Simple Introduction to Version Control

Contents

[What is a Version Control System? 1](#_Toc107912534)

[Some Basic Definitions 2](#_Toc107912535)

[Basic Git Commands 3](#_Toc107912536)

[Commit 3](#_Toc107912537)

[Push 3](#_Toc107912538)

[Pull 3](#_Toc107912539)

[Download Git Fork 4](#_Toc107912540)

[Introduction to GitHub 4](#_Toc107912541)

[Using Git Fork 4](#_Toc107912542)

[Cloning the APSIMStubble Remote Repository 4](#_Toc107912543)

[Committing Files to local repository 5](#_Toc107912544)

[Staging changed files 5](#_Toc107912545)

[Committing changed files 5](#_Toc107912546)

[Pushing Local Revisions to Remote Repository 6](#_Toc107912547)

[Project Version Control Guidelines 11](#_Toc107912548)

# What is a Version Control System?

Not surprisingly, a version control system is a system that maintains information about versions of a set of files for a project. (I’m sure you are now glad that you asked. Seems obvious really).

If you have ever kept versions of files manually using the following?

* renaming each version as version 1,2,3.
* Renaming with a date to know when you made the change
* Adding your name to show that you made changes to the file.
* Kept your own notes on what each version of your files included (e.g. “fixed an error in the weather data”)

A version control system will do all of this, and do it efficiently and effectively. For example, the version control system for APSIM next generation can show you every date of changes for every version of every file, as well as who made the changes and why. You can also view each one of these changes and, if you want, you can go back in time and grab a version from 10 years ago. If is also easy to perform a “diff” of individual files (e.g. place the files side-by-side to see what has changed).

Furthermore, your history of work can be stored on the cloud and so is automatically backed up and available to collaborators. Good version control processes are effective in allowing parallel development on individual files.

# Some Basic Definitions

There are complex definitions for version control using git, but we’ll use the following simple definitions for now.

Project – is a piece of work that consists of a collection of files.

Working Changes – changes to files within your project on your PC

Version – A project has a new version every time you decide to commit your changes to version control. (ie a revision is made).

Repository – holds history of every version (ie every commit) of the project (a revision history). Each version of the project is created from the revision history of files within the project.

Remote Repository – is an on-line copy of the revision history for the entire project. This is stored in GitHub, on the cloud. This is ultimately where all information needs to go.

Local Repository – is your local copy on your PC, where you can store all your revisions until you submit your revision history to the remote repository. Once this is done, the revision history of all work you did on your local repository is merged into the remote repository.

Tip Revision – is the most up to date version of the project.

Branch – it is possible to be working on two tasks for a single project (e.g. fixing two bugs) at the same time. You can keep these two pieces of work separate via having revisions on different branches of your revision history. We will deal with this later.



Figure Description of key Git commands to transfer information between working files between local and remote repositories.

# Basic Git Commands

There are a large number of commands you can apply to files and repositories within Git. Some are used regularly, and some less often. Some are simple and some are complex. Some are safe and some are powerful but dangerous.

For now, we will just cover the important commands that are relatively simple and safe.

## Commit

This adds a revision to your local repository. It does this by recording all changes to individual files and assigns a unique commit number. The date, time and your name is linked to these changes. The user also provides a description of the changes (e.g. “Fixes an error in the weather data”).

## Push

This pushes all revisions to a remote repository. Note: this includes the entire revision history, not just the latest version. This is basically a one-way sync with the remote repository. If the files on your PC were not up to date with the remote before you commenced your revisions, it will alert you, and your command will fail.

## Pull

This pulls down the revision history from the remote repository into your local repository. If others have changed files within the remote repository, their changes will be merged into your files automatically. If there is a conflict (ie to revisions of the same part of a file), you will be notified and given an opportunity to fix the conflict. It is best to follow good work practices to minimise the risk of this occurring.

