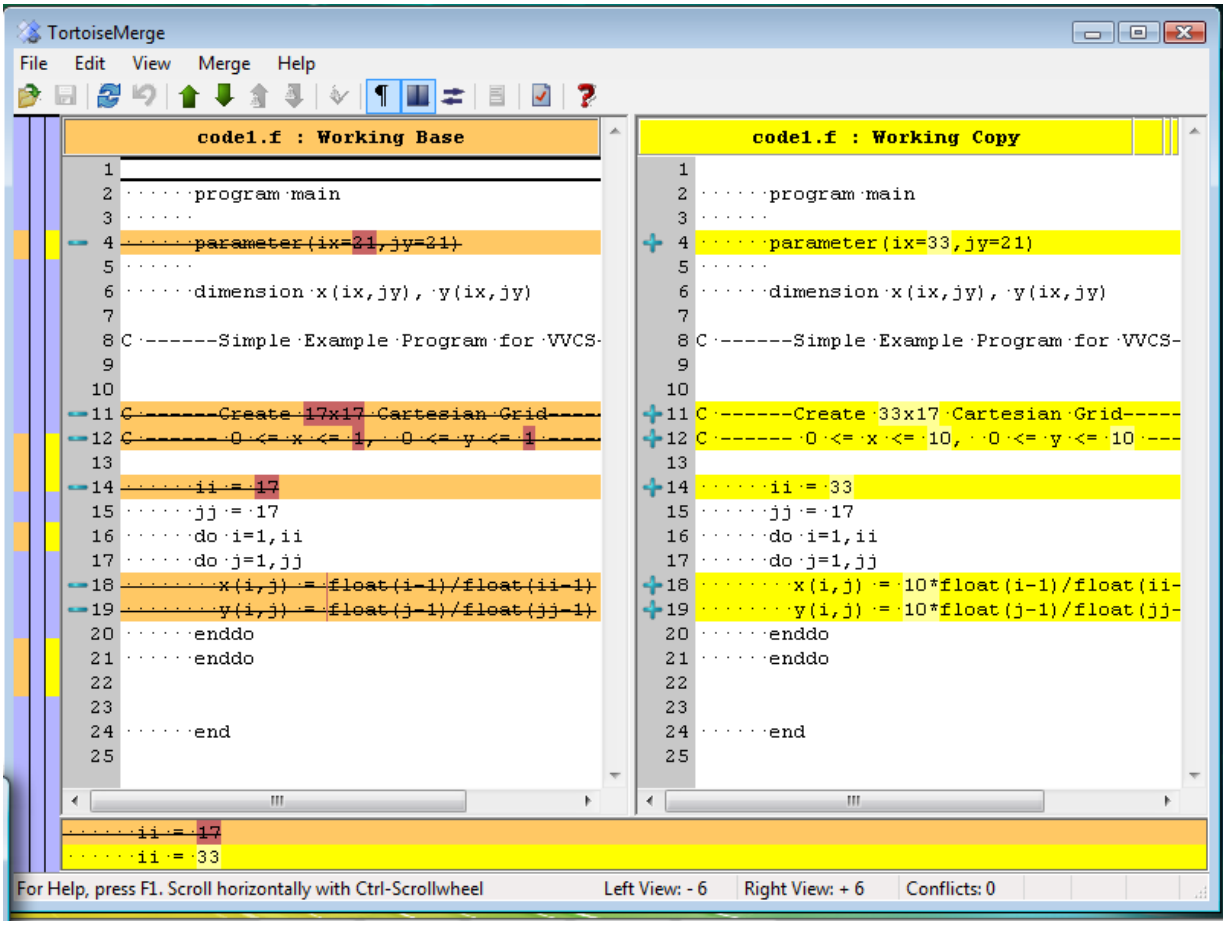
Checking out a working copy from the repository

2.4 Modify the Code and Compare to the Repository Version

The code “code1.f” can now be modified. Here we will change the code to allow the Cartesian grid to contain 33x17 points between the values of zero and ten. Once the code has been modified, you will notice that the green check mark has been replaced by a red exclamation point, indicating that the current working copy has been modified from the version in the repository. To examine these differences, right click on the “code1.f” file, select “TortoiseSVN,” then “Diff.” This opens the “TortoiseMerge” tool which clearly shows the modifications to the repository version (Working Base) that were made in the Working Copy.



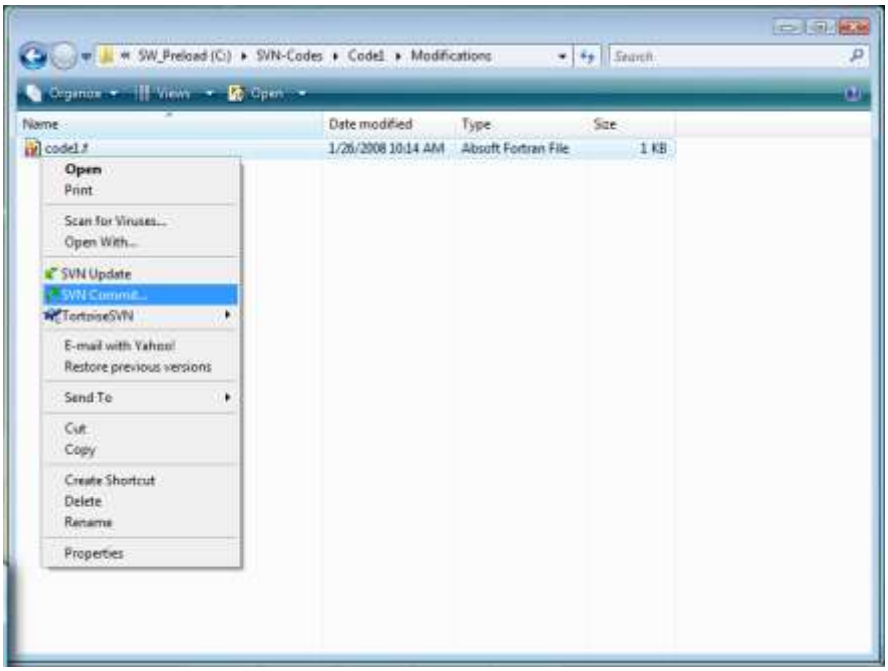
Comparing the Working Copy to the repository version (Working Base)



TortoiseMerge tool which clearly highlights differences between the modified version (Working Copy) and the repository version (Working Base)

2.5 Check the Code back into the Repository

When the changes are complete, the repository can be updated with your modified Working Copy by performing a checkin. Just right click on the file (or directory) and select "SVN Commit…" Enter a message describing the changes that were made, then select "OK." You are now at Revision 2.



