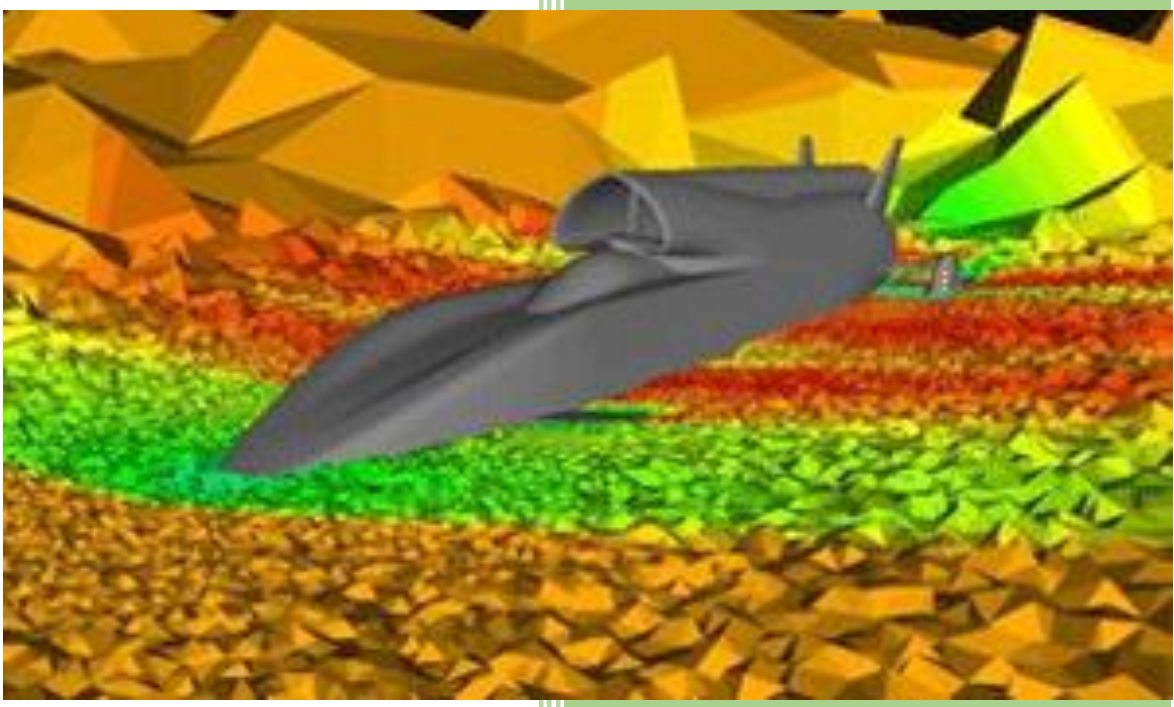
SwanSim - A Guide to Git / SourceTree / GitLab for Windows



Dr Jason W. Jones

College of Engineering, Swansea University

September 2017

Contents

1 Introduction

This document is part of a series of short guides available on the SwanSim web site (http://www.swansim.org).

The purpose of this document is to guide you through the steps of converting a standalone software project into one managed by the Git source code control system hosted by SwanSim's GitLab server (https://git.swansim.org).

This is then followed with the next most common operation, creating a copy of a Repository held on the GitLab server onto another PC that has had nothing previously installed.

It is highly recommended to use this guide on a software project that you are happy to delete in order to get used to the operation of the combination of Git, SourceTree and GitLab.

Once you are familiar with the basic operations then use the follow-up guide available on the SwanSim web site to learn the more advanced features of Git.

2 How to use this Guide

The relevant sections of this guide vary depending on your experience with Git, GitLab and SourceTree.

My experience	Sections to follow
I am new to all this Git Repository stuff	Section 3 - Obtaining the Software Section 4 - Creating a Repository from an existing project Section 5 - Pushing to the GitLab Server
I have a project on GitLab and I need to work on it on another computer	Section 3 - Obtaining the Software Section 6 - Cloning a GitLab Repository to my Computer

The initial setup of SourceTree and linking it to the GitLab server may seem arduous at first (especially if you have never done anything like this before) but once done, the day to day interaction with your repository is very simple.

3 Obtaining the Software

Git is a command-line driven tool that is usually installed as part of most Linux distributions and is available for Windows from https://git-scm.com/download/win.

However, learning to use Git in this manner is hard. For every platform, there is a choice of graphical tools whose aim is to make Git easier to use. The graphical tools used in this document are SourceTree (https://www.sourcetreeapp.com).

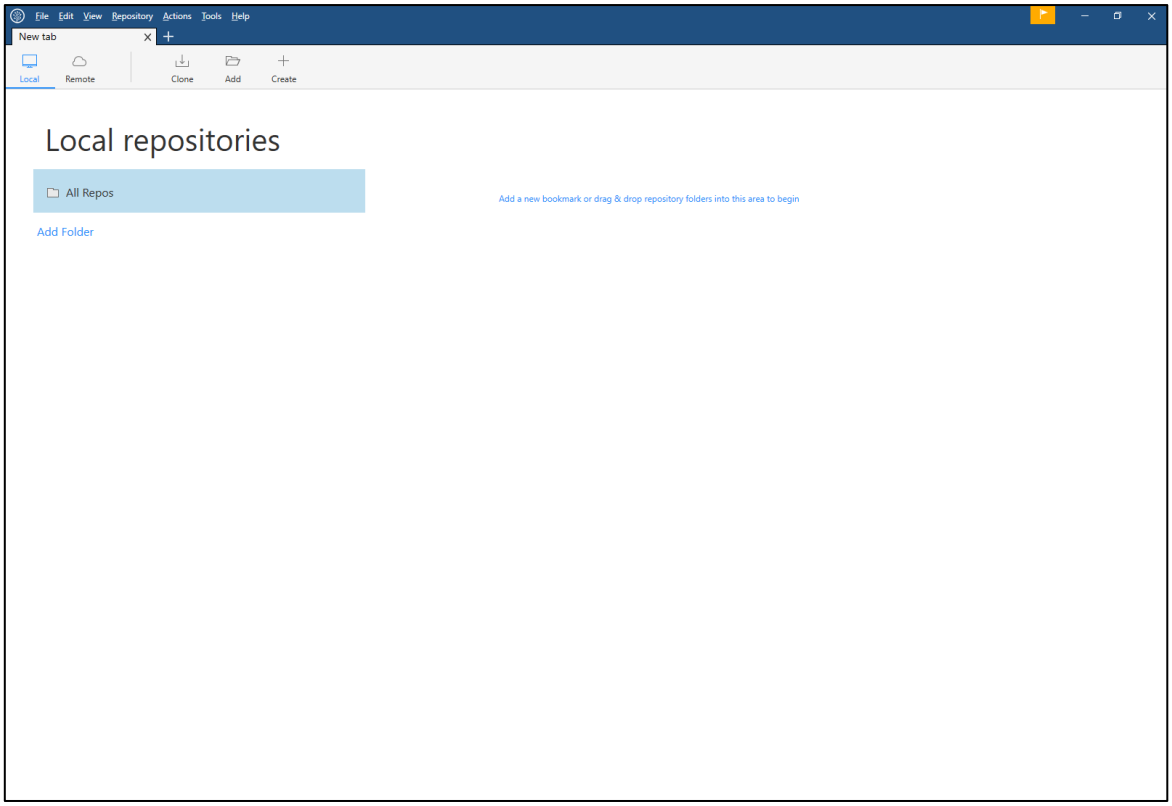
There seems to be an issue with the latest version of SourceTree. Please go to the Download Archive at the bottom of the page and then download version 2.1.10.0.

3.1 Software Installation

Download and run the installer paying particular attention to the following panels:

- **License Agreement** – Just agree.
- **Atlassian Account** – In order to use SourceTree, you will need an account with Atlassian. This is free.
    - **First install of SourceTree** - You will need to create an Atlassian account.
    - **Installed before** – Just select to use the account you registered before.
- **Remotes** – Select 'Skip Setup'
- **Load SSH Key**
    - **First install of** SourceTree – Select 'No'.
    - **Installed before** – Assuming you followed the instructions in this guide you will have stored the SSH keys in a location that can be accessed now.
      Select 'Yes', navigate to where the .ppk file is located and select 'Open'.
- **SourceTree: Git not found** – Unless you have already installed a version of Git on your computer – select 'Download an embedded version of Git for SourceTree'.
  Wait for this to install.
- **SourceTree: Mercurial not found** – We don't want to use Mercurial so just select 'I don't want to use Mercurial'

You will now have a SourceTree window open as shown below.