# Download Git Fork

<https://git-fork.com/>

Fork is a simple Git Client application which allows you to run Git commands and visualise the state of your repositories.

# Introduction to GitHub

We have established a remote repository for this project called “APSIM Stubble” for short. The repository is stored on GitHub. Github is an online service owned by Microsoft. Repositories can be public or private. Our Repository is private and so each team member will be invited to join this repository.

Team members can join Github by establishing a log in at <https://github.com/>

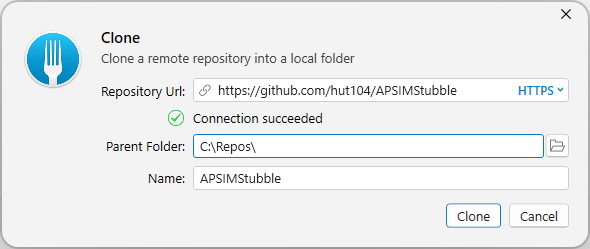
The repository is online at <https://github.com/hut104/APSIMStubble>

# Using Git Fork

## Cloning the APSIMStubble Remote Repository

Cloning downloads the entire revision history for the remote repository onto your PC to create a local repository. As a result, you will have a complete copy of all files within the tip revision on your PC.

To do this, you will need to provide the online address of the remote repository, a folder on your PC into which the local repository will be inserted (the parent folder), and a folder name to be used for the project files. It is usually a good idea to have a common folder for all local repositories, and to use the project name for the folder (ie be consistent).