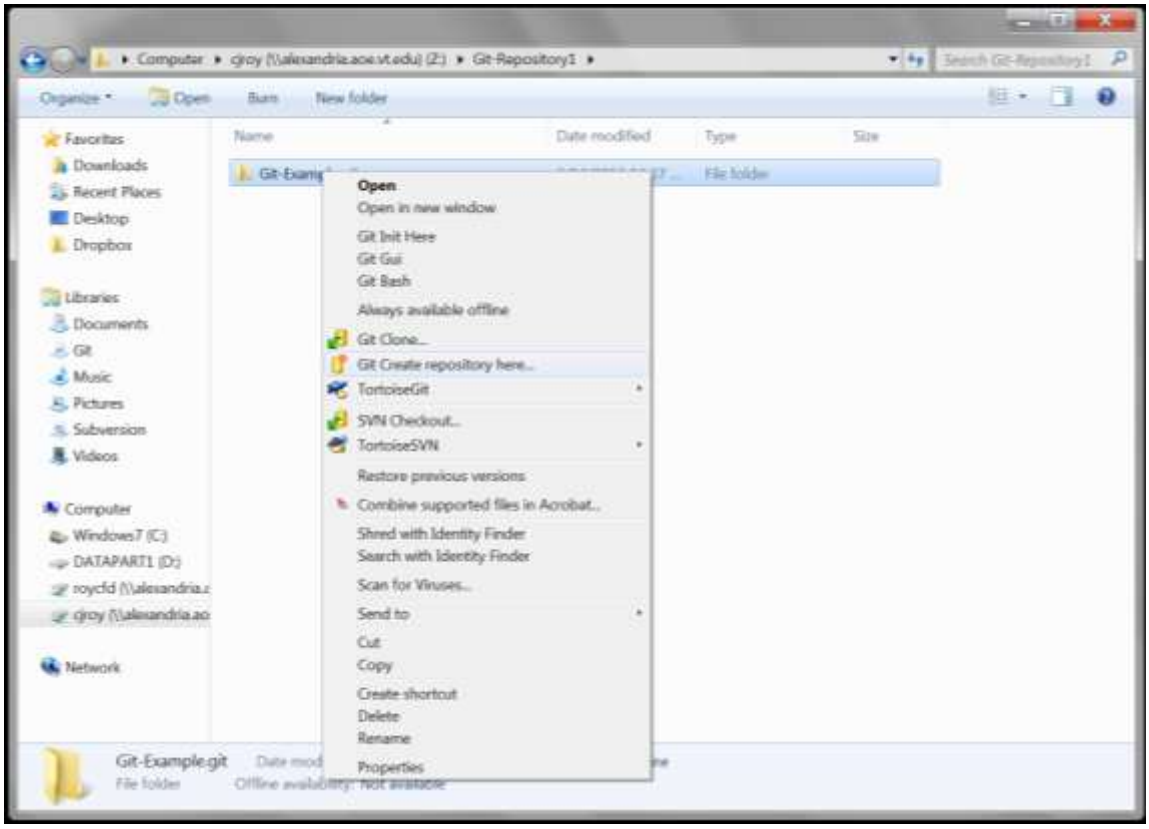
Check the modified version (Working Copy) into the repository

3 TortoiseGit Tutorial

The following tutorial on the Git version control system was created using TortoiseGit version 1.8.1.0 on a computer running Microsoft Windows 7. Another TortoiseGit tutorial is available at the TortoiseGit homepage under screenshots.

3.1 Creating a Master Repository

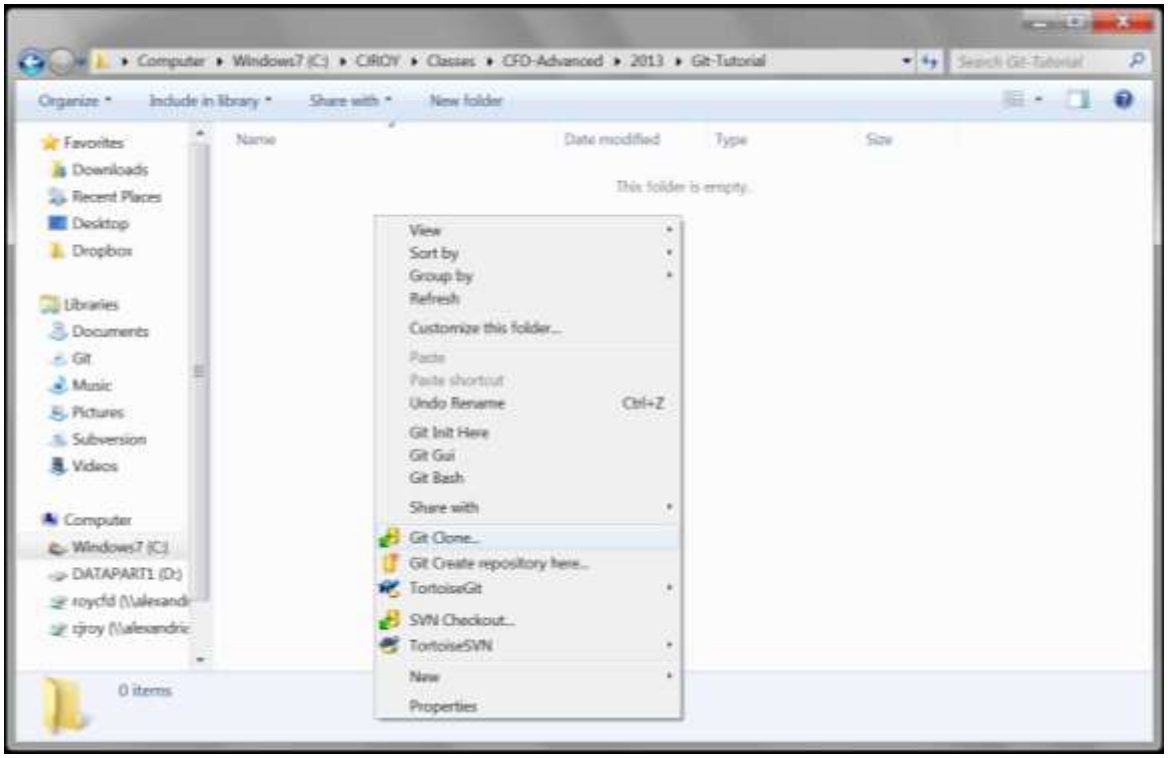
Determine a location for the master repository, ideally on a server which is automatically backed up. Create a folder with the name of the repository; in this example the repository is called “Git-Example.git”. Right click on the folder name, choose “Git Create repository here...”.



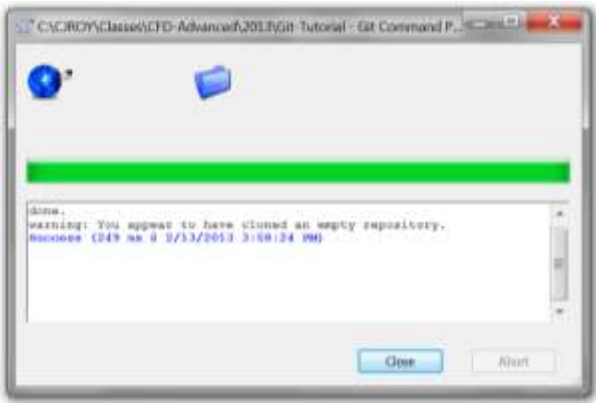
Check the box labeled “Make it Bare (no working directories)”, which means that you cannot do your work in this location, but instead it will serve as the master repository that you can copy locally to do your development work, then select “OK”. You should see a message like “Initialized empty Git repository in Z:/Git-Repository1/Git-Example.git/”, then click on “OK”.

3.2 Cloning the Repository

To create a local repository where you can work, you need to clone the master repository. Navigate to the local location where you want to work, then right click and select “Git Clone…”.