File Edit View Repository Actions Tools Help
New tab
Local
Remote
Clone
Add
Create
Local repositories
All Repos
Add Folder
Add a new bookmark or drag & drop repository folders into this area to begin

4 Creating a Repository from an existing project

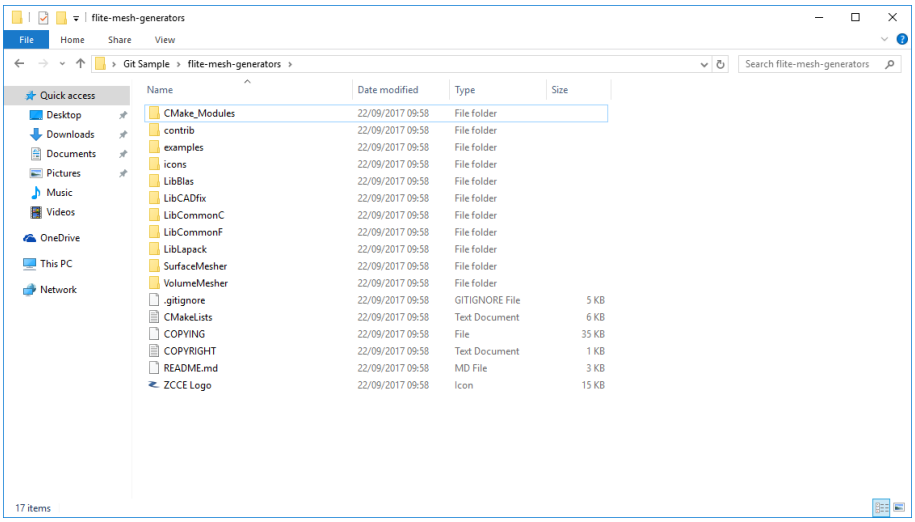
This section assumes you have a folder of source code files that you wish to add to a GitLab repository. It is important to note, at this stage, that there are numerous guidelines as to what should and should not be stored in a repository. However, in general, they can be summarised as follows:

- Files that are required to build your application should go in the repository.
- Files that are generated as part of the build process should not go in the repository

If your project is built in such a way that the object files and executables and stored in the same folders as your source code then you will also need a file called '.gitignore'. This contains a set of patterns that control which files Git will ignore (see Appendix A for more details).

	
COPYING, COPYRIGHT.txt, README.md files (see **https://www.swansim.org/guides/contributors-guidelines/** for samples)	Object files
Source Code	Executables
Build scripts (Makefiles, CMakeFile.txt)	Data files (unless they are used as part of a defined test suite)
Resources (such as images for web sites, GUIs)	
Test data (if used as part of a defined test suite but keep them small)	
Documentation	

We will be using the source code to the FLITE Mesh Generators available from SwanSim (https://www.swansim.org/products/flite-mesh-generation) to illustrate the procedure of adding a project to a Git repository.

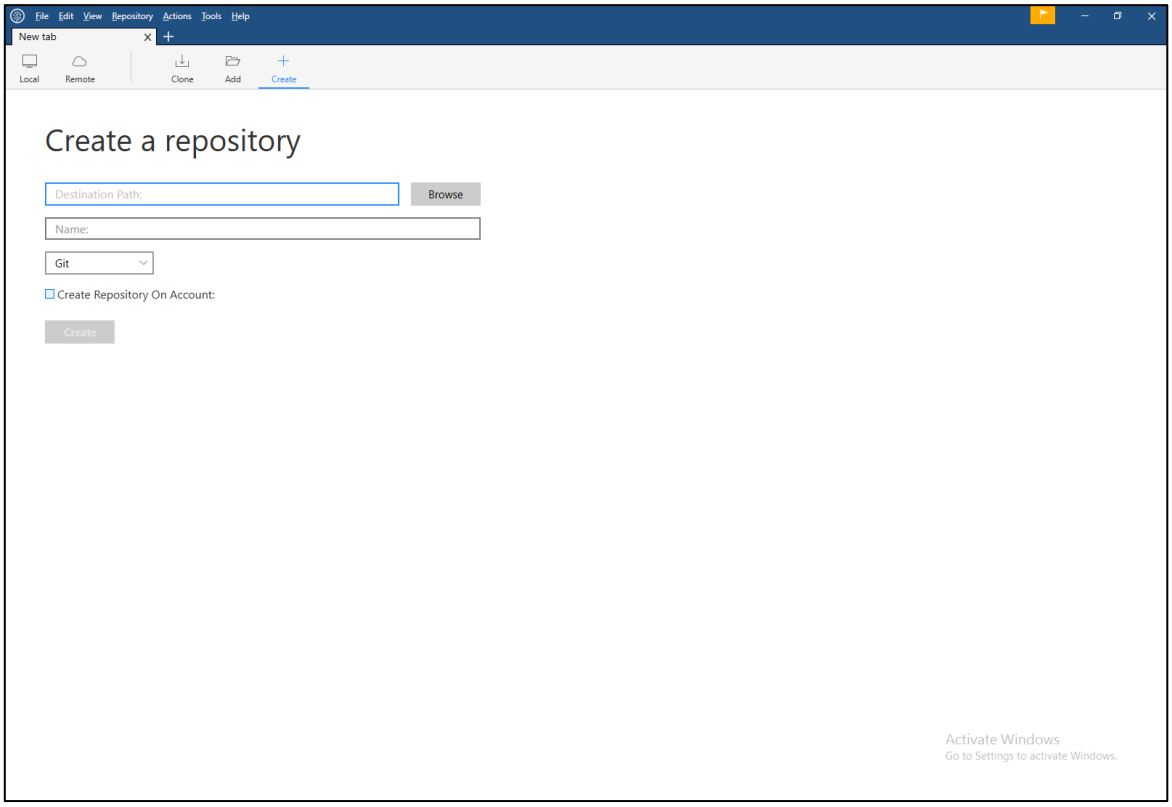


In this folder, the source code is split up into a number of sub-folders, it is built using CMake (see the CMakeLists.txt file) and it contains the COPYING, COPYRIGHT.txt and README.md files.

4.1 Working with a Local Repository

The first thing we need to do is create a local repository in our source code folder. This will be used to maintain a complete history of everything that was committed to the repository.

In SourceTree you need to click on the ‘Create’ icon located in the top toolbar. This opens a window as shown below:



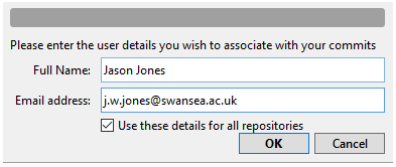
Here you need to:

- Navigate to the root folder of your source code.
- Give your repository a name (this will default to the last name of the folder you selected above but can be changed)
- Leave the repository type as ‘Git’
- Click ‘Create’

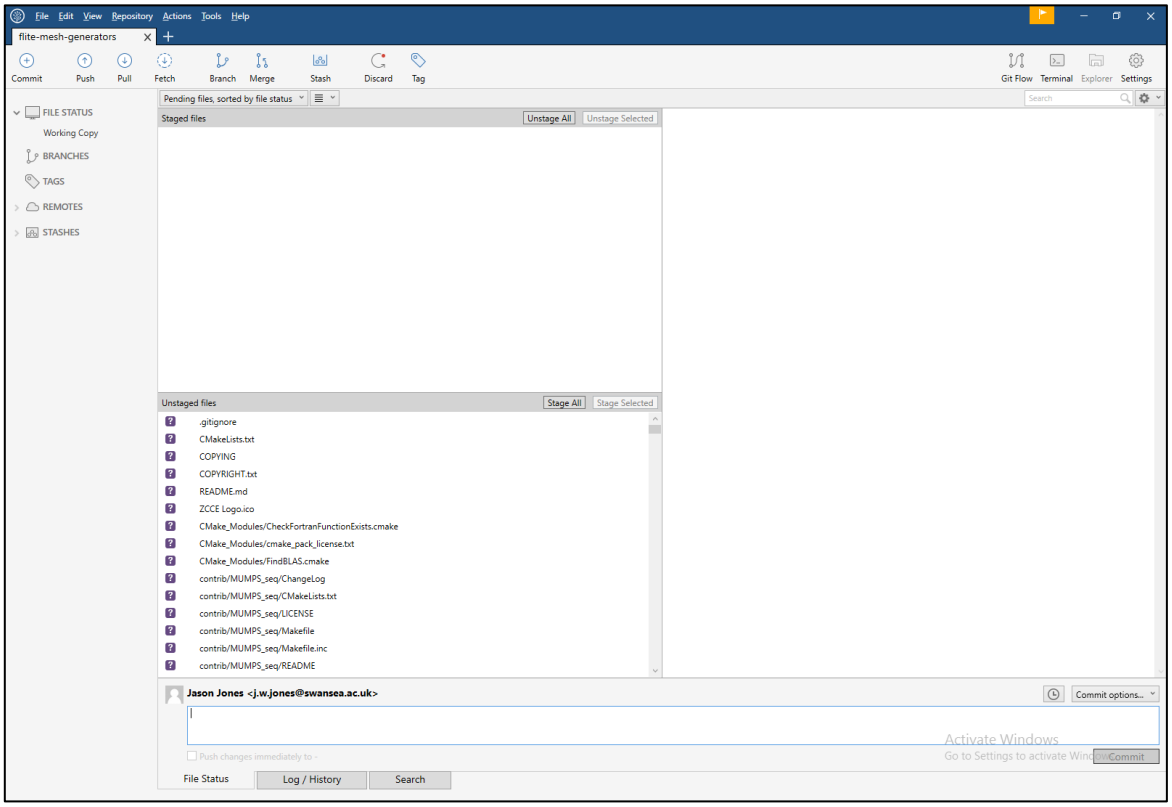
You will be notified that the folder you chose is not empty and asked if you want to create the repository there – select ‘Yes’.

You will then be asked for your details in relation to this repository – namely your name and your email address – as shown below. This will be stored in the repository and is used to track who made which changes when working in a team.

Leave the option, ‘Use these details for all repositories’ ticked so that you will not be asked for this information again.



