Simio
Forward Thinking

Simio API Note: Using Git with Simio

July 2020 (Dhouck)

Contents

Simio

Forward Thinking

Overview

This API Note describes how to use Git with Simio. Although this is not really about API, is shows how to use world's most popular version control software (Git) with Simio to enable multiple engineers to work on the same Simio project, and that would include API work.

This note 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 an 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 event create separate branches for different clients. It also can provide for continuous backups to a remote location. These points are not covered in detail in this document, but a huge amount of Git resources are 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 installation "SchedulingBicylceAssembly" is used. This is a relatively complex model where the benefit of multiple developers might be seen.

Simio
Forward Thinking

Some Background Information on Git

From Wikipedia:

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, 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 Simio Models.

Git Overview

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

For background on Git I would recommend 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. Each developer has a personal 'local' copy of a repository, and then the changes from those repositories can be "pushed" back to a shared 'remote' repository.

A repository holds not only the files, but also meta-data associated with those changes. Meta-data such as dates and user-entered text with the change(s). So, for example, you can see the entire history of how a model changes over time, and then even revert the model to an earlier time. You can even make 'branches'. For example, let us say you want to make experimental changes to your model. If it works, you might then want to merge these changes back to the main branch, but if it does not, then you can simply abandon the branch.

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

There is a serious caveat though. Using Git (or any source repository) 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.

Simio
Forward Thinking

Some Git Terminology

This document assumes you have some Git knowledge. It 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 a *local* repository along with a 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.

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 work 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 the free Git Tortoise) to resolve the conflict, you are shown the files and the changes.

Simio
Forward Thinking

Of course, the easiest way to deal with conflicts is to not have any to begin with, and the best way to do *that* is to divide the work such that only a single person is working on a file at a time.

There is not automatic way to do this with Git since each developer has their own repository. One way, which requires some discipline is to employ collaborative software such as Microsoft Teams to announce to the group what software you are planning on modifying or working on for the next few days.

There are several caveats that must be adhered to when using Git for Simio:

- 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 Simio .gitignore file should be used. This ignores files such as .log and .backup

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 Required Tools/Apps

Requires Simio Version

The versions of Simio that are compatible with these techniques are Simio Professional and better. Other versions will not work for a variety of reasons.

There are a few free and useful tools that will be employed for this demonstration.

Tools/Apps

Git For Windows

This is the underlying Git mechanism. It is used by Tortoise Git (described later).

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

Notepad++

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

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

Tortoise Git

This is a utility that is tied into the Windows File explorer and provides a GUI implementation of many of the features of Git. Its use is assumed in all the examples of this document.

Available Here: https://TortoiseGit.org

GitHub for Windows

This utility is tied into Windows File Explorer and provides a GitHub UI for downloading into Windows.

Example Project - SchedulingBicycleAssembly

We will use SchedulingBicycleAssembly, which is included with the Simio desktop installation, as our example project. The reason is that it is sufficiently large and contains several Simio Models within it.

Scenario 1: Preparing for First Use

In this first scenario we prepare our Simio project for Git use by:

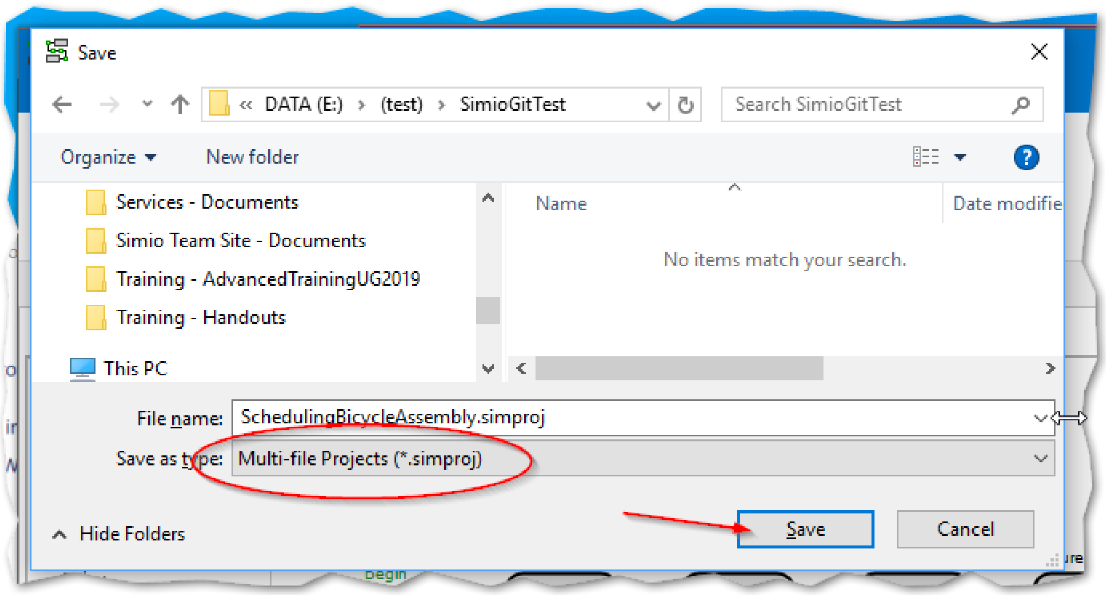
1. Changing it to a multi-file format
2. Create a Git repository, and finally
3. Placing it into a remote (i.e. shared) Git repository (we are going to use GitHub)

The project is located under Users > Public > Public Documents > Simio > SchedulingBicycleAssembly.spfx.