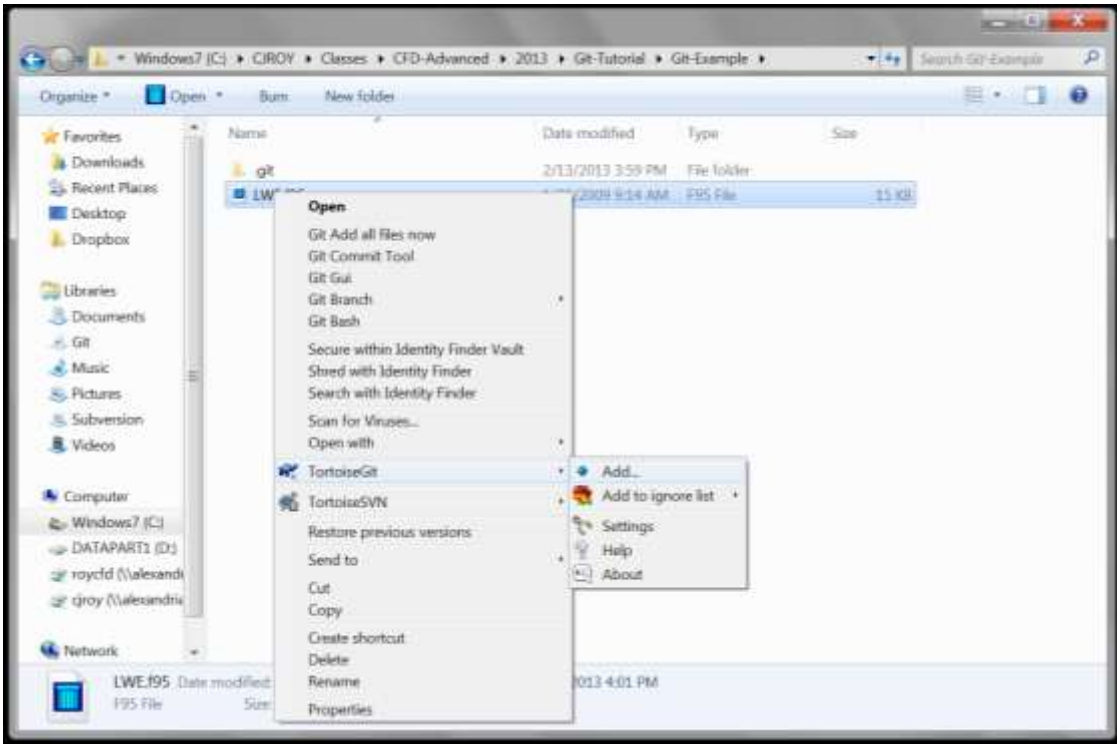


Select the “Dir…” button and navigate to the location of the master repository that you created, then select “OK”. You have now successfully cloned the master repository.

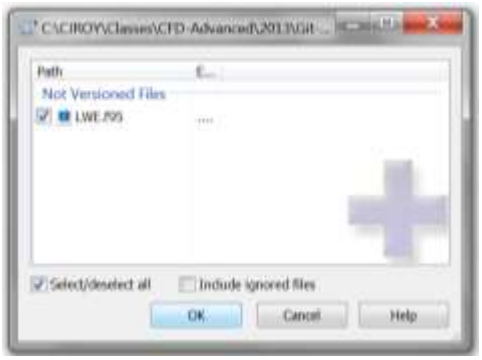


3.3 Adding a File to the Local Repository

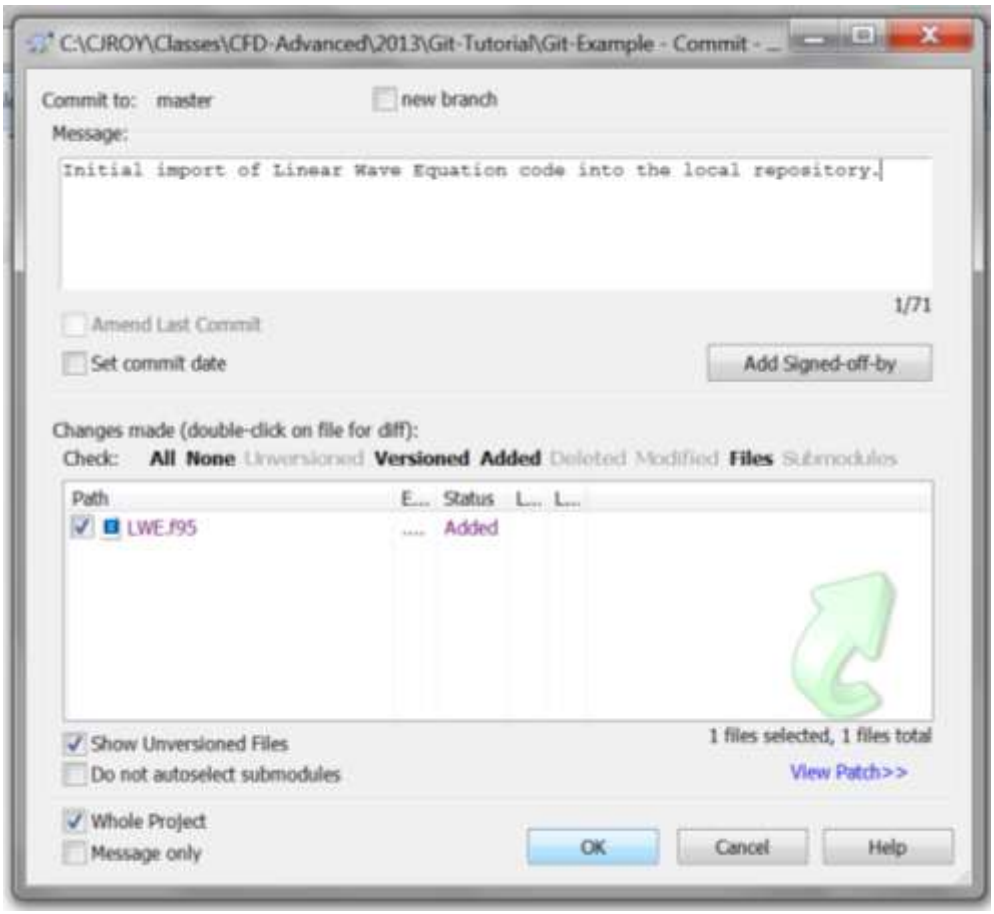
Copy any files you wish to add to the local repository into this "Git-Example" directory, then right-click and choose "TortoiseGit" then "Add…".



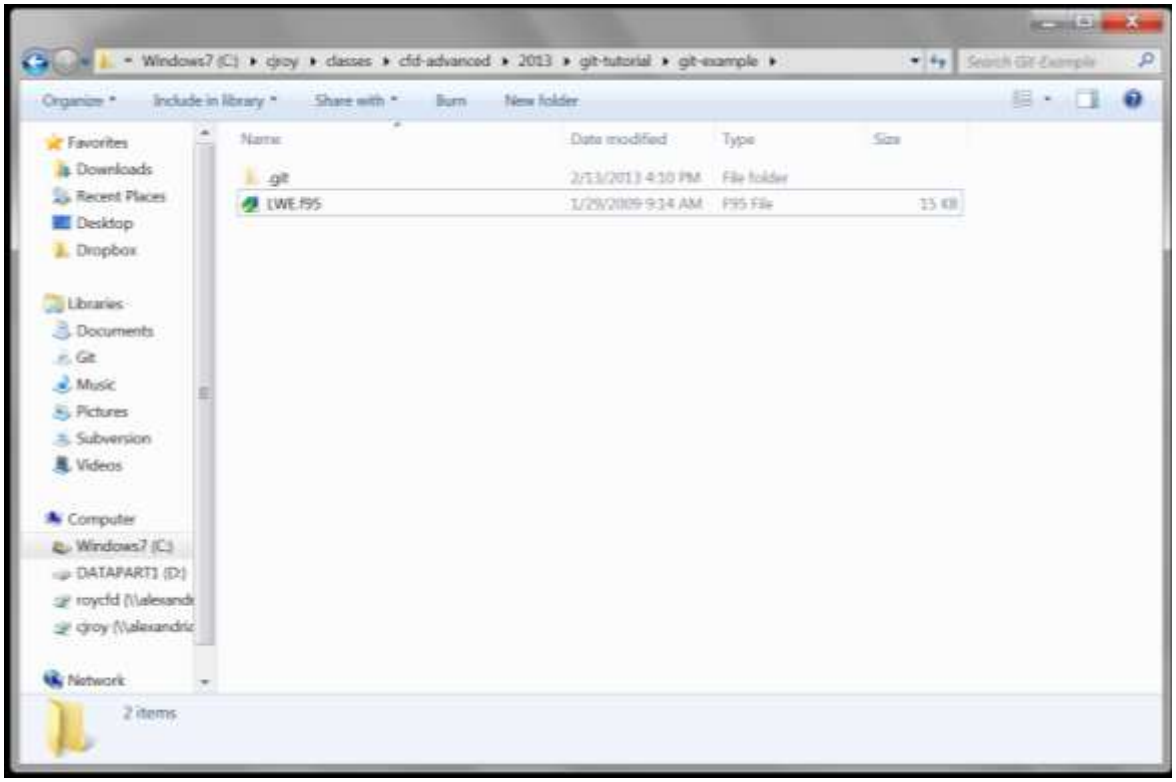
Place a check mark by the file(s) you want to add (here we are adding the single file LWE.f95) then select "OK". You can then select "Commit …" to commit this file to the local repository (note: it will not yet be added to the master repository).