Having completed that, you will be presented with a SourceTree window looking like this:



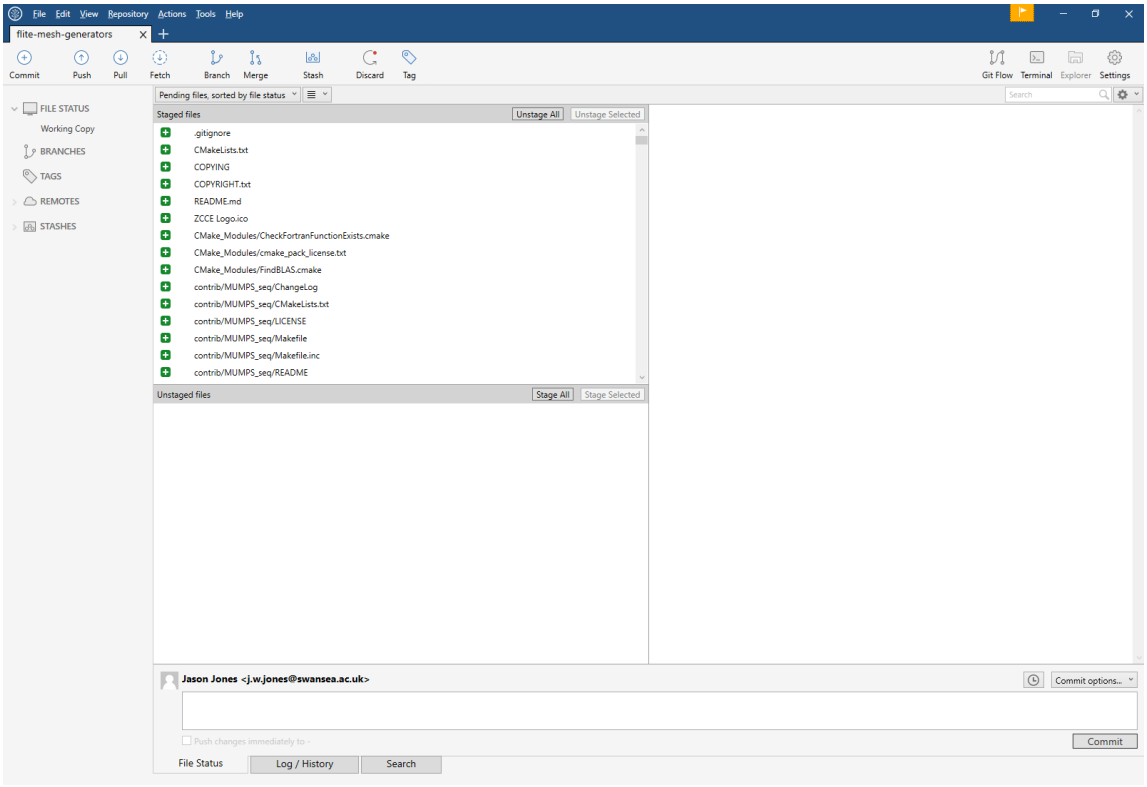
This panel is divided into three main panes of interest at this stage:

1. **Staged Files** – This lists the files that have been staged for adding to the repository. This means the list of files that will be added to the repository when the ‘Commit’ button is pressed.
2. **Unstaged Files** – This lists the files that have changed since the last commit to the repository but have not been staged yet.
3. **File Contents** – The pane on the right will show the changes between the file you are staging and the same file in the repository. Clicking on any file in either of the first two panes will populate this third pane.

   As the repository is empty, this pane just shows the contents of the file as there is nothing in the repository to compare to.

4.2 File Staging

The first step is to *stage* all of our files and folders. You can do this by simply clicking on the ‘Stage All’ button. This will change the window to look like this.



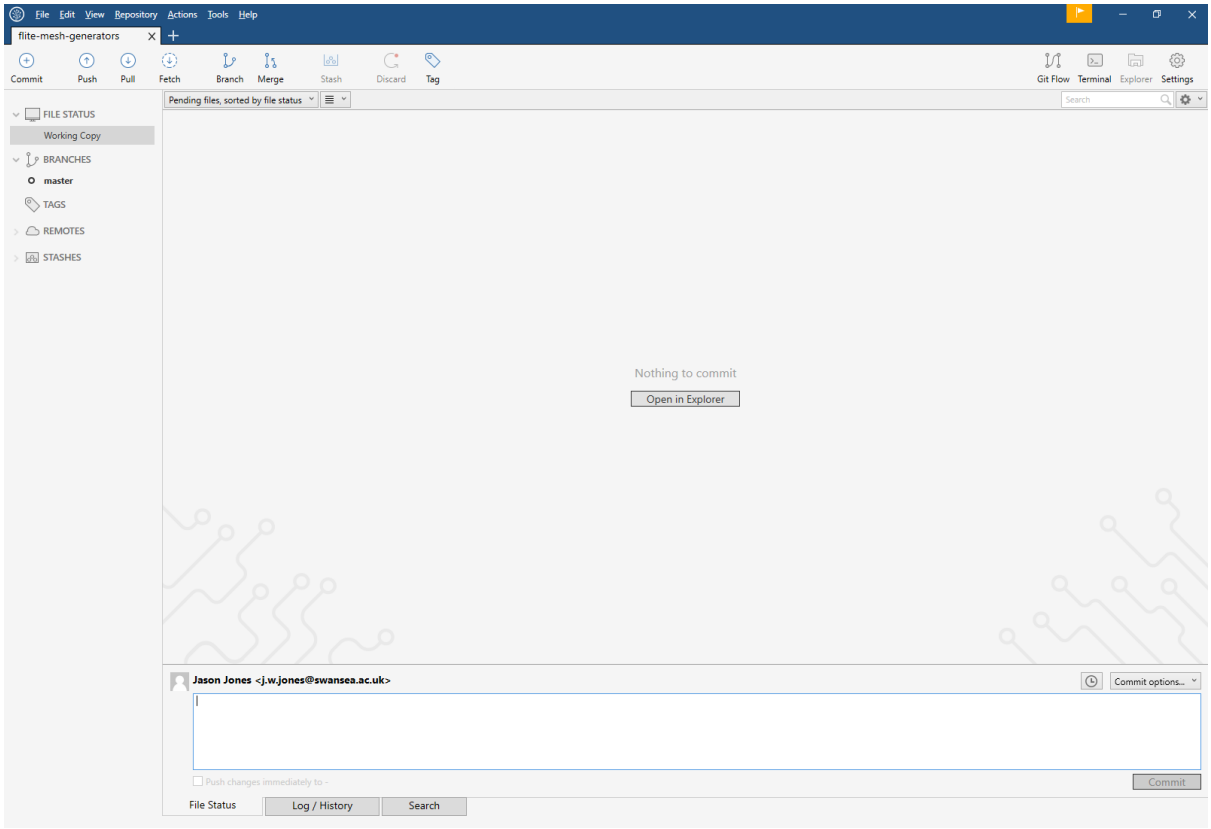
Here we can see all of the files and folders listed in the *Staged Files* pane.

4.3 Committing to the Repository

Once our files have been staged, we need to *commit* them to the repository. Every *commit* in Git must have a message associated with it. This message should be a summary of what changes were made since the last commit. Although this seems tedious, it will be extremely helpful when perusing the repository in months/years to come to find out when and who made certain changes.

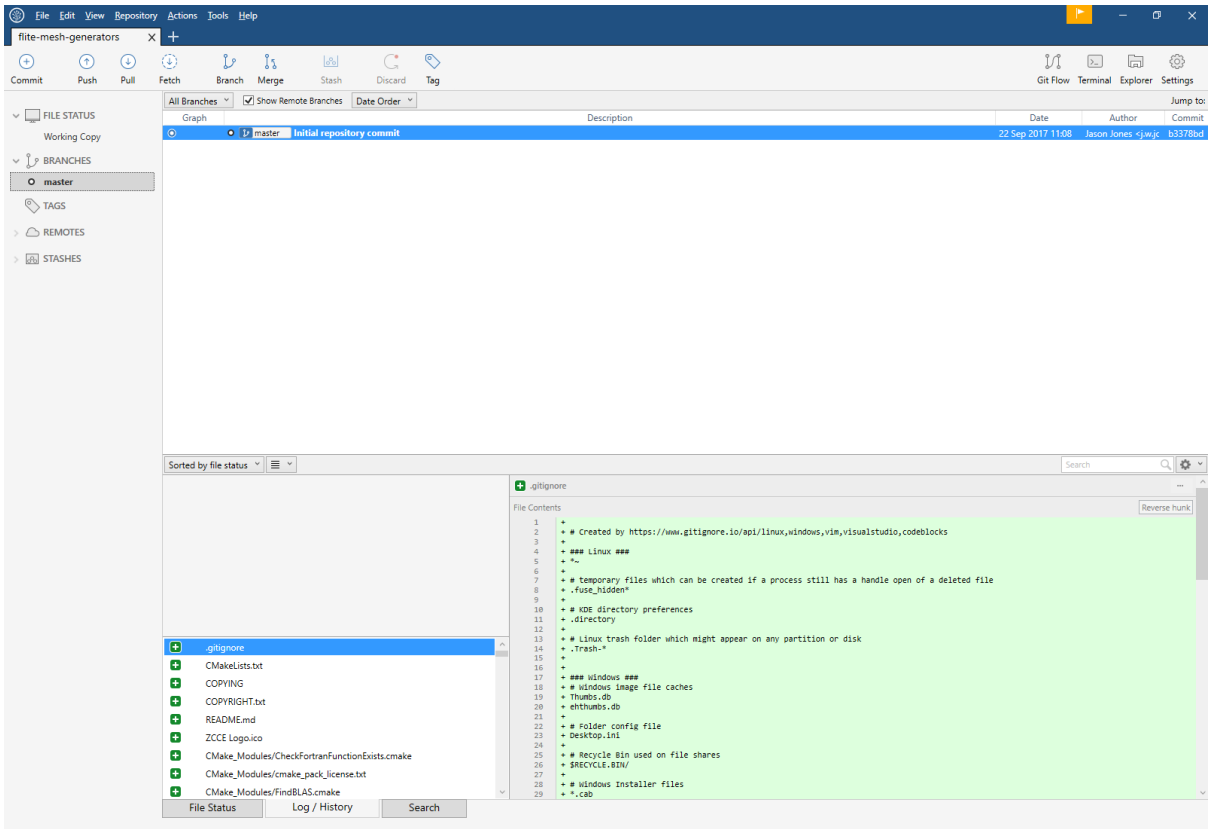
Since this is our first commit, a simple message like 'Initial repository commit' is sufficient – this is entered in the text box at the bottom of the window. Later when committing changes, listing the changes using a "-" character as the bullet point can be useful.