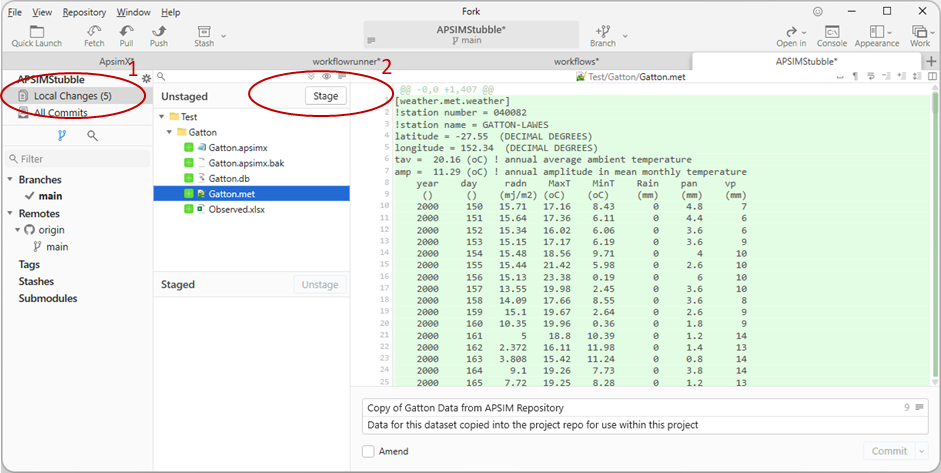


## 

## Committing Files to local repository

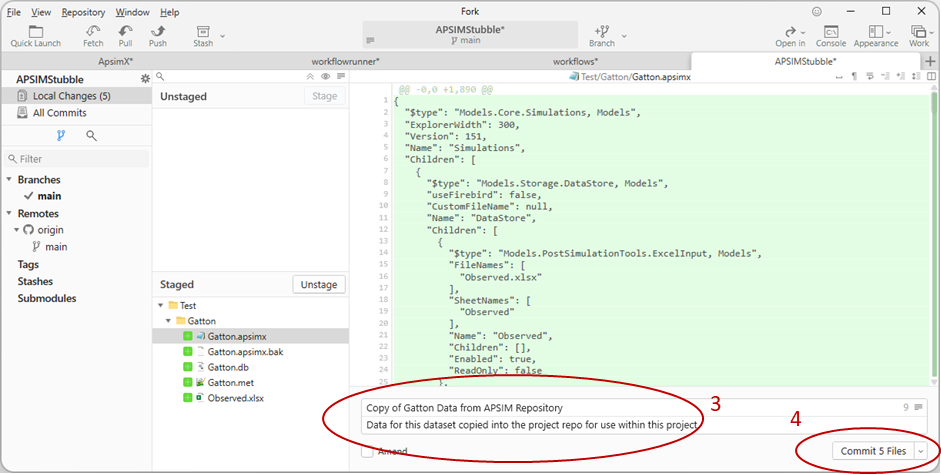
### Staging changed files

1. Click on “Local Changes” in your GitFork user interface to obtain a list of all files that have been changed, deleted or added.
2. Any of the given changes need to be “staged” to include them into a commit. Select each change and click the “Stage” button. The files should move into the “Staged” files list below. Incorrectly staged files can be “Unstaged” using the Unstage button.

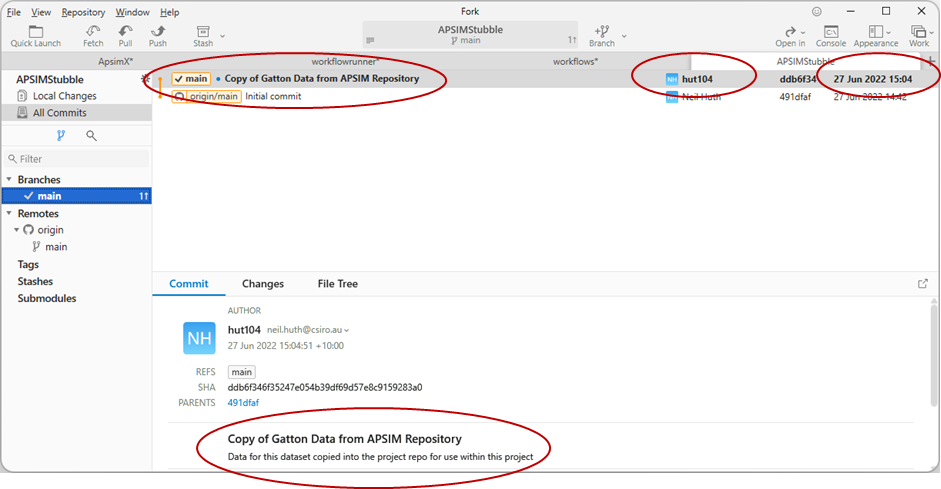


### Committing changed files

1. Add meaningful information for the commit message “Should be a short message”. A longer description of the changes can be added into the commit description.
2. Click the Commit button to add these changes into a revision within your local repository.



The new revision should appear within your revision history in the GitFork user interface. It should show you the commit message, the date of the commit and your github account name attributed to the revision.



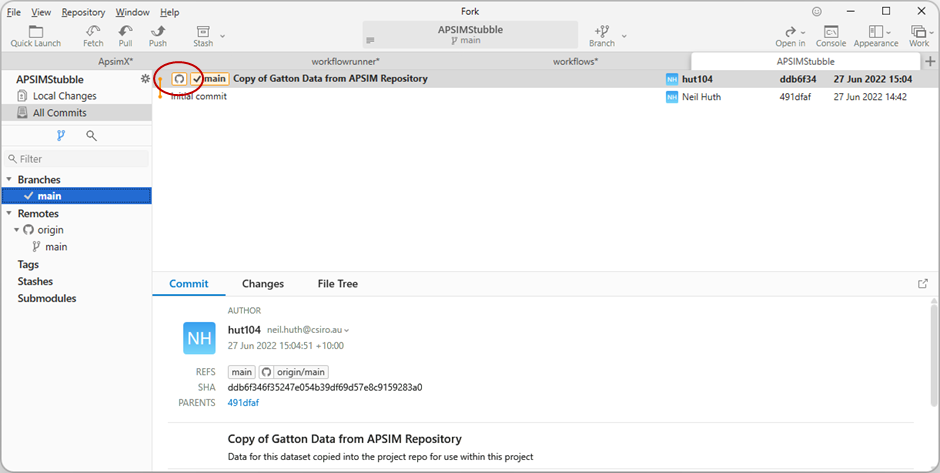
### Pushing Local Revisions to Remote Repository