Please verify that you have Simio Professional or better installed. Other versions will not work for various reasons.

First, we must change to storage format to Multi-File projects. This changes the normal single file storage method (an .SPFX file) to a folder structure with multiple files. It is the "source-code" portion of these files (primarily XML files) that we will individually track with Git.

So, open the project and then "Save As" to a folder with Mult-File Projects selected.


Save

DATA (E:) › (test) › SimioGitTest

Search SimioGitTest

Organize ▾ New folder

Services - Documents
Simio Team Site - Documents
Training - AdvancedTrainingUG2019
Training - Handouts
This PC

Name | Date modified

No items match your search.

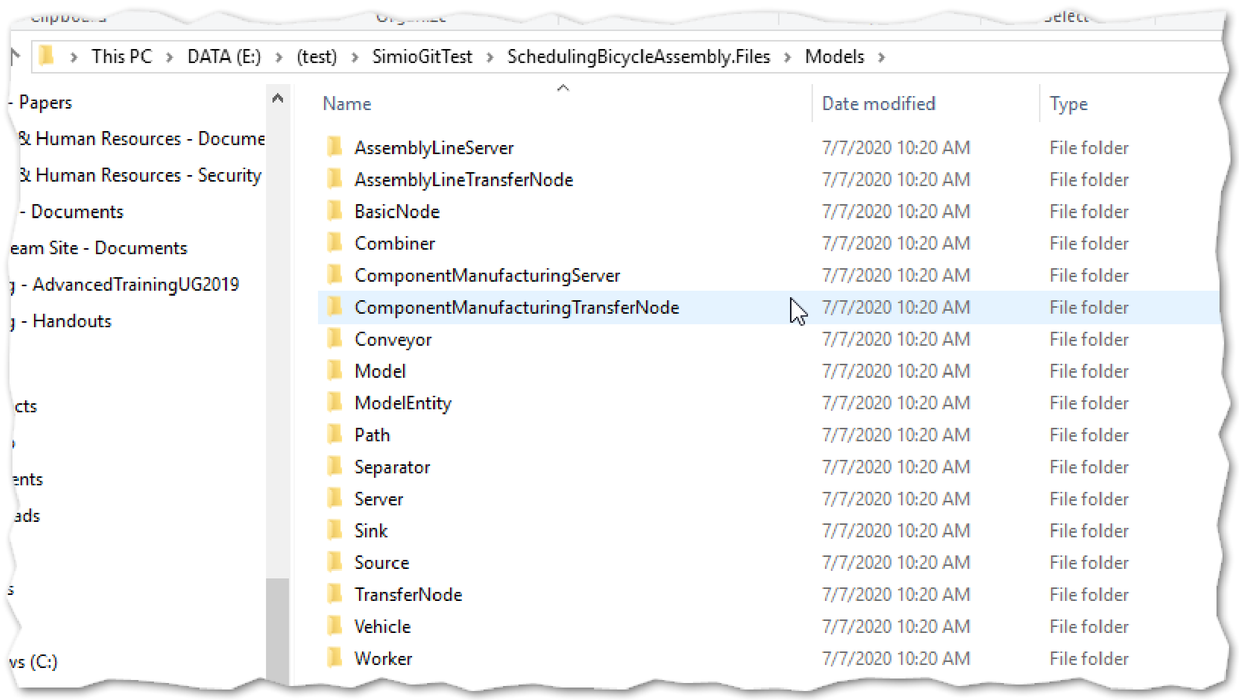
File name: SchedulingBicycleAssembly.simproj
Save as type: Multi-file Projects (*.simproj)

Hide Folders

Save Cancel

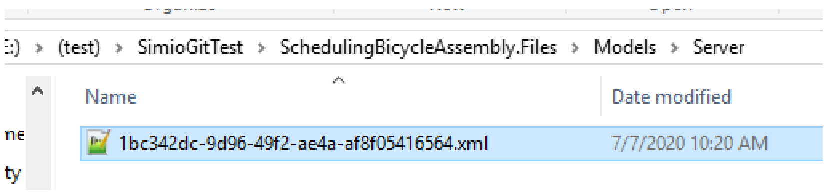
Let us examine what we have just created. Look at the folder where you saved. It should show a folder of the same name as your project with sub-folders:

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


This PC › DATA (E:) › (test) › SimioGitTest › SchedulingBicycleAssembly.Files › Models ›

Name	Date modified	Type
AssemblyLineServer	7/7/2020 10:20 AM	File folder
AssemblyLineTransferNode	7/7/2020 10:20 AM	File folder
BasicNode	7/7/2020 10:20 AM	File folder
Combiner	7/7/2020 10:20 AM	File folder
ComponentManufacturingServer	7/7/2020 10:20 AM	File folder
ComponentManufacturingTransferNode	7/7/2020 10:20 AM	File folder
Conveyor	7/7/2020 10:20 AM	File folder
Model	7/7/2020 10:20 AM	File folder
ModelEntity	7/7/2020 10:20 AM	File folder
Path	7/7/2020 10:20 AM	File folder
Separator	7/7/2020 10:20 AM	File folder
Server	7/7/2020 10:20 AM	File folder
Sink	7/7/2020 10:20 AM	File folder
Source	7/7/2020 10:20 AM	File folder
TransferNode	7/7/2020 10:20 AM	File folder
Vehicle	7/7/2020 10:20 AM	File folder
Worker	7/7/2020 10:20 AM	File folder

Within these folders are XML files with unique global identifier names (GUIDs), such as


:) > (test) > SimioGitTest > SchedulingBicycleAssembly.Files > Models > Server
Name
Date modified

1bc342dc-9d96-49f2-ae4a-af8f05416564.xml 7/7/2020 10:20 AM

If you open this file with an application that can display XML (such as the free NotepadPlusPlus) you will find the hierarchical XML file that begins with the “Fragment” element.

This is the “source code” for Simio. It describes the Server which Simio desktop puts into a more readable and graphical form. You will only have to enter this file if there are conflicts. That is, if two or more developers work on the same base file and make changes.

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

We are going to put these projects in subfolders under c:\Test, so they will be:

C:\Test\SimioGitExampleAlice

C:\Test\SimioGitExampleBob

… and so forth.

Finally, we are going to generally use a graphical UI to manage the project. We are here using TortoiseGit, but all these operations can be done using the Git command line, which you may prefer. Again, refer to resources such as Ry’s Git Tutorial for examples.

Simio
Forward Thinking

Setting up the Remote (Shared) Git Repository

When using Git, each developer has their own repository (usually on a disk local to their computer), but for collaboration there is also a remote repository where global changes made by one developer can be *pushed* to and then can be *pulled* from by other developers so that everyone is in sync.

The examples in this document will use the Microsoft 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 then Git Clone the repository back to our local machine.

You must be careful when setting the visibility GitHub (or any other 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 are to:

1. Set up an account on GitHub
2. Do a one-time Git Push to GitHub
3. Git Clone project back to some other locations on our local disk. This will become the location where we do all our development work.
4. Git Pull the project back to a separate location. This simulates having a second developer working on the Simio project.

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

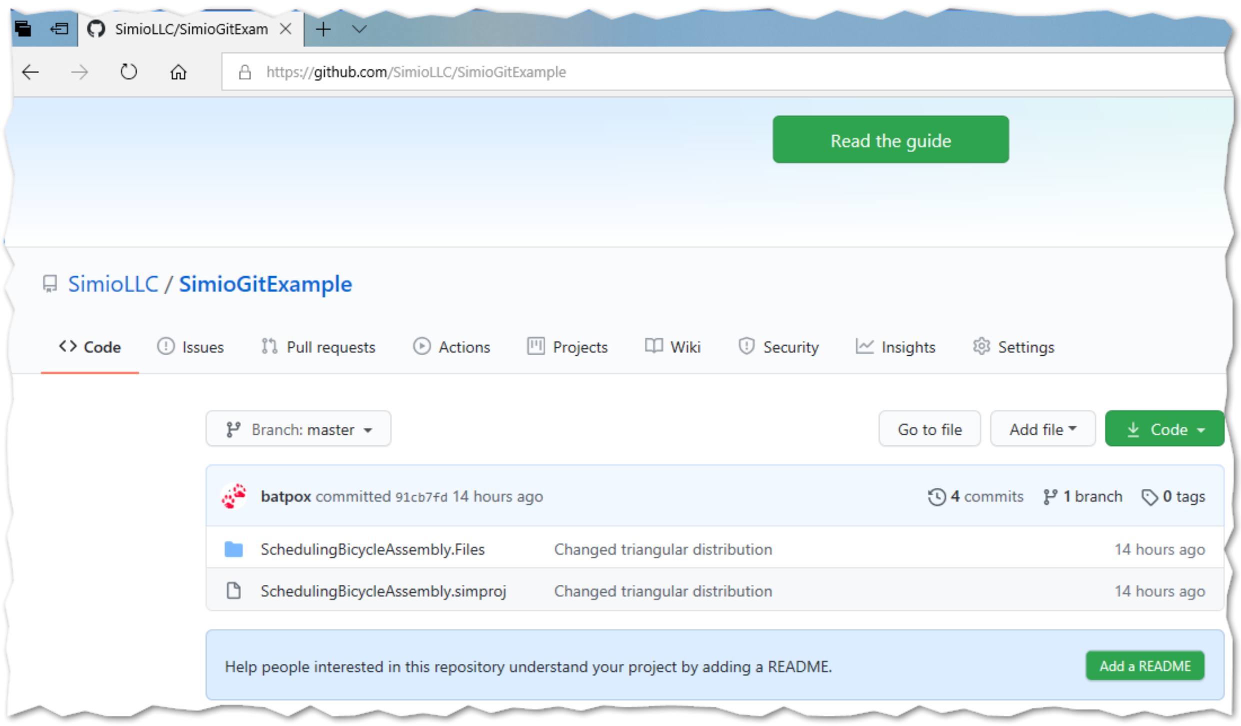
Video: Creating the Remote Git Repository