Once the message has been entered simply select the 'Commit' button. This creates a default branch 'master' in our repository (shown on the left). The branch on which we are currently working is always shown in **bold**. The three staging panes have been removed as there are no more files to be staged.



4.4 Viewing the history of our Repository.

The history and contents of our repository can be viewed by clicking on the 'master' branch on the left of the window. This changes the window to look like this.



This window is divided into four main panes:

1. **Repository History** – Here we can see the history of commits that have been made. Each entry in this list can be selected to show more details in the other panes. (for now there is just one)
2. **Changed Files** – This lists the files that were committed in the currently selected *commit*.
3. **Commit Info** – This displays some information about the commit. (For some reason this pane sometimes remains blank when there is just one commit in the repository).
4. **File Changes** – Selecting any file in the Changed Files pane, causes its contents to appear in this pane colour coded depending on how it differs from the previous version of the file.

4.5 Making changes in our Repository

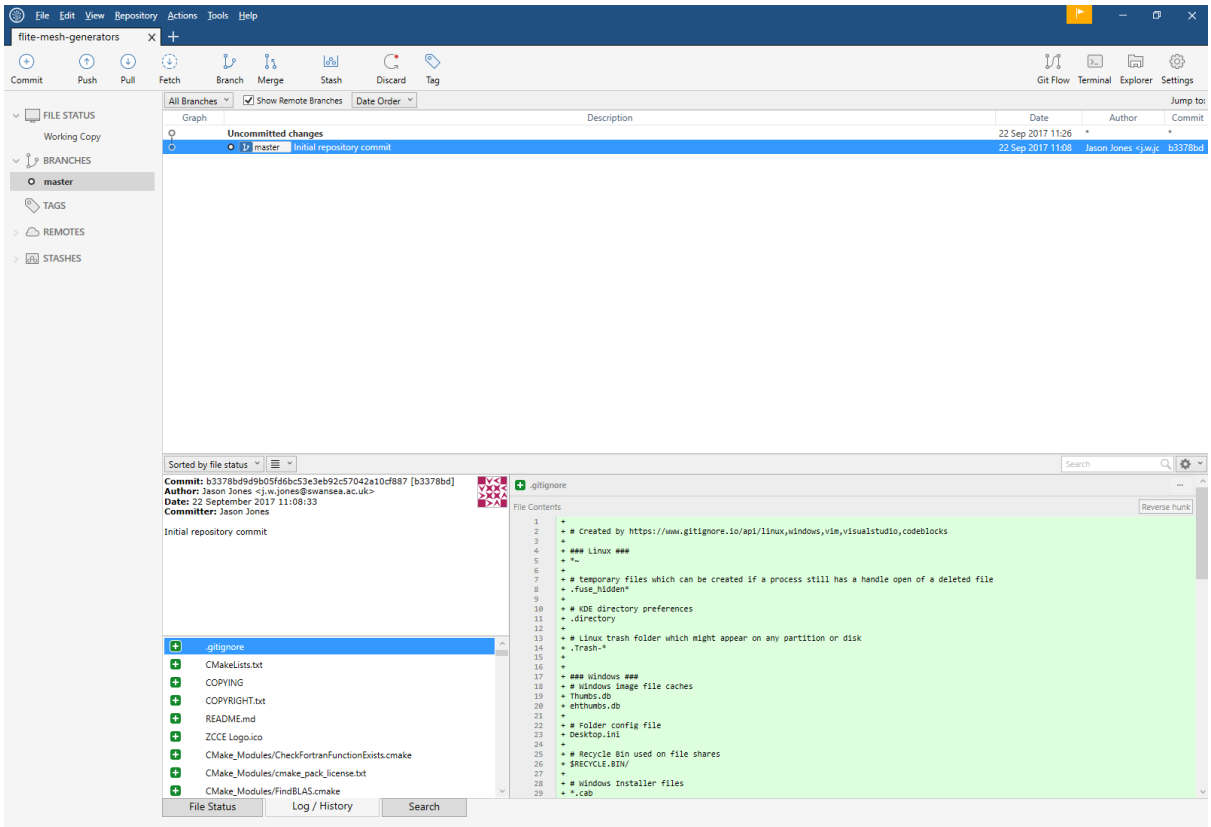
Now we have our repository set up with our source code committed, we will make some changes and commit those to the repository.

The changes we will make are:

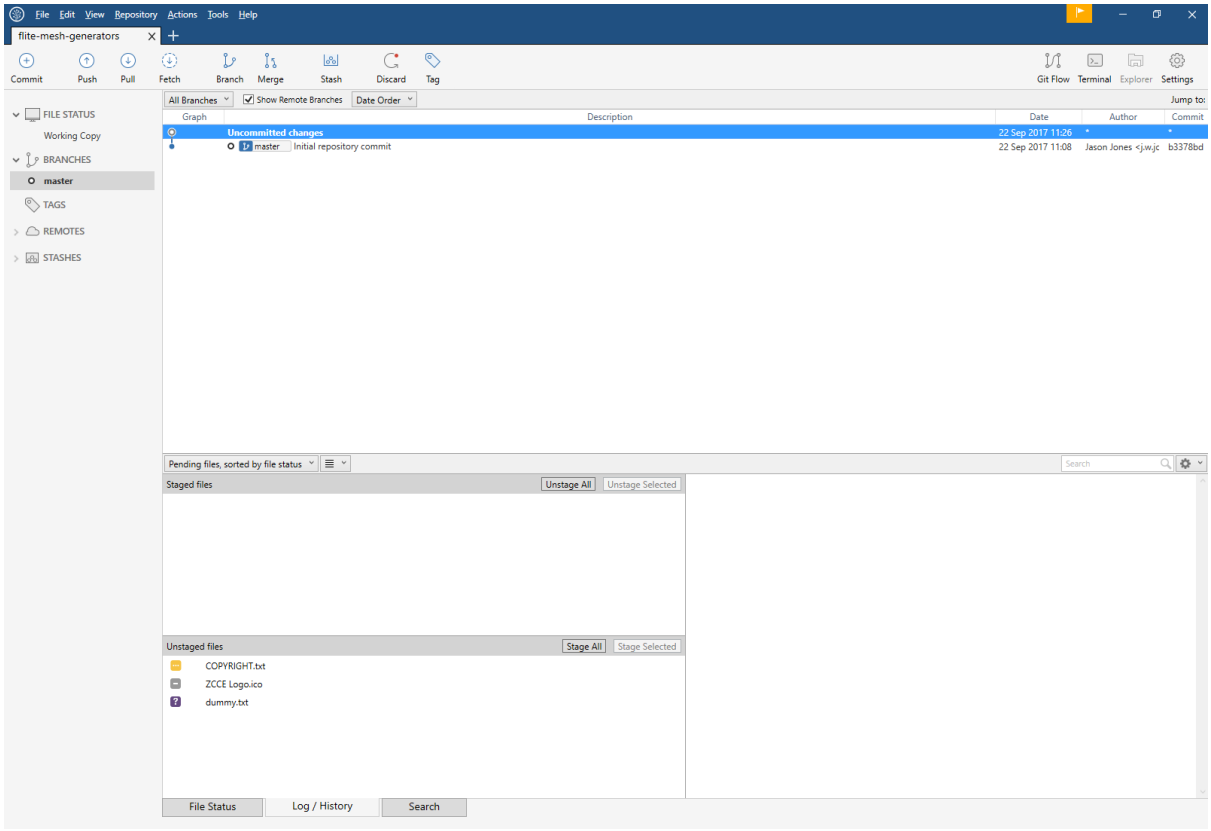
- Remove the ZCCE Logo.ico file.
- Add a new text file ‘dummy.txt’
- Modify the ‘COPYRIGHT.txt file’

These changes will all be performed using a normal Windows tools – in this case the File Explorer and Notepad.

Once these changes have been made, SourceTree will automatically detect them and update its window. (For some very large projects there may be a slight delay but this can be expedited by either using the ‘View’ menu and selecting ‘Refresh’ or hitting ‘F5’.



In the Repository View, we can see a new entry in our repository history – ‘Uncommitted changes’. Selecting this shows the files that have been changed but not *staged*.



