# Using Simio with Git

Last Update: July 2024

Contents

[Using Simio with Git 1](#_Toc172011314)

[Overview 2](#_Toc172011315)

[Git and the Simio Language 3](#_Toc172011316)

[Git Overview 5](#_Toc172011317)

[Some Git Terminology 6](#_Toc172011318)

[Using Git with Multiple Developers 6](#_Toc172011319)

[Prerequisites: Simio Version and Recommended Tools/Apps 8](#_Toc172011320)

[Required Simio Editions 8](#_Toc172011321)

[Recommended Tools 8](#_Toc172011322)

[Git For Windows 8](#_Toc172011323)

[Notepad++ 8](#_Toc172011324)

[Visual Studio Code (VS Code) 8](#_Toc172011325)

[Example Project - SchedulingBicycleAssembly 9](#_Toc172011326)

[Preparing for First Use 9](#_Toc172011327)

[Setting up the Remote (Shared) Git Repository 12](#_Toc172011328)

[Setting up a GitHub Repository 12](#_Toc172011329)

[Doing the Initial Push to GitHub 13](#_Toc172011330)

[Adding Alice 18](#_Toc172011331)

[Back to Batpox 20](#_Toc172011332)

[Summary and the Road Ahead 21](#_Toc172011333)

[Appendix – Simio Project File Details 22](#_Toc172011334)

[Appendix – First Use In GitHub with VSCode 26](#_Toc172011335)

[The GitIgnore File 27](#_Toc172011336)

[Appendix - The Concept of theProject Folder 29](#_Toc172011337)

# Overview

This document describes how to use Git with Simio. It shows how to use world’s most popular version control software (Git) with Simio. This allows multiple engineers to work on the same Simio project with improved coordination and the benefits of source backup and documentation.

This document was created in response to a frequent issue – both internally and externally at Simio - about how multiple developers can best work on the same Simio project. Although Simio does not yet have a fully integrated version control system, this document demonstrates one way that Simio can be used with one version control product (Git). Git was chosen because it is currently the most popular solution and it is the one that is used internally at Simio.

There are a few things worth mentioning.

The first is that versioning software is complicated by the nature of the problem(s) that it is trying to solve, and Git is no exception to this rule. Even with tightly integrated IDE’s (Integrated Development Environment) such as Visual Studio, a team of developers can find themselves in a scrambled state if workflow rules and conventions are not followed. A rule that we have found useful at Simio is to keep the workflow simple and consistent and check the state of the repository often.

A second point is that there are advantages to using Git even for a single developer. Git can keep track of all changes (which includes human descriptions for the changes) as well as allow the developer to revert to previous development points or even create separate branches for different clients. It also can provide continuous backups to a remote location. These points are not covered in detail in this document, but a huge amount of Git resources is available on the internet.

Finally, Git is just a tool and only part of the team development problem. Building a Simio model is no different than building any piece of software; As a Simio model becomes more complex, the rules governing software complexity become more important. Techniques such as separating out independent software components (e.g. Library objects) for development and testing become more important. Additionally, work-items such as process and dashboards - that are less likely to intersect - can be assigned to different people.

For demonstration, the example project that is included with each Simio desktop installation “SchedulingBicylceAssembly” is used. This is a relatively complex model where the benefit of multiple developers might be seen.

You will see in the example how Git, Simio, and VS Code can be combined to provide a SCM (Source Control Management) environment for Simio Projects.

## Git and the Simio Language

Most users of Simio never see or interact directly with the underlying Simio language, which is a declarative language implemented a hierarchical series of XML files. It is rarely seen because Simio’s rich UI provides a visual object-oriented view of the project and handles the details of those files for the user. What the Simio user ***is*** familiar with is an .SPFX file, which is really a compressed file containing all these files.

However, source control management (SCM) systems need direct access to those files and for this purpose Simio provides a storage option where the files are made available to the SCM. Additionally, there are Simio properties that instruct Simio to make changes so that the storage is more suitable for SCMs.

To do this you need to set two project-level properties in Simio (by right-clicking on the project in the Project Pane:

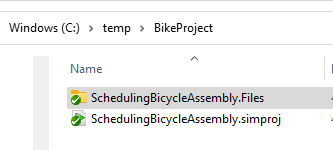
1. Save Project as Multiple Files (leave as True)
2. Support Source Control (set to True)

And then save the file as format (.SIMPROJ).

Graphical user interface, text, application, email

Description automatically generated

The result for the ScheduleBicycleAssemblies – when place in a folder called “BikeProject” - will look like this in File Explorer:



And expanding on the folder “SchedulingBicycleAssemly.Files” reveals the structure of the Simio project:

Graphical user interface, text, application

Description automatically generated

Drilling all the way down to Processes shows an example of the XML files:

Graphical user interface

Description automatically generated

It is the XML files that represent the Simio model which can be altered by the user in the Simio UI, and therefore be employed in an SCM like Git.

# Git Overview

From Wikipedia:

**Git** is a [distributed version-control](https://en.wikipedia.org/wiki/Distributed_version_control) system for tracking changes in [source code](https://en.wikipedia.org/wiki/Source_code) during [software development](https://en.wikipedia.org/wiki/Software_development). It is designed for coordinating work among [programmers](https://en.wikipedia.org/wiki/Programmer), but it can be used to track changes in any set of [files](https://en.wikipedia.org/wiki/Computer_file). Its goals include speed, [data integrity](https://en.wikipedia.org/wiki/Data_integrity), and support for distributed, non-linear workflows.