1. Select the “Push” button on the GitFork ToolBar.
2. The following figure shows that we will push the main branch of your local repository into the remote repository.
   1. “Origin” is the official project remote repository. This is the standard naming convention for users of Git.
   2. The “main” branch on origin is the default branch name.
3. Click the push button.
4. If you get an error at this point, it usually means that you have not merged in all other changes prior to conducting your work. In this case, pull down all changes from the remote repository (See section on “Pull) and retry.

Graphical user interface, text, application, email

Description automatically generated

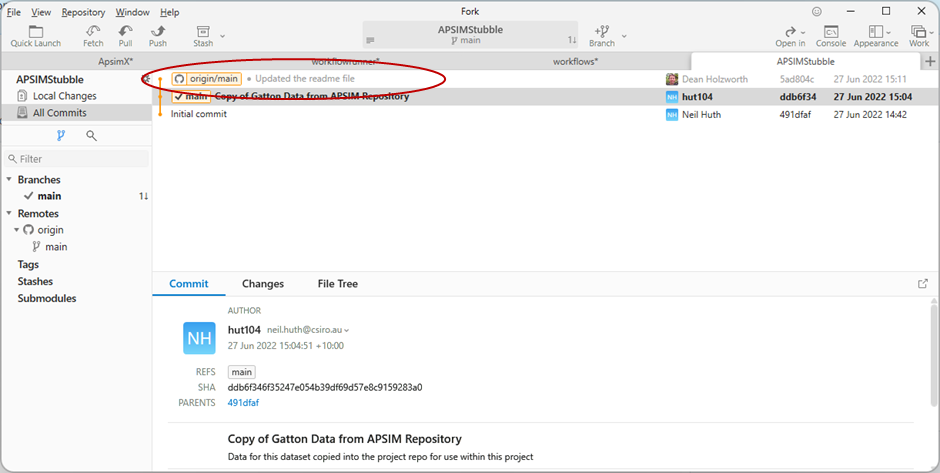
If you have successfully completed the Push to remote, the github tag should appear alongside your commit



## Pulling latest version from Remote Repository

You can pull in changes from other developers if they have pushed their changes into the remote repository. Any available changes will appear within the revision history, above your current version within the GitFork user interface. In the example below, Dean has pushed a change to the project after Neil’s addition of Gatton data. This can be pulled into the local repository using the “Pull” button on the GitFork Toolbar.

**NOTE: ONLY DO THIS IF THERE ARE NO LOCAL CHANGES. IF THERE ARE LOCAL CHANGES, COMMIT THESE FIRST.**



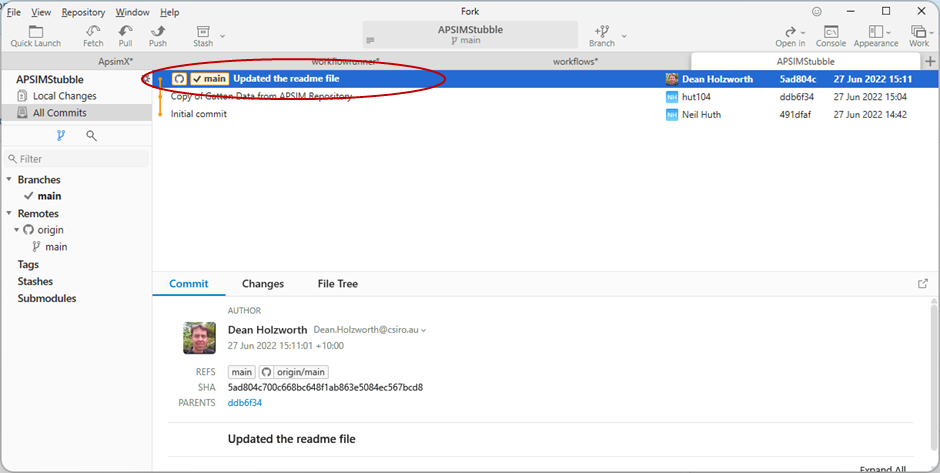
In the following figure, we are pulling the main (ie default) branch of origin (ie the remote repository) into our main branch of our local repository.