There are two ways of staging our changes to the repository in SourceTree:

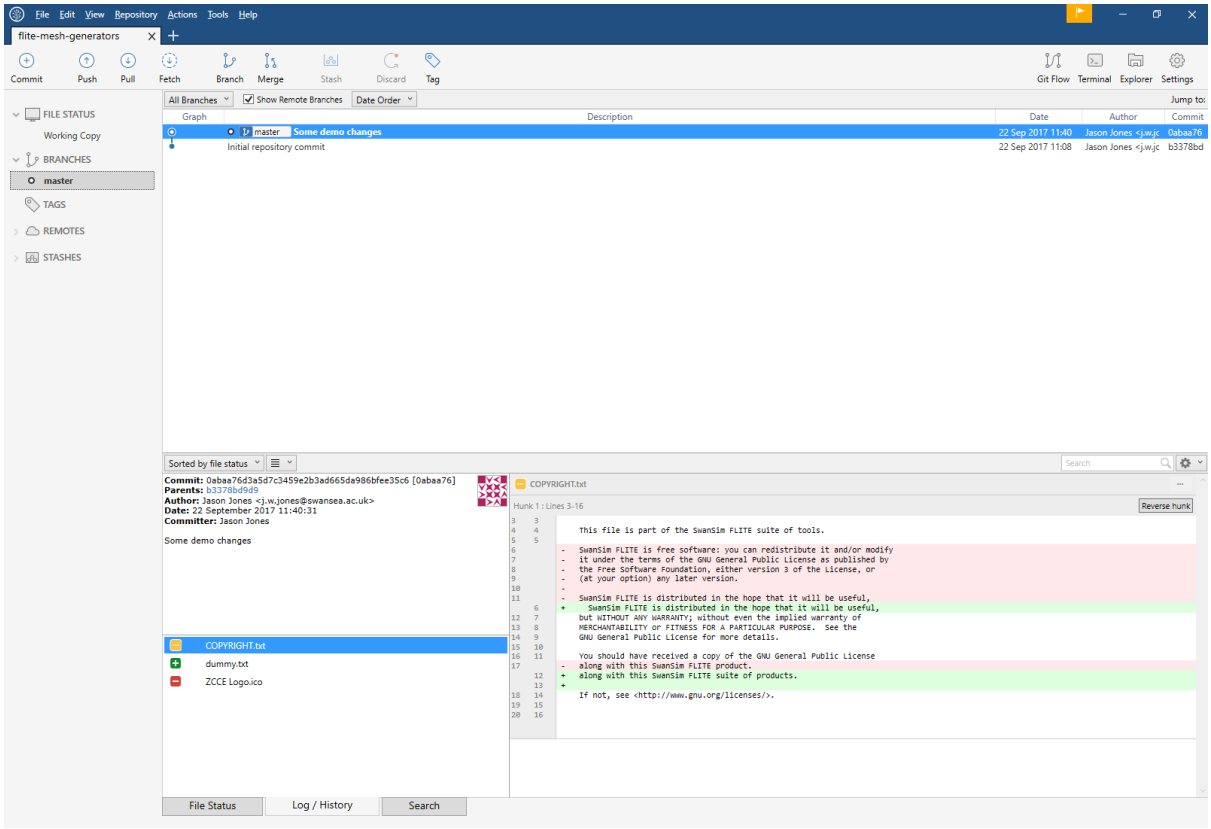
- Using the Repository view (the one shown above)
- Using the ‘Working copy’ view (the first SourceTree window shown in this Section)

They both perform identical actions and it is up to user preference which one to use.

Staging our changes using the ‘Repository View’ is performed by simply clicking on the ‘Stage All’ button. Clicking on the ‘Commit’ icon at the top of the window then takes you to the ‘Working copy’ window. Alternatively, the ‘Working copy’ entry in the pane on the left can be selected to take you to the ‘Working copy’ window and the staging of files can be performed there as described at the start of this section.

Regardless of which way this is performed, a message for the commit must be entered (in this case we will enter ‘Some demo changes’) and the Commit button is clicked.

If we move back to the ‘Repository View’ by clicking on ‘Master’ we can see two entries relating to the two commits we have made to the repository.


File Edit View Repository Actions Tools Help
flite-mesh-generators
Commit
Push
Pull
Fetch
Branch
Merge
Stash
Discard
Tag
Git Flow
Terminal
Explorer
Settings
FILE STATUS
Working Copy
BRANCHES
master
TAGS
REMOTES
STASHES
All Branches
Show Remote Branches
Date Order
Jump to:
Graph
Description
Date
Author
Commit
master
Some demo changes
22 Sep 2017 11:40
Jason Jones <j.w.jc
0abaa76
Initial repository commit
22 Sep 2017 11:08
Jason Jones <j.w.jc
b3378bd
Sorted by file status
Search
Commit: 0abaa76d3a5d7c3459e2b3ad665da986bfee35c6 [0abaa76]
Parents: b3378bd9d9
Author: Jason Jones <j.w.jones@swansea.ac.uk>
Date: 22 September 2017 11:40:31
Committer: Jason Jones
Some demo changes
COPYRIGHT.txt
dummy.txt
ZCCE Logo.ico
COPYRIGHT.txt
Hunk 1 : Lines 3-16
Reverse hunk
This file is part of the SwanSim FLITE suite of tools.
- SwanSim FLITE is free software: you can redistribute it and/or modify
- it under the terms of the GNU General Public License as published by
- the Free Software Foundation, either version 3 of the License, or
- (at your option) any later version.
-
- SwanSim FLITE is distributed in the hope that it will be useful,
+ SwanSim FLITE is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
- along with this SwanSim FLITE product.
+ along with this SwanSim FLITE suite of products.
+
If not, see <http://www.gnu.org/licenses/>.
File Status
Log / History
Search

5 Pushing to the GitLab Server

So far we have created a Git repository, committed our source code files and folders to the repository and then made some changes and committed those. This, however, has all been performed on the local PC, and the entire repository is stored within our source code folder.

(For the curious amongst you – the repository is actually stored in a folder '.git' in the root folder of our source code as a set of folders and files with very odd looking names. You may have to show 'Hidden files' in order to see this folder.)

In order to be able to share our repository easily with others in order to work as a team, or simply to have a central, backed-up area in which our repositories are kept, we will need to push our repository to the SwanSim GitLab server.

Before we can use the GitLab server, we need an account. If one has not already been created then please email Jason Jones (j.w.jones@swansea.ac.uk) for an account.

5.1 Setting up Authentication with GitLab

Logging in to the GitLab server (https://git.swansim.org) is performed by manually entering your username (or email) and your password into the log in page.

Git, on the other hand, uses SSH (Secure Shell) keys to perform the authentication without the need to enter any passwords. SSH authentication relies on a public and private key – the private key should be kept secret but the public key can be shared.

5.1.1 Creating the SSH keys

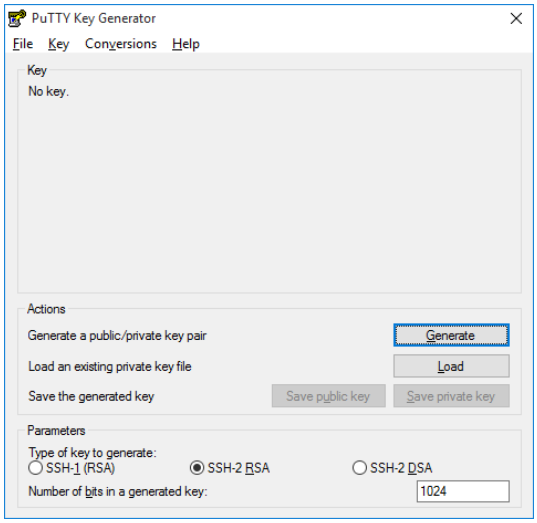
This is the most difficult part of the setup to get right. Luckily creating your SSH keys only has to be done once. It is then advisable to store these somewhere secure but easily accessible (e.g. Dropbox).

There are two main file formats used to store the public and private keys – Putty (the tool used by Windows users to access Linux) and OpenSSH (the standard Linux tools). SourceTree uses the Putty format, whereas GitLab uses the OpenSSH format. SourceTree has the ability to generate both formats but the steps are not the most intuitive.

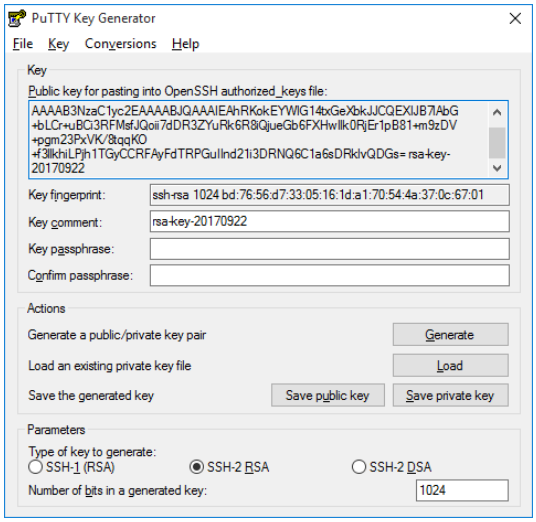
The files we are going to create are:

- 'id_rsa.ppk' – This is the Putty file format that stores the public and private keys in one file.
- 'id_rsa.pub' – This stores the public key in the OpenSSH file format.
- 'id_rsa.key' – This stores the private key in the OpenSSH file format.

In SourceTree, go to the 'Tools' menu and select 'Create or Import SSH keys'. This opens a window as shown below.



Click on the 'Generate' button. It asks you to move the move around inside the blank area of the window to create some randomness. When finished the window will look like this.



Creating 'id_rsa.ppk' – Click on the 'Save private key' and then navigate to the same folder and enter the filename 'id_rsa.ppk'. It will ask you if you are sure you don't want to save it without a passphrase – click 'Yes'.