Git is used extensively by Simio’s internal developers to build Simio products, but it can also be employed by Simio users when building and maintaining Simio Models.

There are many great tutorials on Git available from the Internet, so here we shall just cover the simplest and basic features.

A good background book on Git is Ry’s Git Tutorial by Ryan Hodson. It is not only a tutorial, but a good reference book and is free on Amazon. It is particularly useful if you like visual aids for the workflow.

Git is very file oriented and has the concept of a “repository” of files that need to be maintained. For a given project, each developer has:

* First, a working directory, which contains the Simio source files that are being worked on.
* Secondly - to track changes - Git creates a personal ‘local’ repository that contains all changes made to these files, and then
* Thirdly, the changes from the local repositories can be “pushed” back to a ‘remote’ repository that is the source-of-truth and shared by all developers of the project.

If a new developer wants to start working on the project, they clone their local repository from the remote repository and are given a working directory along with their very own local repository that contains the entire history of the project.

If you wish to try Git at the lowest level (assuming Git is installed), use the File Explorer navigate to the project folder, right click, and select “Git Bash Here”. Git “Bash” is a command level tool installed with Git that allows direct interaction with Git using git commands. In our example we will use a higher-level tool (VS Code) that will make the process simpler.

As previously mentioned, using Git is valuable even if there is only a single developer as your entire building process and history is documented and recorded.

There is a serious caveat though; using Git (or any SCM) requires rigor and discipline on your part. You cannot simply use it sometimes and ignore it at others. Although the tools will force you to enter information when you commit, it is up to you to make sure that the text is meaningful to you and others.

## Some Git Terminology

This document assumes you have some Git knowledge. In other words, this is not a tutorial on Git.

When you are about to work on your Simio project, here is a standard workflow:

1. Pull a fresh copy from the remote repository
2. You make changes to your Simio project. This is your Working Directory.
3. When you have performed a group of related changes, you save your project in Simio, which forces the files back to the working directory.
4. You then commit those changes to the project history, which places them in your *local* repository along with a commit description that you add.
5. When you are ready, you push your changes to the remote repository.
6. If there are conflicts, you resolve those changes, and push again.

Note: you will often see the work “Origin”. This refers to the remote repository.

In practice, you may often combine steps 4 and 5 into a commit-and-push.

## Using Git with Multiple Developers

When you have multiple users, each user can independently make changes to the files and push them to the central repository.

This works fine until two people (say Alice and Bob) working with the same version of a file make changes and attempt to push them back to the main shared repository. The first user (Alice) can push it just fine, but then Bob tries to push it and gets an error that there is a conflict. Now what?

Clearly, the program does not know what to do, and so the conflict must be manually resolved by a human.

In Simio these “source” files are XML files and therefore contain coded information about the Simio objects (like Models).

If you use a Git tool (like VS Code) to resolve the conflict, you are provided with an editor that shows the differences between the files, along with ways to resolve those differences.

Of course, the easiest way to deal with conflicts is to not have any to begin with, and the best way to do *that* are:

1. Divide the work such that only a single person is working on a file at a time.
2. Communicate with the other developers so each knows what the others are doing. Collaboration tools like Microsoft Teams can be employed for this.
3. Always pull the most recent copy the project from the remote repository before starting work on a new branch.
4. If you are already working on a branch, good Git hygiene recommends merging your branch with the main/master branch before beginning work.

There are several caveats that must be adhered to when using Git for Simio for multi-user development:

* All developers must be using the same version and license of Simio.
* The Simio version must be Professional or better.
* All developers must be using the same user extensions.
* The recommended Simio .gitIgnore file should be used. By doing this the temporary files generated by Simio - such as .log and .backup – will be ignored by Git. In other words, these are un-versioned files that you really don’t want to in the repository anyway.

The next sections will describe various scenarios of using Simio with Git. Each has an accompanying video that demonstrates the scenario.

# Prerequisites: Simio Version and Recommended Tools/Apps

## Required Simio Editions

The Simio editions that are compatible with these techniques are Simio Professional and better. Other editions will not work with these Git techniques. You must use Simio version 15.255 or greater.

## Recommended Tools

Here are the tools that Simio recommends for use with Simio. All of them are free and open source.

### Git For Windows

This is the underlying Git mechanism. It is used by higher-level tools (described later). It also comes with a command line tool for Git, called Git Bash.

Available here: <https://git-scm.com/download/win>

### Notepad++

Recommended: This is a free notepad-like application and is quite useful for examining XML files.

Available here: <https://notpad-plus-plus.org>

### Visual Studio Code (VS Code)

This is a source-code editor made by Microsoft for Windows, Linux, and macOS. It provides editor capabilities as well as integrated Git. It is the most popular developer environment according to the 2022 Stack Overflow developer survey.

Available here: [Download Visual Studio Code - Mac, Linux, Windows](https://code.visualstudio.com/download)

The Simio GitHelper Add-In. This is an Simio extension (add-in) that assists in the SCM process.

A screenshot of a computer

Description automatically generated

All the examples here will employ VS Code for this process.