You should now enter a message associated with this initial import of the file into your local repository then select “OK”.

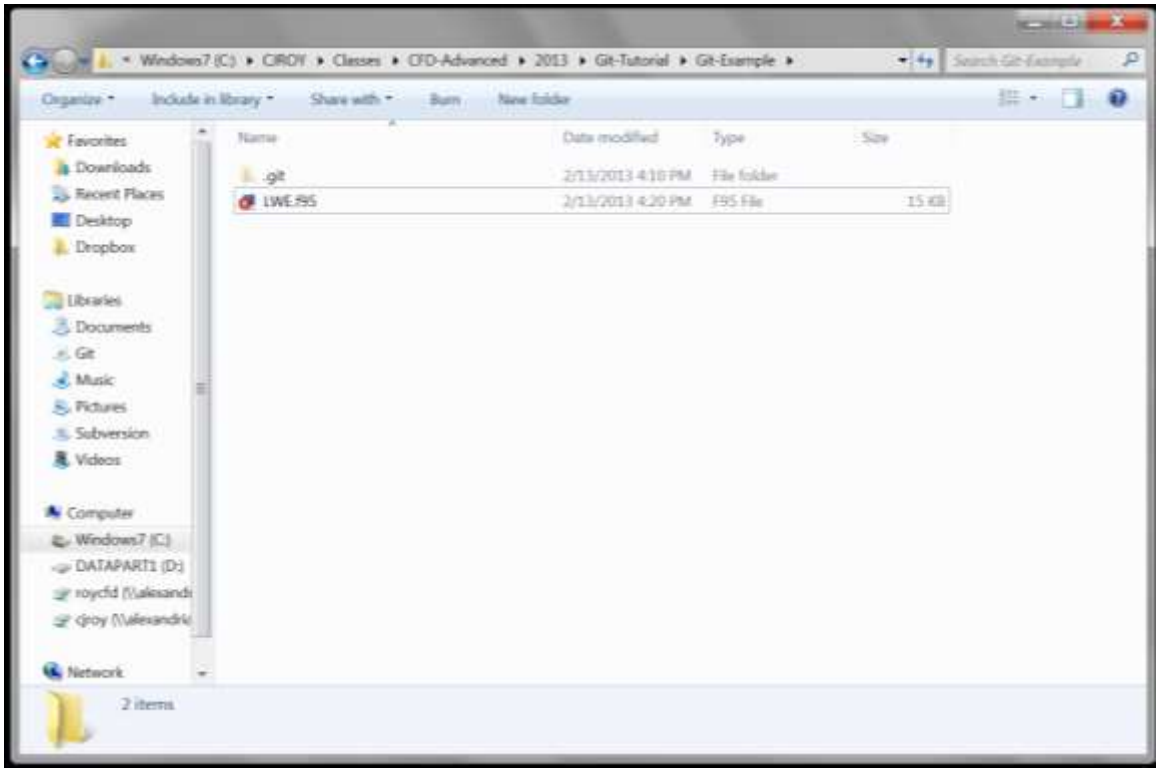


At this point, the file should show up with a green check mark indicating that this version is up to date relative to your local repository.