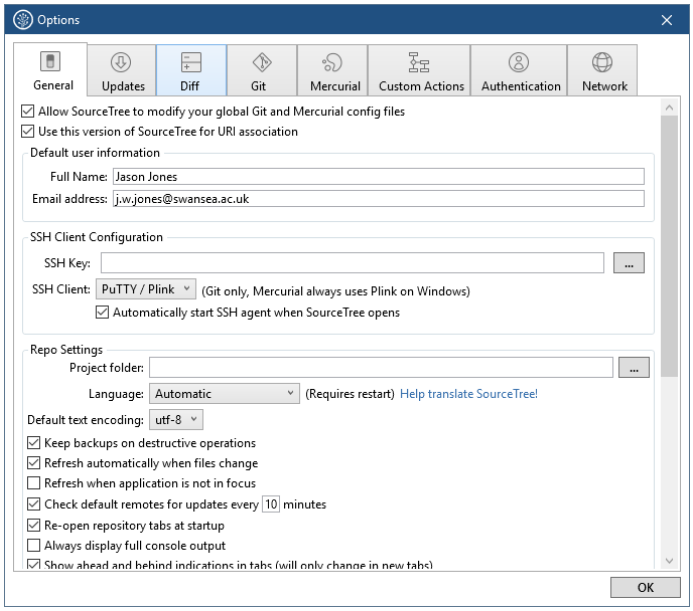
Creating 'id_rsa.key' – Select the 'Conversions -> Export OpenSSH key' menu option, navigate to the same folder and save it as 'id_rsa.key'.

Creating 'id_rsa.pub' – Create a text file called 'id_rsa.pub' using any text editor then select all the text in the 'Public Key' window at the top and paste it into the text editor and save the file.
(**Ensure you select every character of the text)**

5.1.2 Using the SSH keys

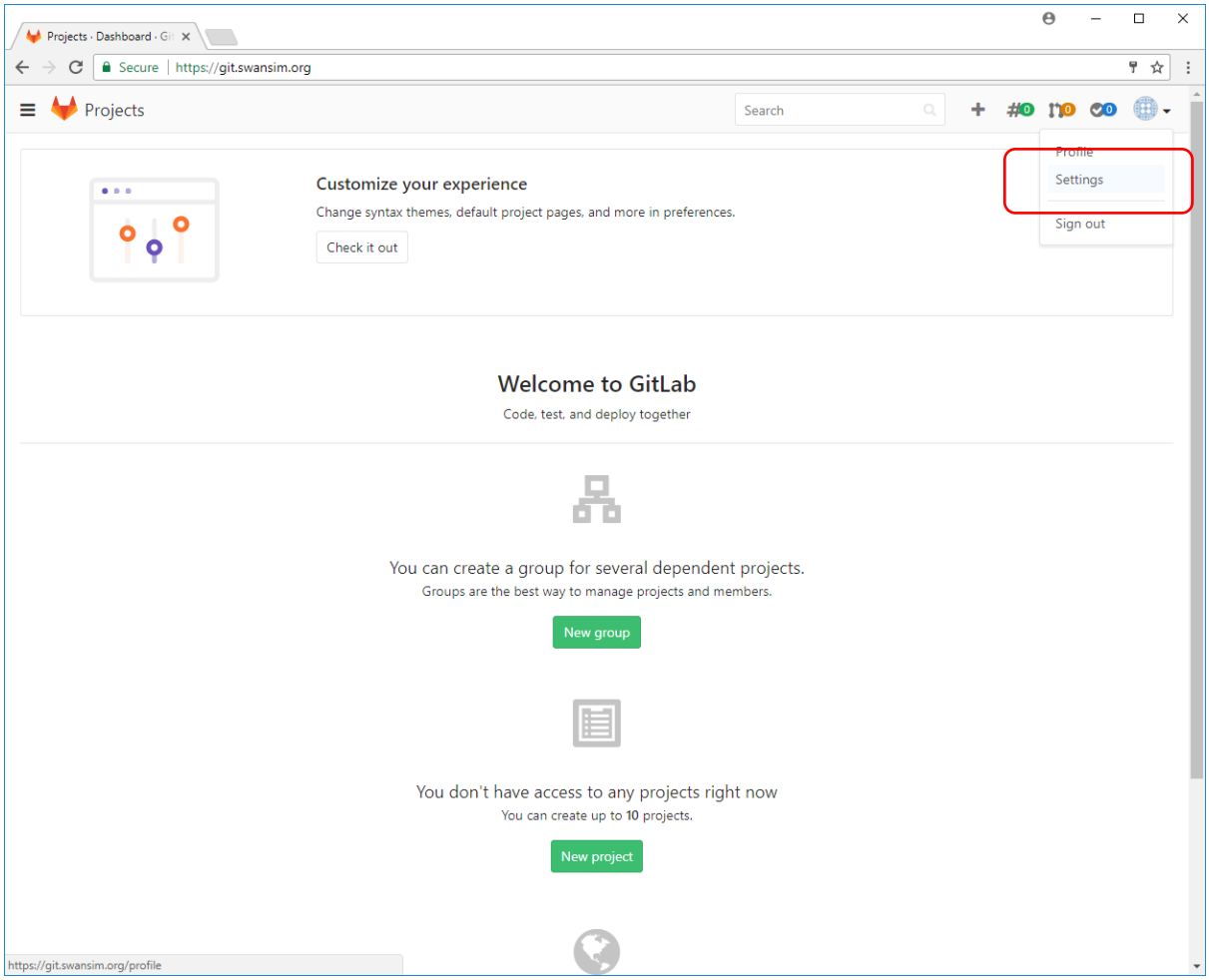
In order to use these keys that have just been generated, we need to tell SourceTree where to find the key to use and tell GitLab which key to trust.

In SourceTree, select the 'Tools/Options' menu item to get the Options panel.

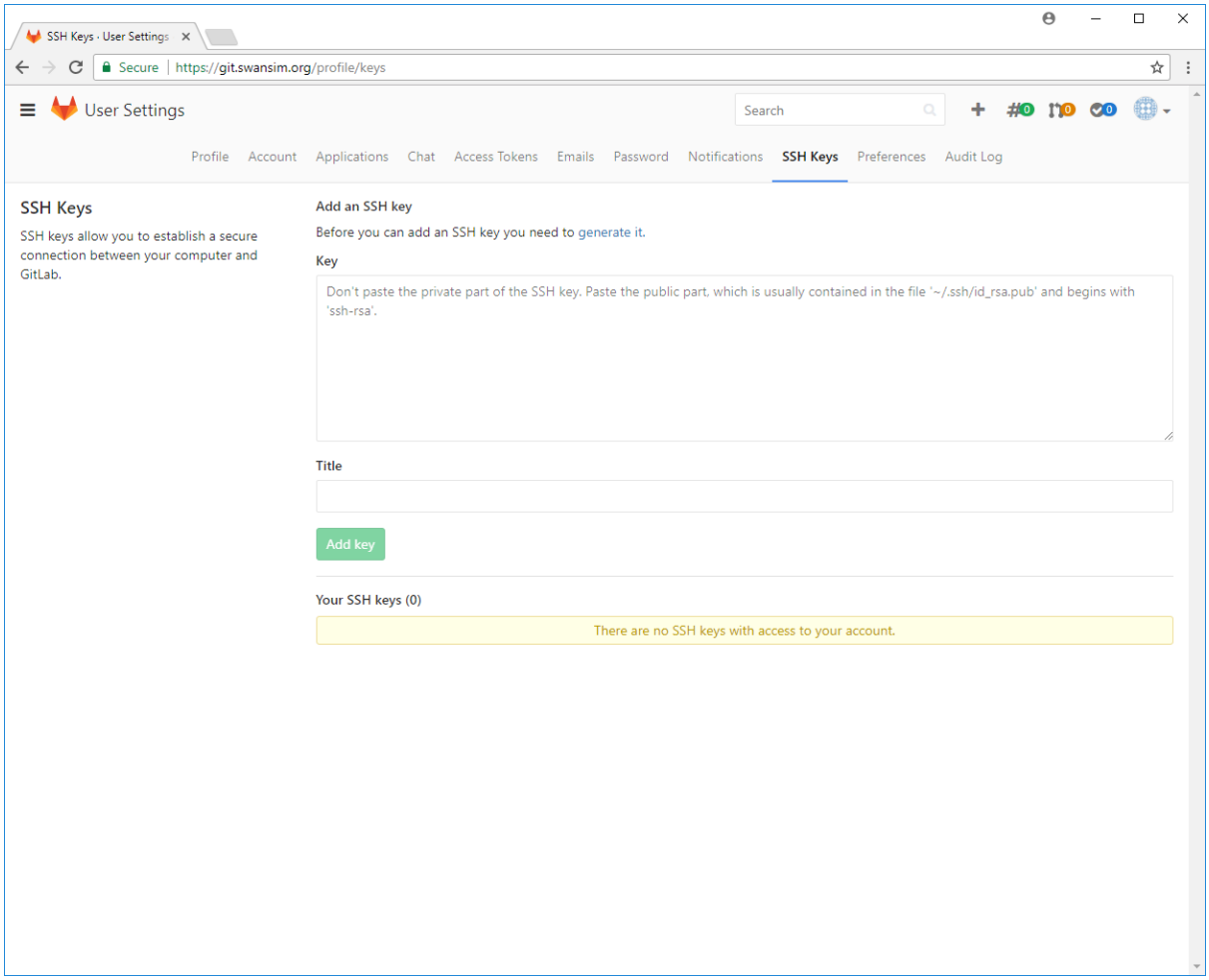


In the SSH Key box, navigate to the 'id_rsa.ppk' file saved previously.

Log into the GitLab server (https://git.swansim.org) and go to the 'Settings' menu as shown below.



And then select the 'SSH Keys' tab as shown below.