# Example Project - SchedulingBicycleAssembly

The Simio project SchedulingBicycleAssembly, which is an included example from the Simio desktop installation, is our example project. The reason it was chosen is that it is sufficiently large and contains several Simio Models within it.

## Preparing for First Use

The Simio project is prepared for Git use by:

1. Changing it to a multi-file format (a .simproj file with a folder structure)
2. Setting appropriate project properties, which are (1) Save Project as multi-file and (2) Support Source Control
3. Create a Git repository with the recommended .gitignore file (so that unnecessary files are not tracked), and
4. Placing it into a remote (i.e. shared) Git repository (we are going to use GitHub for our examples)

The selected bike assembly project is installed with every edition of Simio, and is located under Program Files > Simio LLC > Simio > Examples > SchedulingBicycleAssembly.spfx.

Please verify that you have Simio Professional version 15.255 *or better* installed. Other editions will not work for historical reasons.

We must change the storage format to Multi-File projects. This changes the traditional single file storage method to a folder structure with multiple files. Again, it is primarily the “source-code” portion of these files (e.g. XML files) that we will individually track with Git.

1. Set to “True” the Project Properties “Save Project as Multiple Files” and “Support Source Control”
2. Open the project (the .SPFX) and then “Save As” to a folder with Multi-File Projects selected (the .simproj option).

A screenshot of a social media post

Description automatically generated

Let us examine what has just been created. Look at the folder where you saved. It should show a folder of the same name as your project with sub-folders:

Open the Models sub-folder and you should see something like:

A screenshot of a social media post

Description automatically generated

Within these folders are several types of files, including XML files. Note that some may be named with unique global identifier names (GUIDs), such as

A screenshot of a cell phone

Description automatically generated

If you open any file with an application that can display XML (such as VS Code or Notepad++) you will find the hierarchical XML file that begins with the “Fragment” element. This simply indicates that this information is a fragment of a larger hierarchy that has been placed into its own file.

This XML is the “source code” for your Simio Project. It describes objects such as the Server which the Simio desktop UI puts into a more readable and graphical form. In a SCM like Git, you will only have to edit this file when there are conflicting changes. That is, if two or more developers work on the same base file and both commit changes.

For this document, we are going to create multiple copies of this project to demonstrate multiple users, who we will call Alice and Batpox …

The local repositories can be named anything you wish, but in this example, we are going to name the folder that contains the project BikeProject and have it live under a folder called “repos” (for repositories… a standard naming convention).

We will open these folders with VS Code so that they are treated as VS Code projects.

We will use these users and repositories to demonstrate the workflow and show how Simio and VS Code interact to utilize the power of Git.

## Setting up the Remote (Shared) Git Repository

When using Git, each developer has their own “local” repository (usually on a disk local to their computer), as well as a “remote” repository where shared changes made by any developer can be pushed to, and then can be pulled from team members so that everyone stays in sync.

The examples in this document will use the free Microsoft repository site called GitHub to demonstrate creating and using a remote shared repository for use by multiple developers***.*** We will create a GitHub repository for our project and each user will initially Git Clone the repository back to our local machine.

*If you decide to use GitHub as your remote repository, you must be careful when setting the visibility (this is true of almost any Git repository). If you set it to public, you may find that you have unintentionally shared your proprietary software with the world!*

So far, we have created the requisite multi-file version of our Simio project.

The steps we will now take to create our remote repository are:

1. Set up an account on GitHub
2. Open the VS Code project and do a one-time Git Push to GitHub. This would be done once and only by one member of the team.
3. Git Clone project back to the locations used by each person of the team. This is then the local repository that each team member will use.

After this is complete, we can discuss the workflow necessary for multiple developers to work on the same project by employing several example scenarios.

## Setting up a GitHub Repository

Note: These examples will show setting up a repository by using the SimioLLC GitHub account. The ability to create Git Repositories here is limited to Simio employees. You will need to set up your own GitHub site and account. (but clearly you would want your own anyway).

For this example, we will create a GitHub repository named BikeProject project that has been saved in mult-file format.

After creating the site, GitHub might ask if you want to do a quick setup. But we will decline, since we’ll use VS Code to do our initial setup.

We will need to use the URL for the remote repository that is shown in the GitHub page below “<> Code” (<https://github.com/batpox/BikeProject.git>). Again, note that you will need to set up your own remote repository; this one is used for demonstration purposes for this document.

## Doing the Initial Push to GitHub

In our fictitious project, our team leader has created the project in the correct format and placed it under a folder called BikeProject.

Graphical user interface, text, application, email

Description automatically generated

We now want to do our one-time Push of this project to our GitHub repository.

Using VS Code, navigate to the folder holding the Bike project files and do:

File > Open Folder

This establishes the folder as a VS Code Project, and you will see something like this:

A screenshot of a computer

Description automatically generated with medium confidence

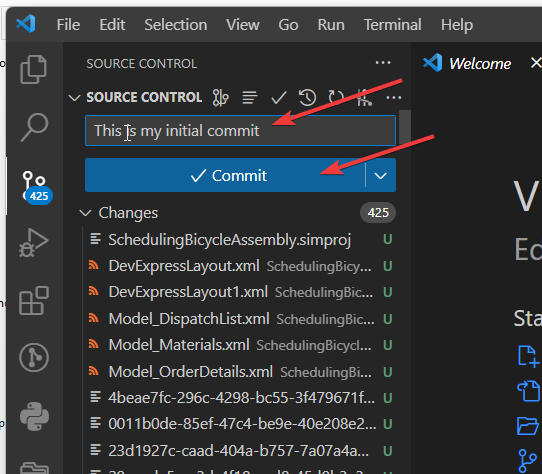
Select the Git symbol on the left panel A picture containing icon

Description automatically generated and you are presented with actions to create a local repository and to push the repository to a remote repository.