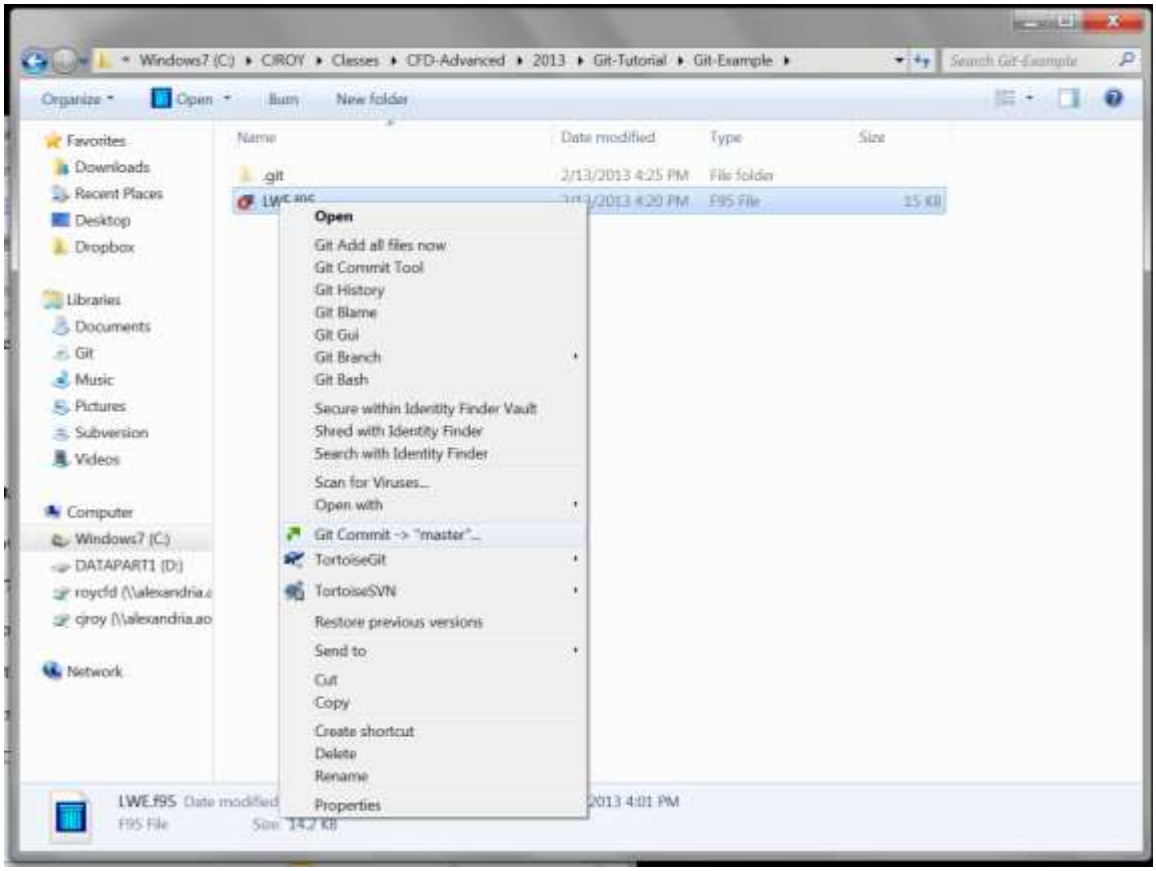


3.4 Modify the Code in the Local Repository

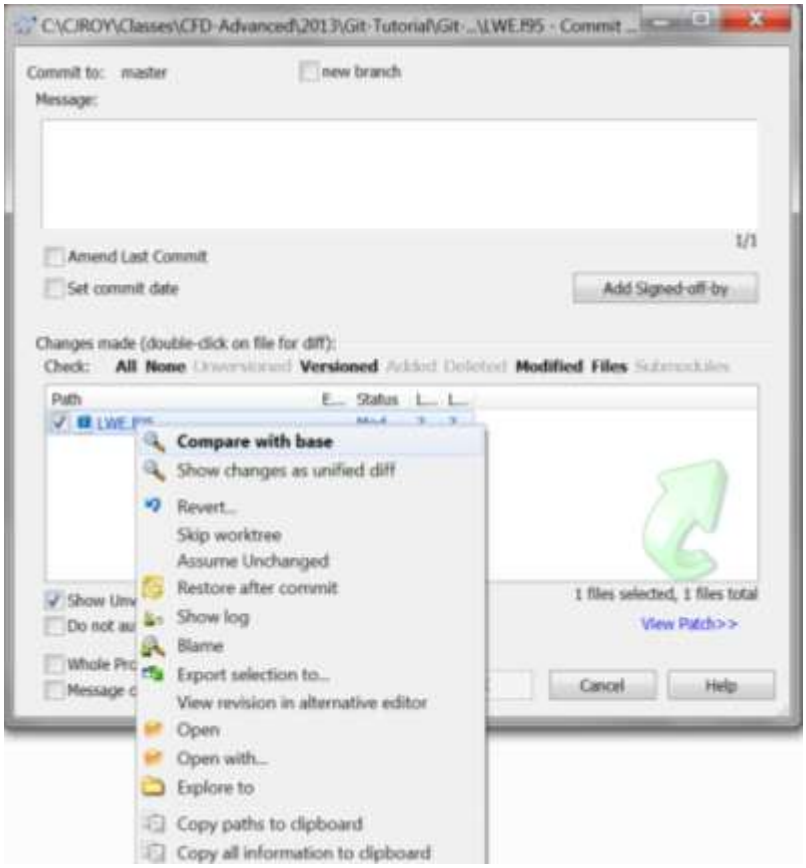
You now have the code “LWE.f95” safely placed in the local repository. You can now modify the code by changing some of the input parameters. Notice that the green check mark has now changed to a red exclamation mark, indicating that the file is no longer at the same revision level as the file in the repository.



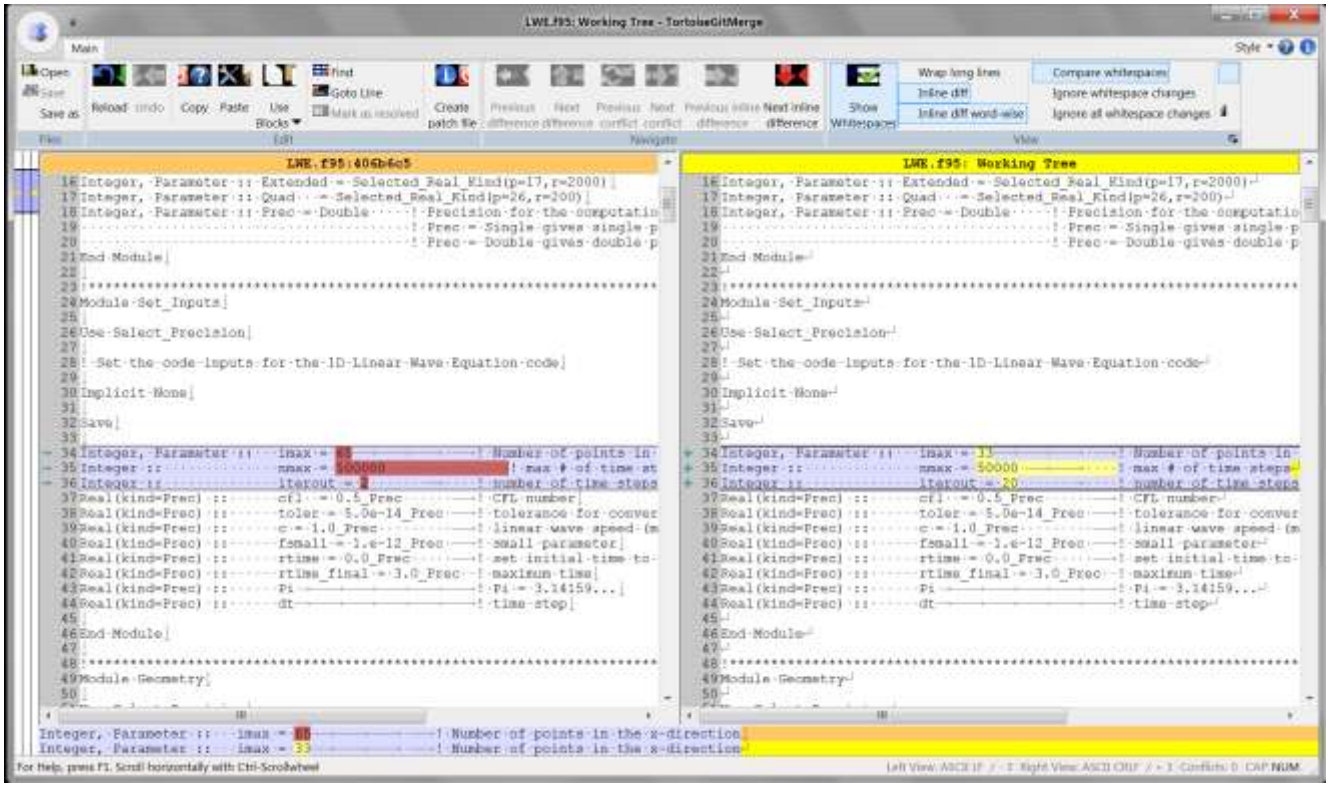
To check these changes in to the local repository, there are (at least) two options. First, one could right-click on the file name and select “Git Commit -> “master”…”.

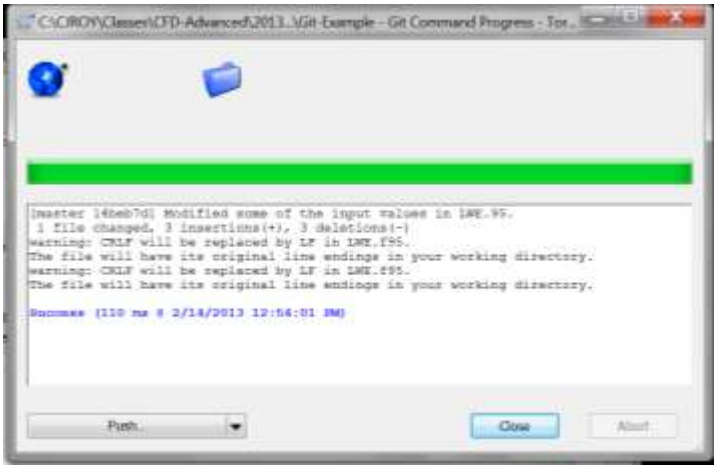


At this point, it is often useful to review the changes that were made to the file by right-clicking the file name and selecting “Compare with base”.