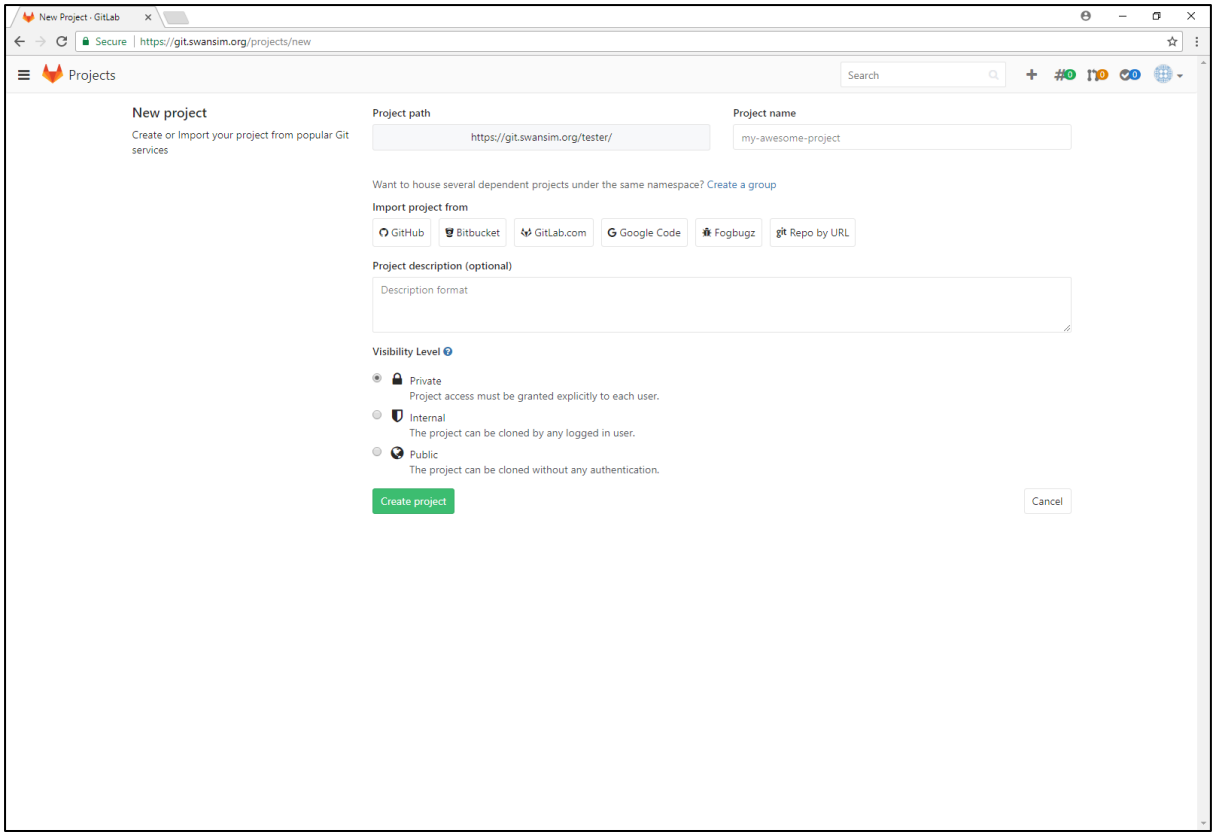
- Copy and paste the contents of the ‘id_rsa.pub’ file into the ‘Key’ box.
- Enter a Title to identify this key (for example, a combination of Your name and your PC’s name).
- Select ‘Add key’

5.2 Creating a GitLab Project

In order to store our repository on the GitLab Server, we need to create a project. Click on the orange icon in the top-left of the page to go back to our home page.

From this we need to create a new project that will store our repository. Each project stores one, and only one, repository. Groups can be created to organise projects if required.

Click on the ‘New project’ button to display this web page.



- Enter a project name and a description. (The name does not need to be the same as the repository name)
- For now, we will leave the project as ‘private’.
- Select ‘Create project’

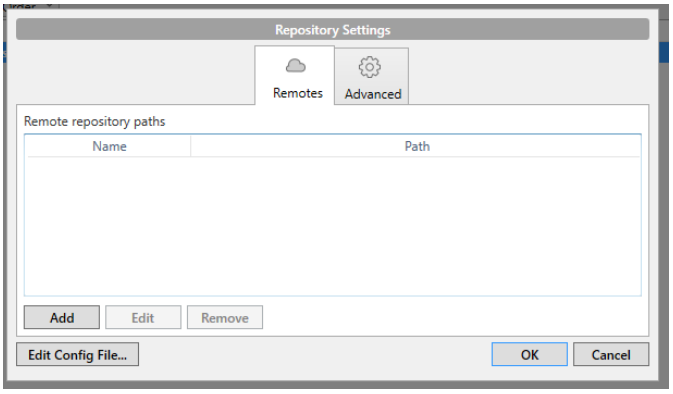
This takes you to the home page of your project.

On this web page are a number of sets of instructions to get going with the command-line version of Git. Since we are using SourceTree, we can ignore these.

The only piece of useful information is the URL near the top of the page. We will be entering this in SourceTree now.

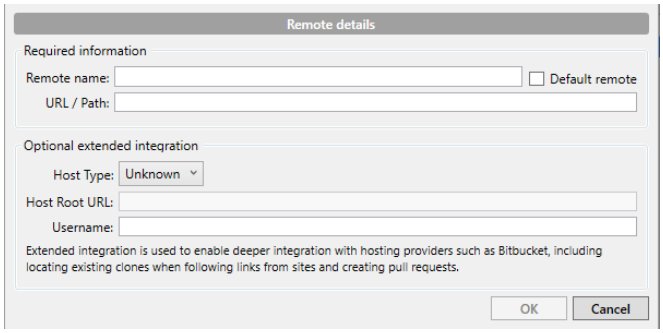
5.2.1 Linking our local repository to our GitLab project

Back in SourceTree, select the ‘Repository/Repository Settings’ menu item to open the window shown below.



In this window:

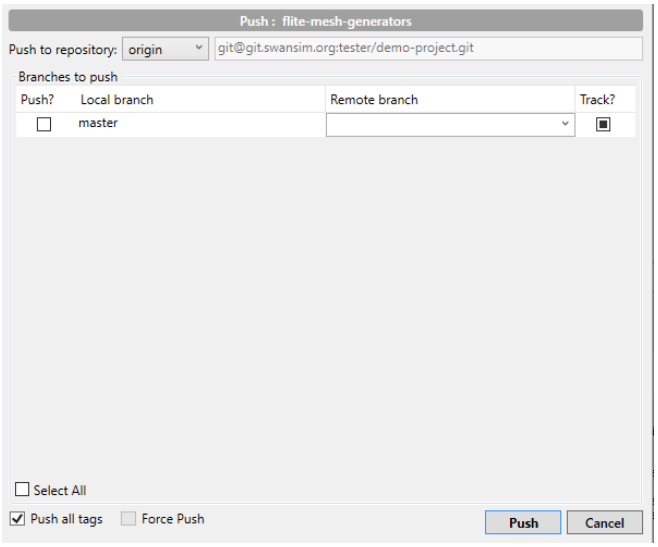
- Select ‘Add’



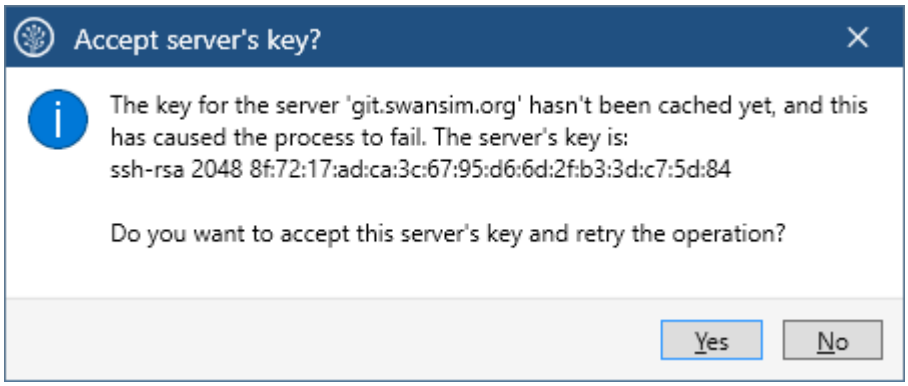
- Click ‘Default remote’. This populates the name with ‘origin’.
- Enter the URL from the GitLab project web page
- Leave the rest of the panel blank.
- Select ‘OK’
- Select ‘OK’ again to close the window completely.

5.2.2 Pushing our Local Repository to GitLab

Our repository can be *pushed* to the SwanSim GitLab server by selecting the ‘Push’ icon in SourceTree. This opens a dialog as shown below.



Since this is the first time we have pushed the ‘master’ branch, we need to tick that branch and then select ‘Push’. This is also the first time that SourceTree on this particular computer has contacted the SwanSim GitLab server so the following dialog pops up.



Click ‘Yes’ and then cancel all operations. And try again.