A screenshot of a computer

Description automatically generated with medium confidence

Push Initialize Repository to create your local repository. Once you do this, you’ll see all the files and a button to Commit. Note that you must ***always*** add a Commit message prior to a Commit.



At this point (if you have set Windows Explorer with the ability to see hidden files) you will see:

Graphical user interface, text, application

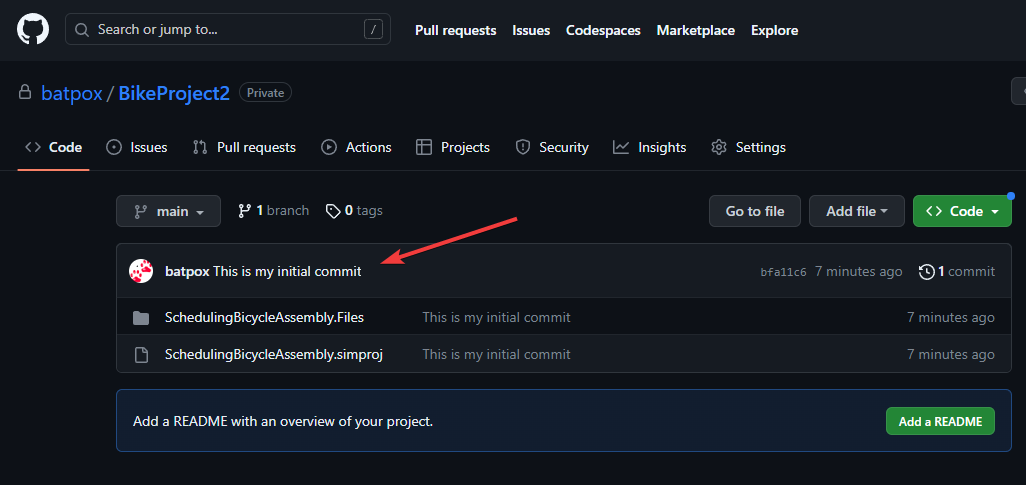
Description automatically generated

The “.git” hidden folder contains files used internally by Git. This is our local repository.

*Note that you will need a GitHub account with the ability to create private repositories prior to this next step.*

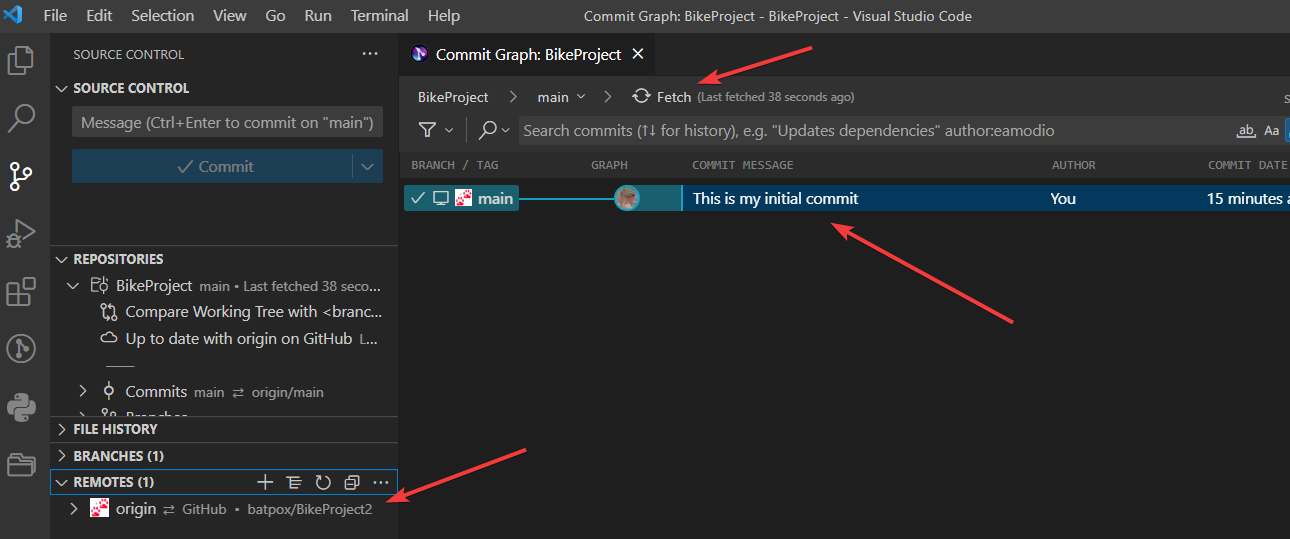
After the commit the local repository, the “Publish Branch” button will allow to you select a name and publish to GitHub. If you try to publish to an already existing repository, you will get an error.

We have named this BikeProject2 and Published it and here is the result on GitHub (Note the Commit message):



Before we add another team member, let’s make a change in Simio and commit the change to see what happens.

Here is what we see in VS Code. Note the initial commit



Also note the ability to fetch (from the remote repository) to our local repository.