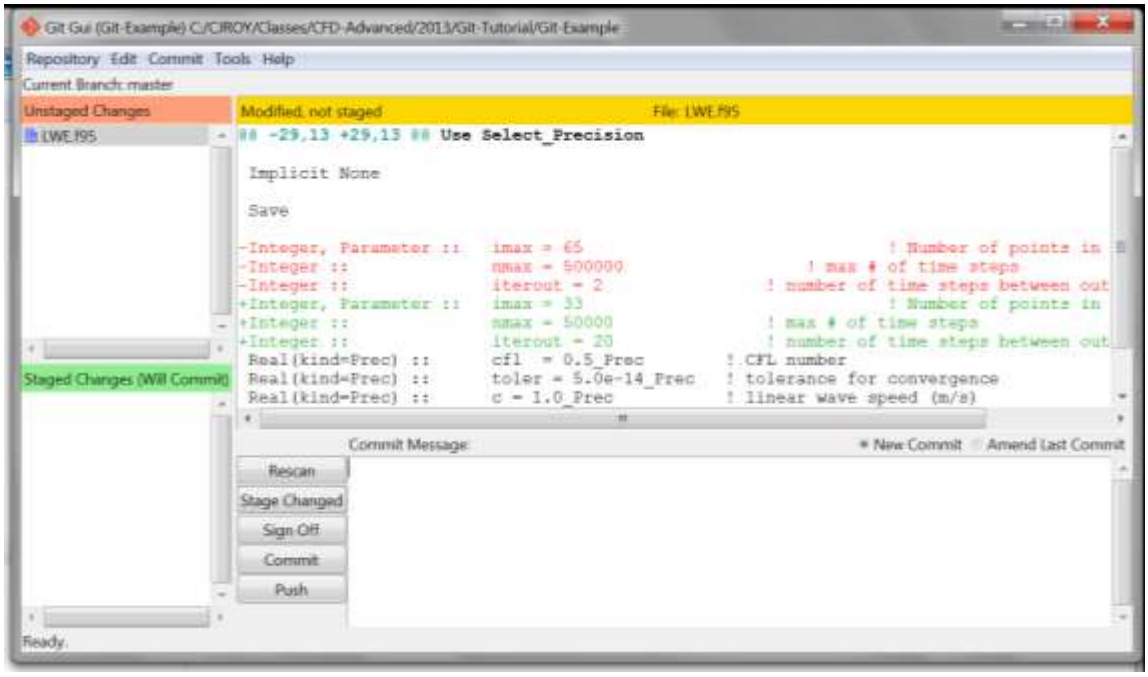


This bring up a valuable tool called “TortoiseGitMerge”, which highlights the changes that were made in the file by comparing the differences graphically (and with color) side-by-side: the original code on the left and the modified code (the “working tree”) on the right. The left side bar show a graphical summary of where all code changes are located (in this case, they are all near the top of the file). In addition, placing the cursor over any line with bring up the old and new lines at the bottom of the tool so they can be compared in more detail. Once you have reviewed the changes, enter a message in the “Commit” dialog box and select “OK”. If your commit was successful, then select “Close”.





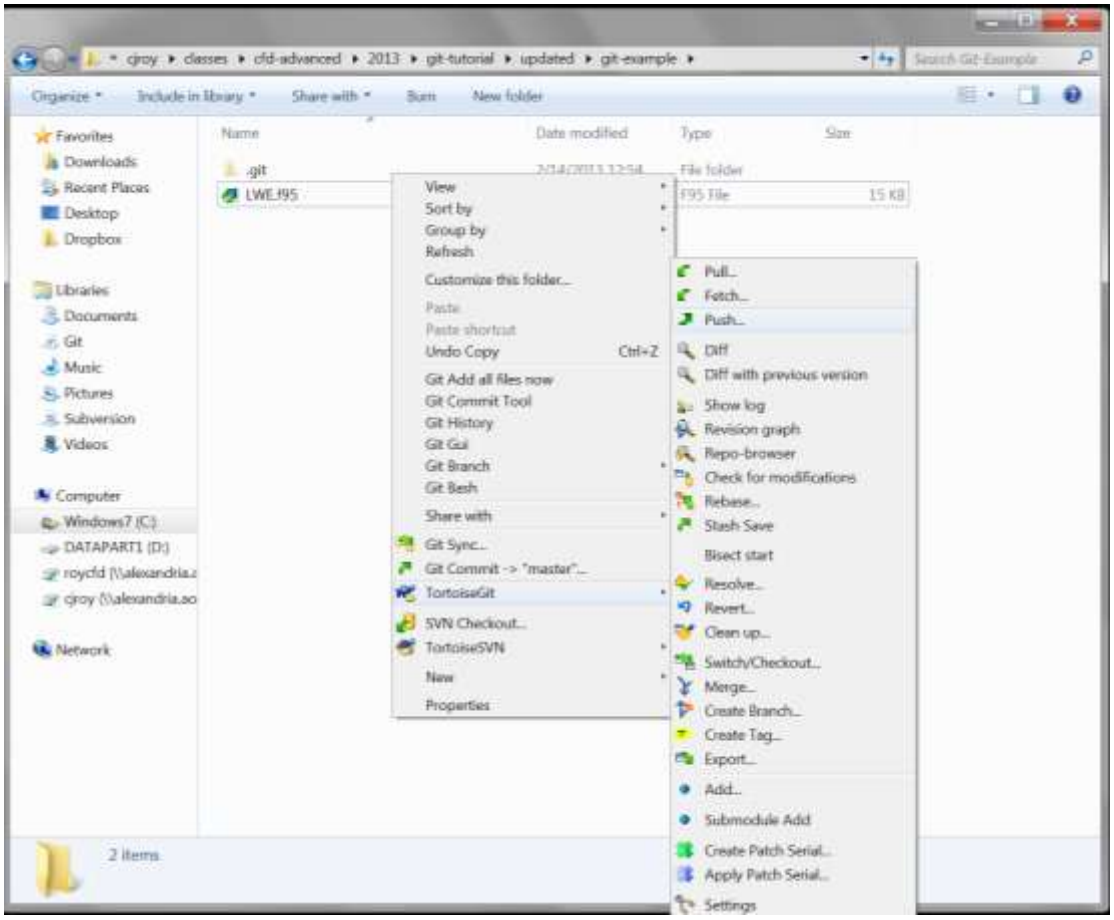
Alternatively, one could right-click on the file and select the “Git Commit Tool”, which brings up the following window indicating that three lines were changed from the version in red to the version in green.