NOTE: NEVER CLICK THE TWO CHECK BOXES UNLESS YOU KNOW WHAT YOU ARE DOING.

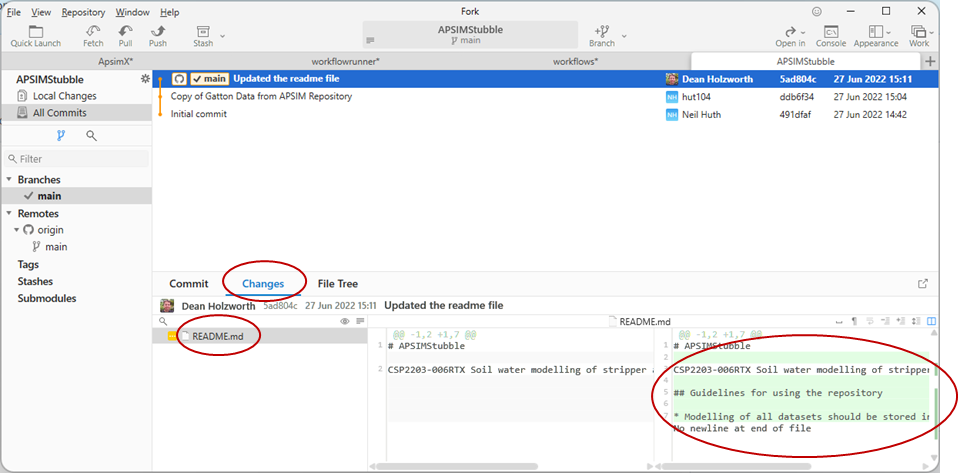
Graphical user interface, text, application

Description automatically generated

If you have been successful, the github tag should once again be on the same line as your branch tag (ie main).



You can click on “Changes” you can see the files that were changed by the revision. If you click on individual files, you can see the diff to that file.



# Project Version Control Guidelines

1. For now, separate discrete pieces of work into separate files and folders to minimise the risk of conflicts. (NOTE: Conflicts occur when two people edit the same section of a file at the same time)
   1. Each experiment will be a separate apsimx file to start with.
   2. Each experiment will store observed data in separate xlsx files to start with.
2. **EVERYDAY BEFORE YOU START WORKING**
   1. do a pull from the remote to ensure that you have the latest version of the project
   2. If you get an error message about a conflict when Git does the merge, call Dean or Neil first. It is likely that you have broken one of the rules.
3. **THROUGHOUT THE DAY AS YOU WORK**
   1. commit regularly as you complete a section of work on the files. This will allow you to roll back to a working version if you make mistakes.
   2. Use meaningful comments for each commit
4. **AT THE END OF EVERY DAY OR WORK EFFORT**
   1. Commit the last revision to your local repository
   2. Push the last revision to your remote repository
5. If you are about to work on a file usually worked on by another person, talk to them first to ensure they
   1. have pushed their version to the remote repository.
   2. Are not working on the same files at the same time
6. Do not change the naming conventions
   1. Observed data will be in spreadsheets called “observed.xlsx” with a sheet called “Observed”.
   2. The apsimx file will be named after the experiment (SiteName + Year + “.apsimx”)
   3. The met file will follow a similar convention.
   4. Use standard APSIM names (e.g. SurfaceOrganicMatter.Wt) wherever possible (ie minimise the use of aliases)
      1. for output variables in APSIM
      2. In the observed data within Observed.xlsx