The REMOTES in the left panel shows the remote repository that we are fetching from.

Start Simio desktop and open our project from repos > BikeProject > SchedulingBicycleAssembly.simproj

We’ll make example two changes:

1. Move the Stock server in the Facility view to be close to the exit.
2. Go to Processes and change the Search Step to a light shade of green.
3. Press File > Save and then Exit Simio.

If you look at the folder in File Explorer, you will see red circles with exclamation marks indicating changes:

Graphical user interface, text, application

Description automatically generated

And in VS Code you will see the changes. Selecting the OnRunInitialize.XML from “Changes” shows the changes to the file:

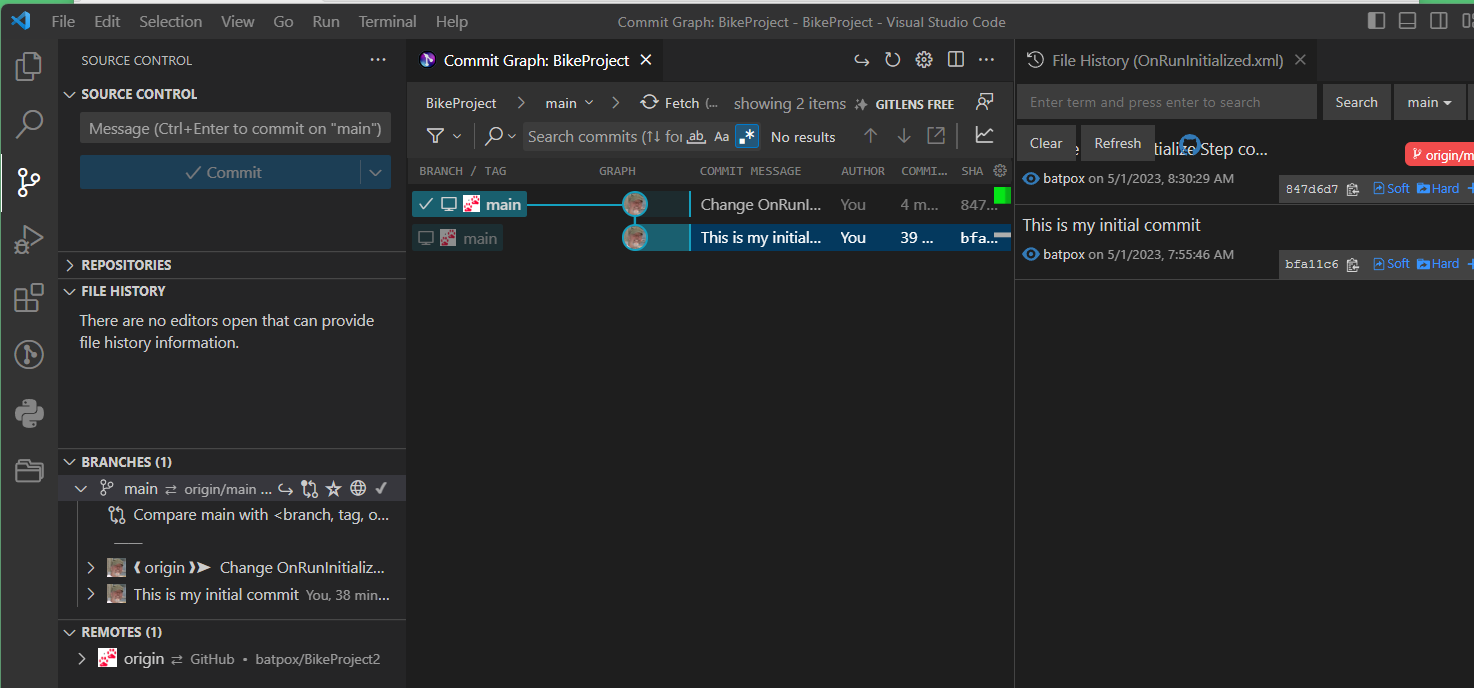
Text

Description automatically generated

Now we add the obligatory commit message and the press “Commit”.

This commits the changes to our local repository and the button is now “Sync Changes”. Once pushed, the action sends the changes to our remote repository on GitHub.

We can now use VS Code to see a history of our changes:

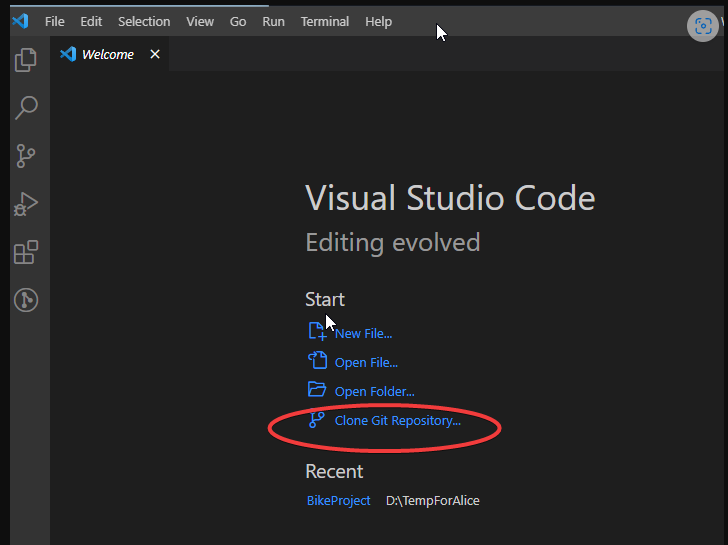


## Adding Alice

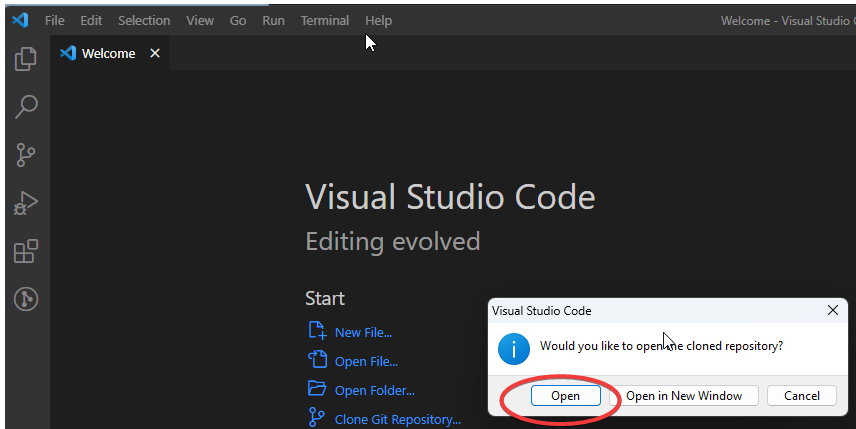
Now let’s add another user. This user (Alice) need only set up a folder where the project will be placed and then:

1. Run VS Code
2. Clone the remote repository from GitHub.

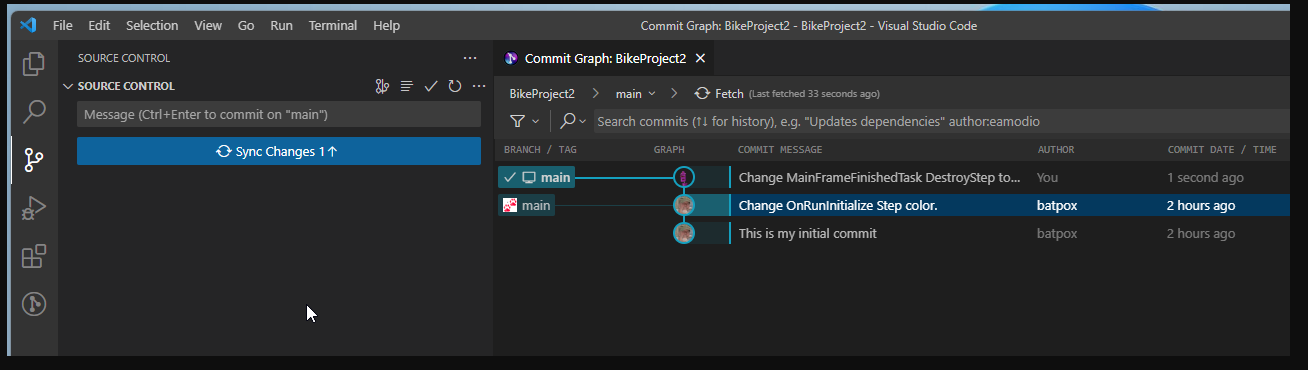
Upon running VS Code Alice sees the following and selects “Clone Git Repository”.



She is presented with options to clone the repository to a folder of her choosing (in this case Documents > repos) and then a choice to Open in VS Code:



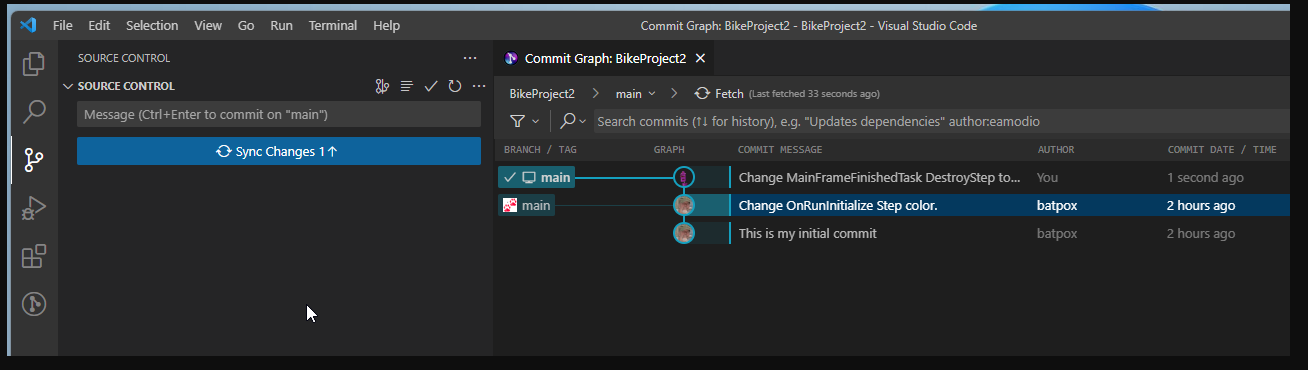
Upon opening, she can now see the repository as well as the history of changes:



Alice can now go to her local repository and make changes. Let’s say she decides to move the Paint Servers on the Facility view and move the camera in tight to show a closeup of those Servers.