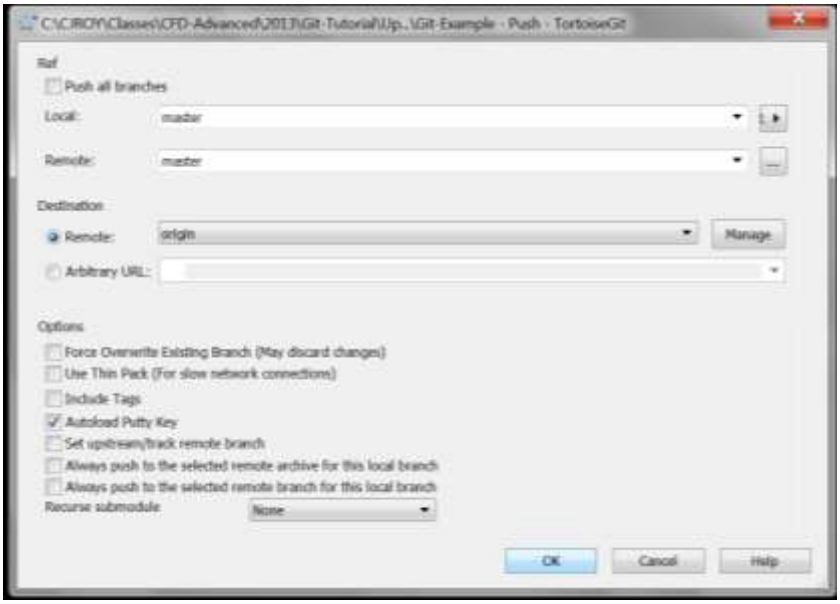
CAUTION: At this point, the file import and changes have not yet been "pushed' into the master repository, they have only been added to the local repository.

3.5 Pushing Changes to the Master Repository

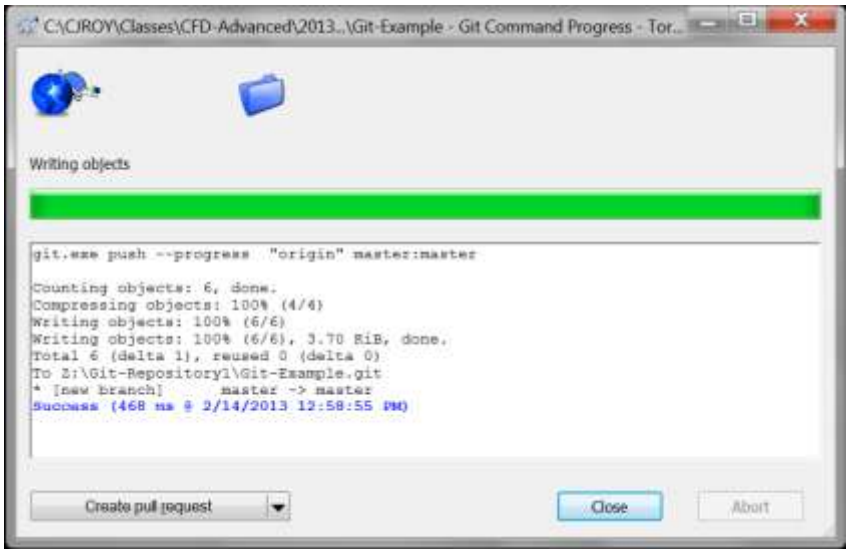
To push the changes from the local repository to the master repository, right-click in the window area (NOT on the LWE.f95 file), select "TortoiseGit", then "Push".



This will pull up the Git Push dialog box, then click on “OK”.



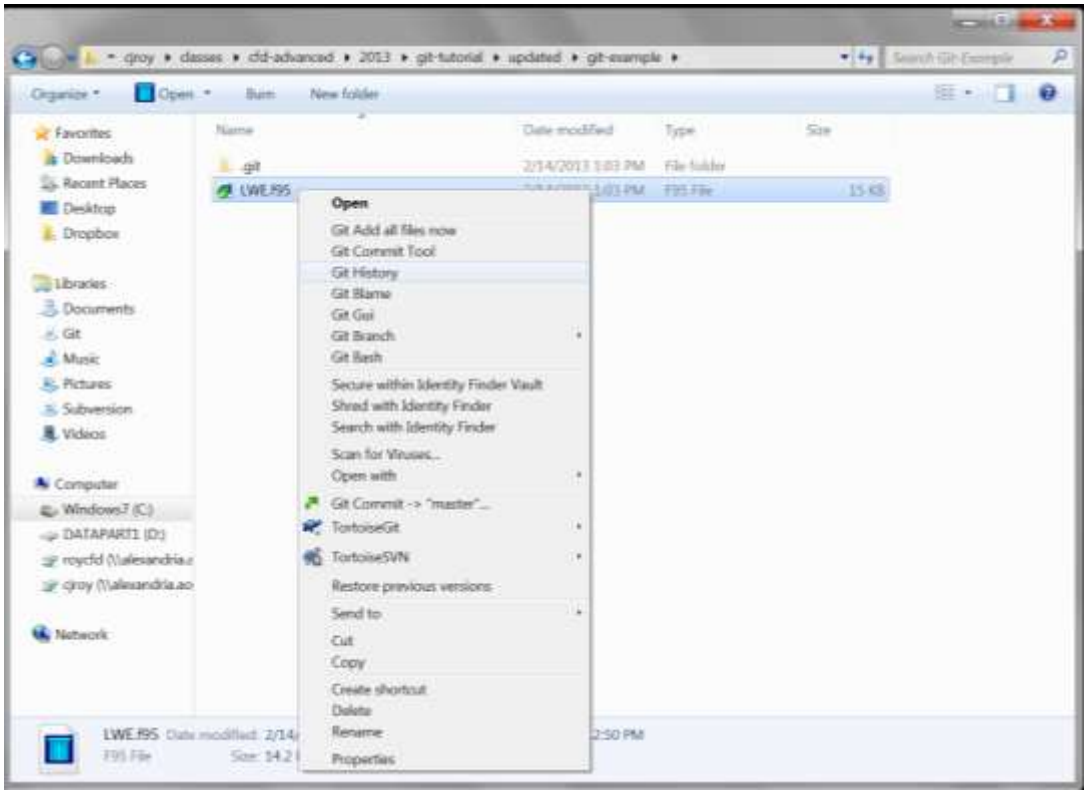
If you get the message that the push was successful, then the changes to your local repository have been successfully pushed to the master repository.