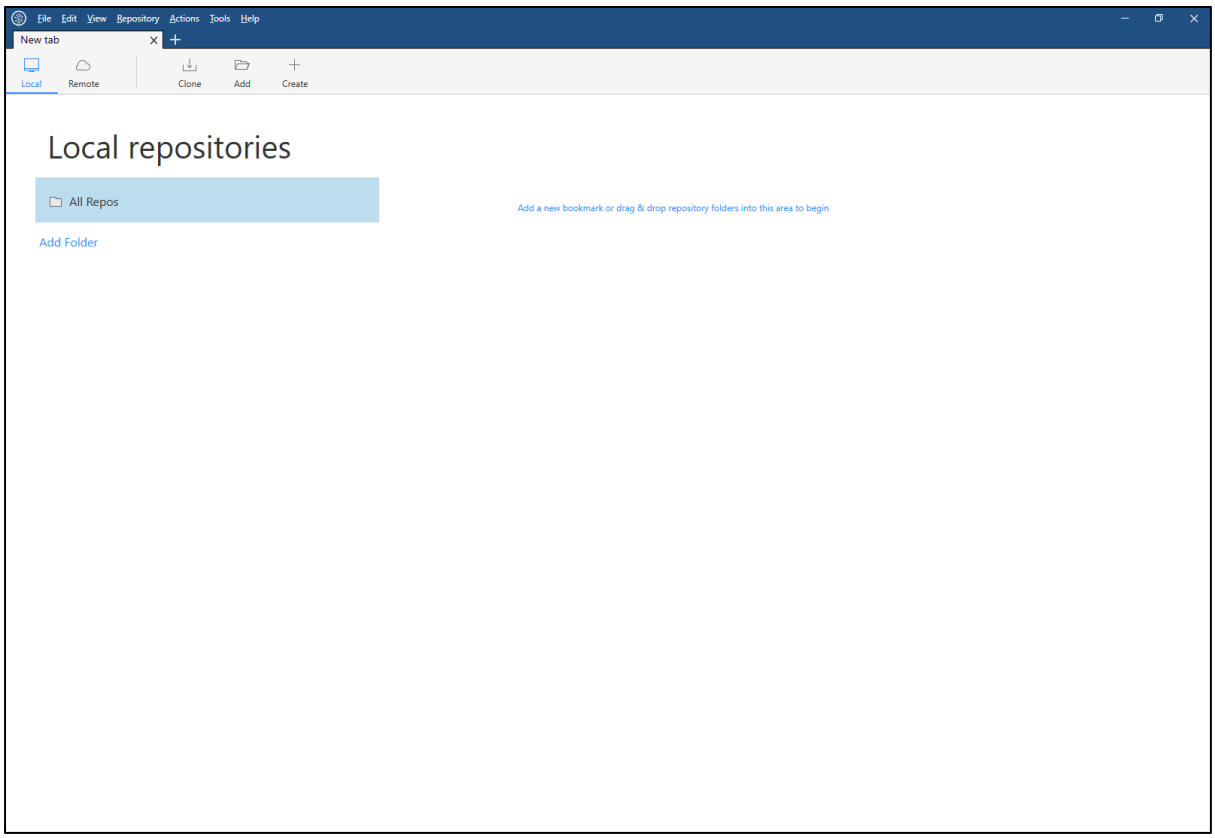
Sometimes retrying the operation fails again. The best thing to do is exit SourceTree and start it again. **This only ever happens the first time you use the SSH keys you have just installed. Creating further projects and adding repositories will work with no problem.**

Congratulations!! You have now pushed your entire repository onto the GitLab Server.

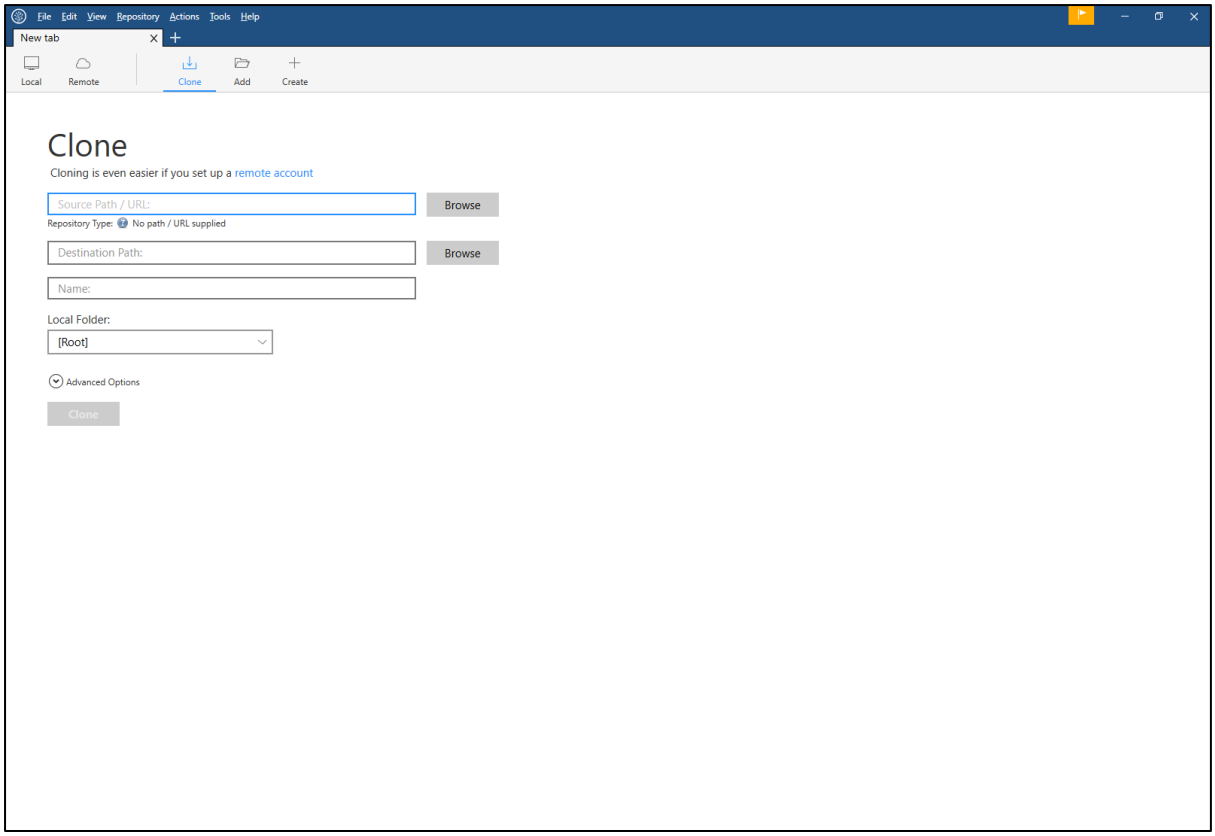
6 Cloning a GitLab Repository to my Computer

In this section it is assumed that SourceTree is installed on your computer along with the SSH keys in order to gain access to your GitLab repositories.

Starting SourceTree will show a window like below.



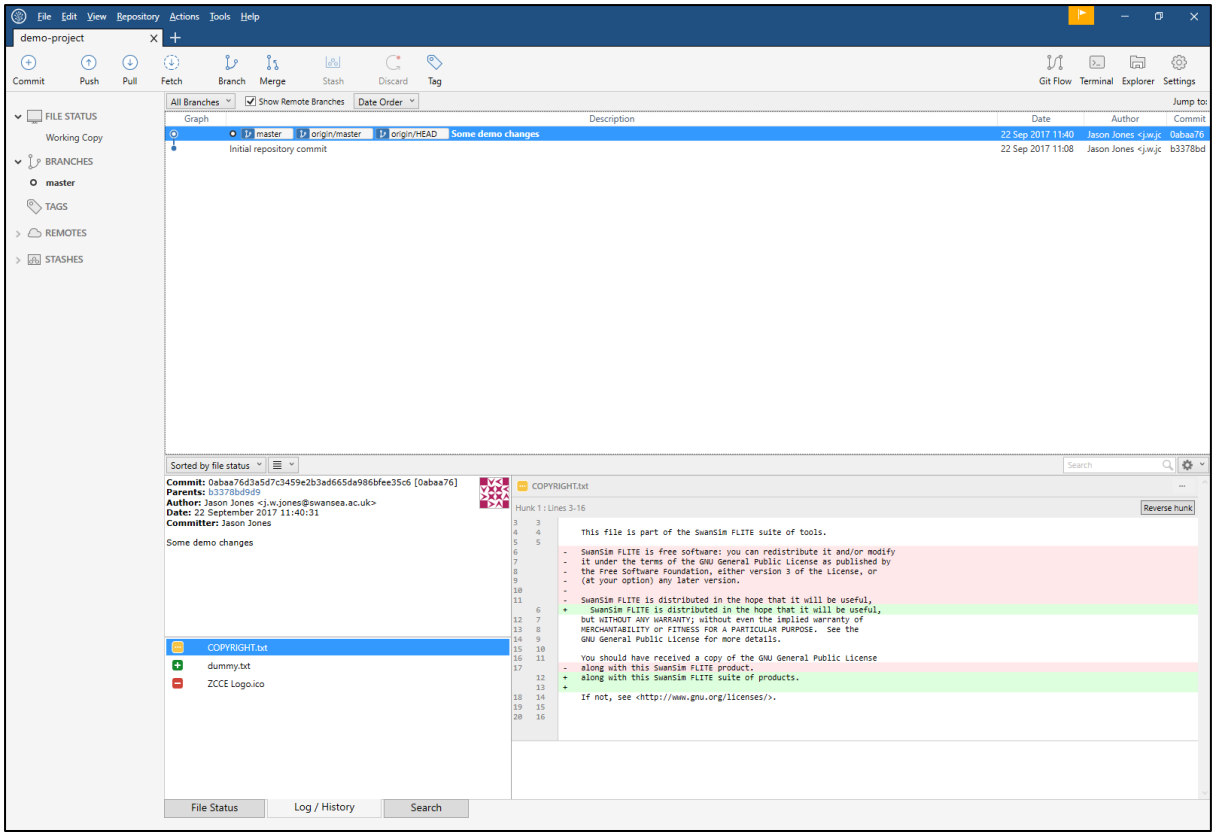
Select the ‘Clone’ icon.



In this window, enter:

1. The URL of the repository (This is available in the GitLab Project home page)
2. Browse to a folder in which you want to store your local copy of the repository

Select ‘Clone’ to copy across the entire repository from the GitLab server.



This can then be worked on in the normal manner and any changes can be pushed back to the server with none of the complicated setup required in Section 5.