She also makes a color change to the Destroy Step on the Mainframe Add-on Process, and then Commits the changes to her local repository.

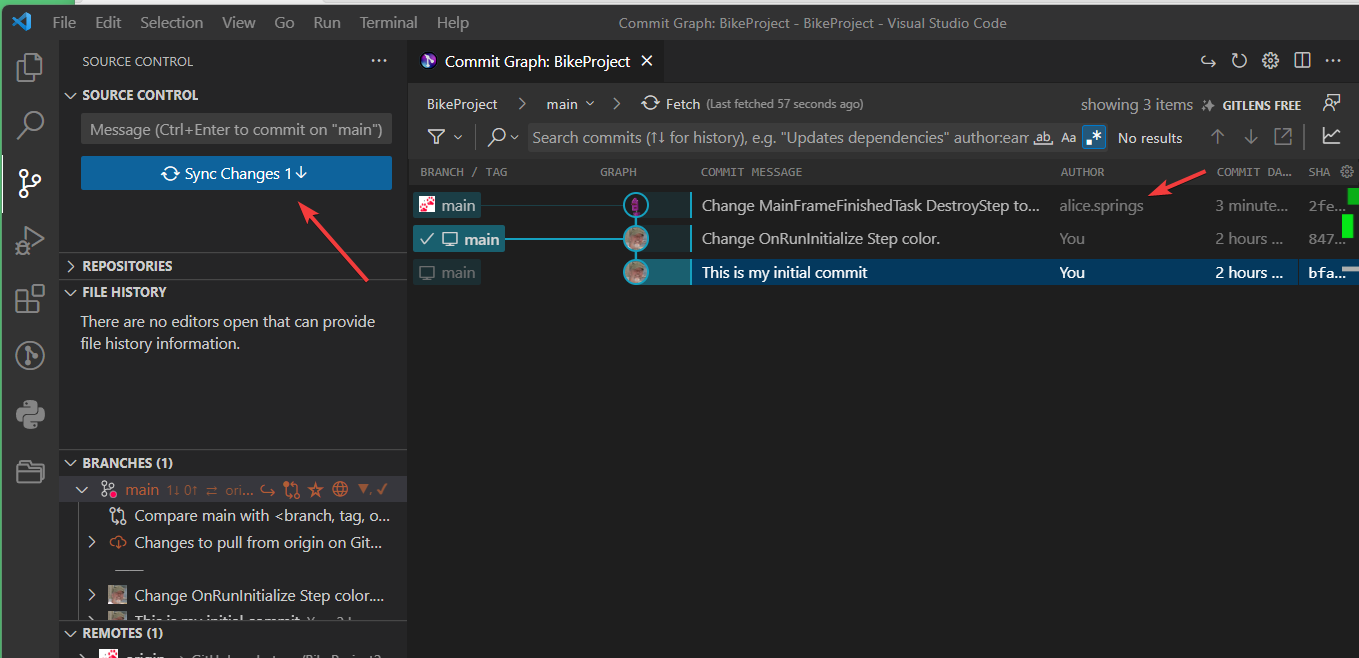
Finally, she Pushes (Sync Changes) the changes to the remote repository.



## Back to Batpox

Switching back to Batpox we now see Alice’s Changes that must be Sync’d (pulled by Batpox).

It is very important to stop Simio before continuing because it will change the Simio project files, but Simio currently does not have the ability to recognize this and will not re-load them. If Simio were left running and a Simio “Save” was issued it would overwrite the Git changes.



Pressing the Sync button and launching Simio shows the changes to MainFrameAdd-On Step Destroy:

Diagram

Description automatically generated

Bringing up the Facility view shows the Shape Servers were moved by Alice; but not only that, so is our camera location and direction! This is because we accepted all changes including ViewInfo information. Changes like this can be selectively chosen (e.g. by using .gitIgnore)and this option will addressed in a future tutorial.

# Summary and the Road Ahead

As mentioned before, Simio is actively working internally on how to improve the workflow for multiple developers and recognizes the importance of this feature.

This is the first in a series of documents and videos that explain how to best use these tools.

Also planned is tighter integration of Simio and VS Code to provide a more seamless interface.

# Appendix – Simio Project File Details

This appendix gives more insight into the Simio Project file, and specifically the Bicycle Assembly .simproj file.

There is a free tool called WinDirStat that provides a visual breakdown of any folder. Here it is for our BikeProject Folder:

A screenshot of a computer

Description automatically generated

The types of files (as indicated by their extension) are shown in the upper right panel and are color-coded and include the size (bytes and percentage) as well as the count. The 214 XML files contain the Simio declarative source “language” and so are the files that need to be tracked by Git.

The bottom panel shows the files by volume.

The upper left panel shows the hierarchy of folders and files.

Clearly the heavy hitter is the Data > Geometry folder with over 88% of the total size. Theses are visualization files and are binary and Simio-proprietary format. The extensions generally begin with S3d, which means “Simio 3D” data. These are textures, animations, etc. and take of over 77% of the entire volume. There are at present no editors for these, so they might be added or deleted, but conflicts are unlikely. They are included in the Git repository because they are needed for the project to run.

So let’s do a run and look at how the folder structure is altered.