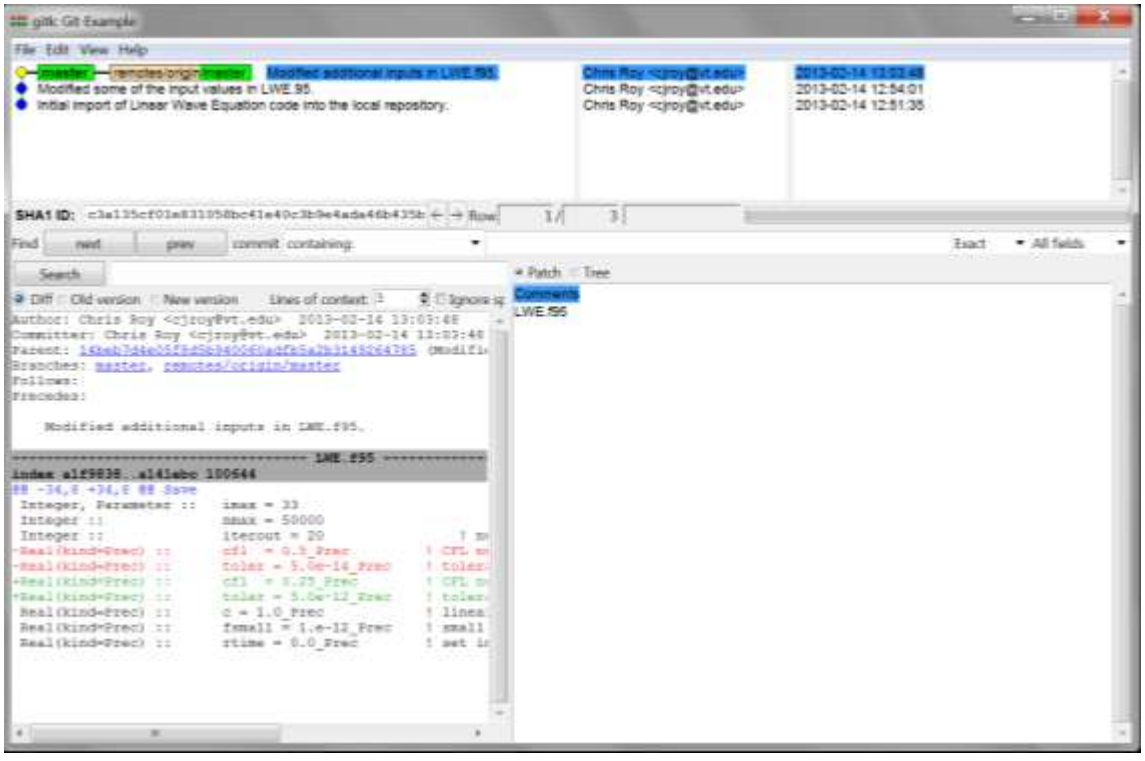


If you are developing code with other developers, before you "push" your changes to the master repository, you should first "pull" any changes that the other developers have pushed to the master repository since you last synchronized with it. To pull any changes in the master repository to your local repository, right-click in the window area (NOT on the LWE.f95 file), select "TortoiseGit", then "Pull".

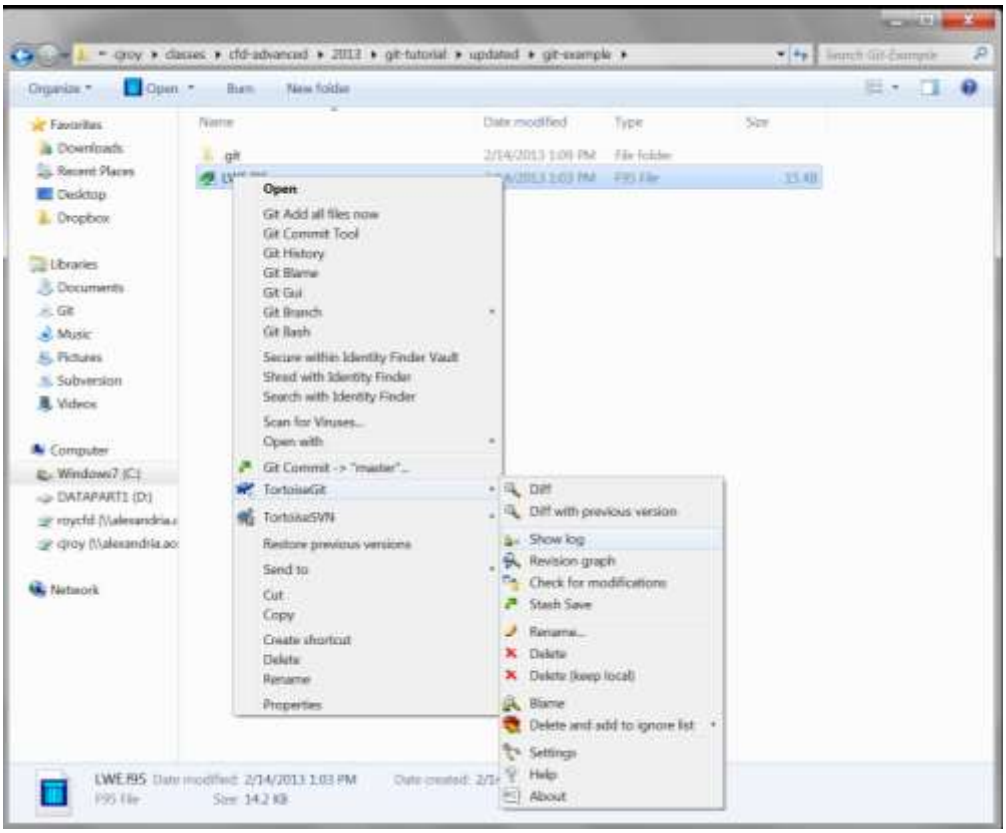
3.6 Other Useful Tools

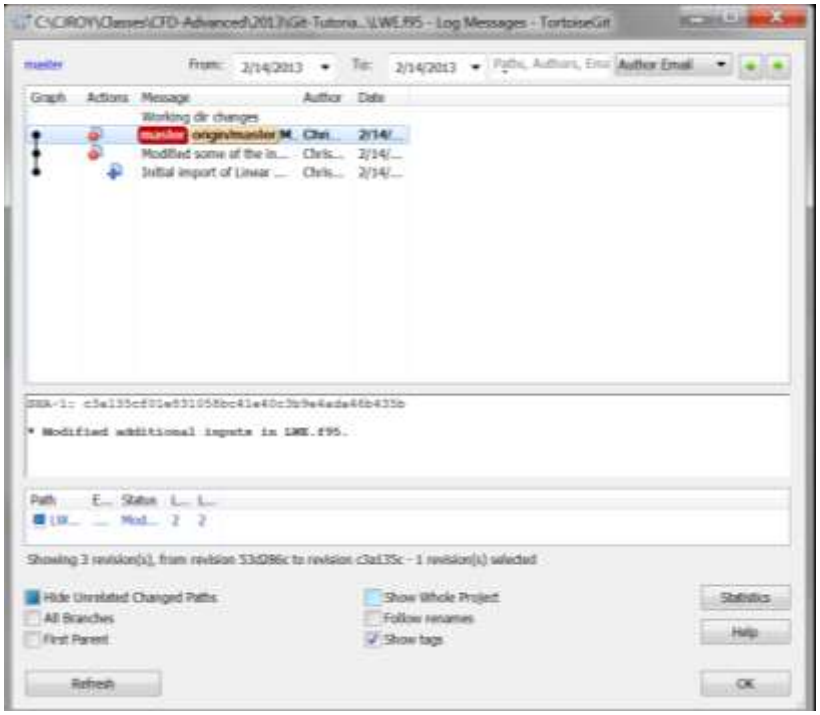
After you have made additional changes to the code, you may want to examine the history of the changes to a file/project by right-clicking on the file and selecting "Git History". This will pull up the "gitk" tool that is also available in Unix/Linux.


Open
Git Add all files now
Git Commit Tool
Git History
Git Blame
Git Gui
Git Branch
Git Bash
Secure within Identity Finder Vault
Shred with Identity Finder
Search with Identity Finder
Scan for Viruses...
Open with
Git Commit -> "master"...
TortoiseGit
TortoiseSVN
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties


gitk: Git Example
File Edit View Help
Modified additional inputs in LWE.f95.
Modified some of the input values in LWE.f95.
Initial import of Linear Wave Equation code into the local repository.
Chris Roy <cjroy@vt.edu>
2013-02-14 13:03:48
2013-02-14 12:54:01
2013-02-14 12:51:35
SHA1 ID:
Find
next
prev
commit containing:
Search
Diff
Old version
New version
Patch
Tree
Comments
LWE.f95

You may also want to see the log file for a given file by right-clicking on the file, selecting “TortoiseGit”, then “Show log”. This brings up a GUI which shows useful information about the file in question.





References

Steve Eddins (2006), "Taking Control of Your Code: Essential Software Development Tools for Engineers," International Conference on Image Processing, Atlanta, GA, Oct. 9, 2006 (see also: http://blogs.mathworks.com/images/steve/92/handout_final_icip2006.pdf).

Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato, *Version Control with Subversion: For Subversion 1.4* (Compiled from r2866), (see http://svnbook.red-bean.com/en/1.4/svn-book.pdf), last accessed January 16, 2008.