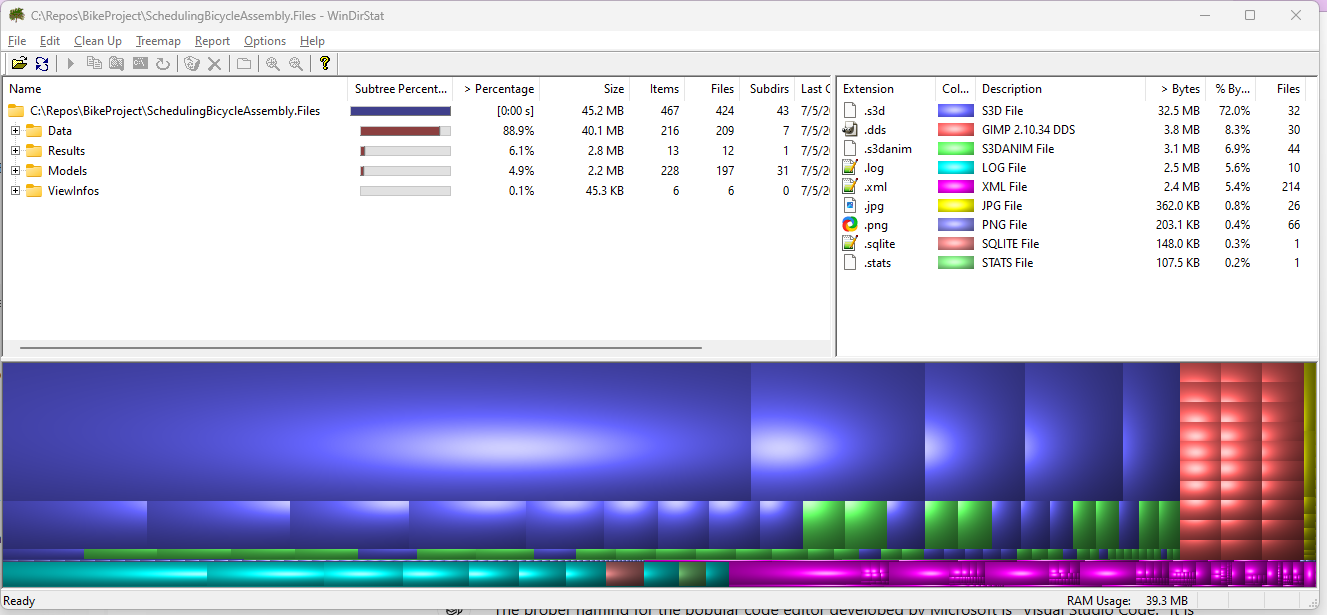
Now contrast that diagram with this one, which was taken after a simulation run:

A screenshot of a computer

Description automatically generated

You see we have many extension-less files under .git > objects. These are not part of Simio at all, but reflect the fact that our git repository is growing. There are also a number of .log files that are produced as part of the simulation run.

If we just looked at the folder SchedulingBicycleAssembly.Files (below) we would see that the biggest increase was due to the run was the .log files.



Note that the number of XML files (which contain the Simio declarative language/logic) has not changed from the original 214.

Also note the .DDS files. These are Direct Draw Surface files, which is a Microsoft format for storing and compressing images, mipmaps, and textures that are used in the Facility display.

If you install software such as the free Gimp software, you can examine the DDS files:

# Appendix – First Use In GitHub with VSCode

The assumption is that you’ve converted your Simio project to the Simproj multi-file format and set the Project Settings to “Use Source Control” and “Use Multiple Files”.

What you have is a folder (in this example let’s call it BikeExample and it should contain a file “SchedulingBicycleAssembly.simproj” and a sub-folder called “SchedulingBicycleAssemble”.

When in VSCode, you should open the folder and then select the Source-Control Icon to show this:

A screenshot of a computer

Description automatically generated

Pressing “Publish to GitHub” allows you to choose which files you want to be in the repository. Here we have accidentally placed some extra files, plus the Simio backup is there, and we don’t want it, so we only select the files we want, which create a .gitignore, which we’ll explain directly.

A screenshot of a computer

Description automatically generated

Once you press the “OK” a repository will be setup for you and you’ll see you project in GitHub as this:

A screenshot of a computer

Description automatically generated

This is your remote repository.

Look at your folder in File Explorer and you’ll notice a new “.git” folder. This holds your local repository.

## The GitIgnore File

**It *is very important to set up your .gitignore file correctly as soon as you can.***

So let’s do it now. This is a text file that begins with a dot “.”. You can open it with notepad and it will show what files or files are to be ignored by git. In our example we already told it to not include the accidental .spfx file and the normal Simio backup:

A screenshot of a computer

Description automatically generated

But we should ignore some additional files as well. Which files you choosed to ignore may vary. For example, perhaps you want to not include any simulation results because you only want to track logic changes.

We are going to make our file look like this. A line starting with the pound symbol (#) indicates a commenct:

# no need for the simio backup files

\*.backup

# sqllite will regenerate itself

\*.sqllite

# .logs will regenerate

\*.log

# Appendix - The Concept of theProject Folder

A concept that can be confusing at first is that our folder that holds the Simio project (.simproj and .files) is also a project for VSCode and a repository for Git.

Git knows this by the presence of the .git subfolder, and VSCode by the presence of the .vscode subfolder.

***Note that you can examine these ‘dot’ subfolders, but do not change them!***