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 SimioGitExample that will contain the Simio SchedulingBicycleAssembly project that has been saved in mult-file format.

Simio
Forward Thinking

When we are done, it will look like this:


https://github.com/SimioLLC/SimioGitExample

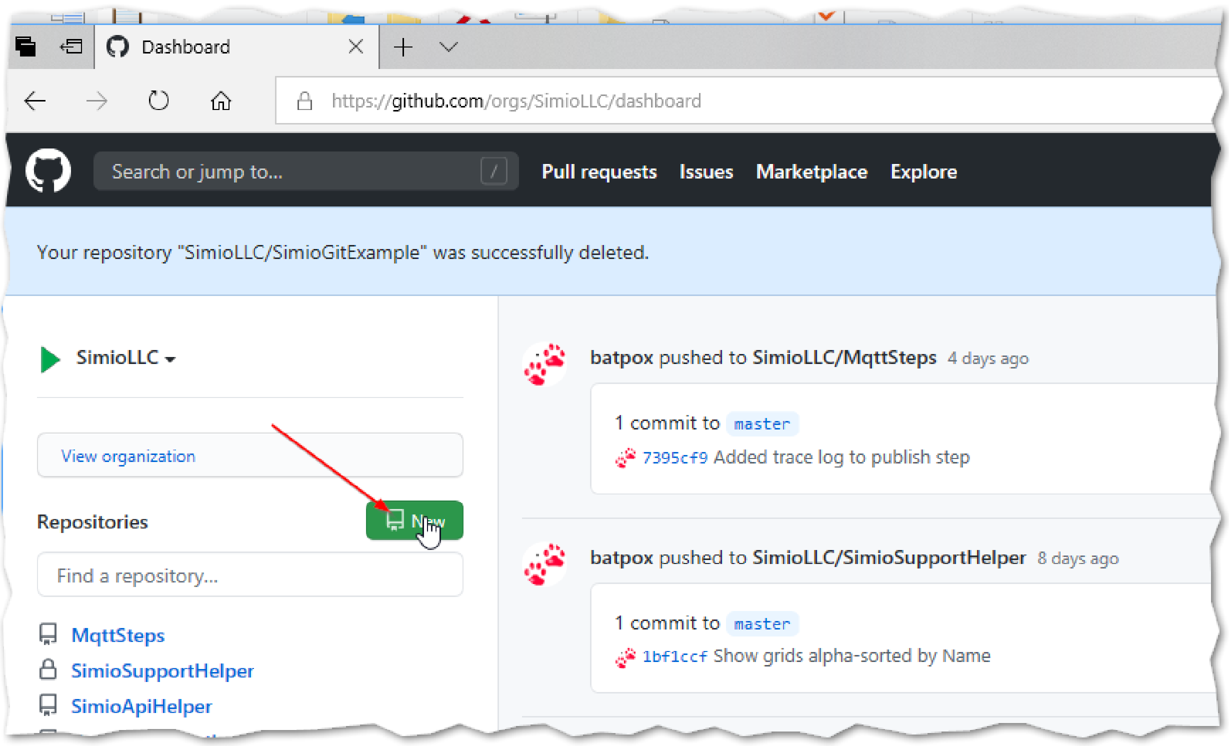
SimioLLC / SimioGitExample

batpox committed 91cb7fd 14 hours ago — 4 commits, 1 branch, 0 tags

SchedulingBicycleAssembly.Files	Changed triangular distribution	14 hours ago
SchedulingBicycleAssembly.simproj	Changed triangular distribution	14 hours ago

Help people interested in this repository understand your project by adding a README.

We start by going to our account and creating a New Repository:

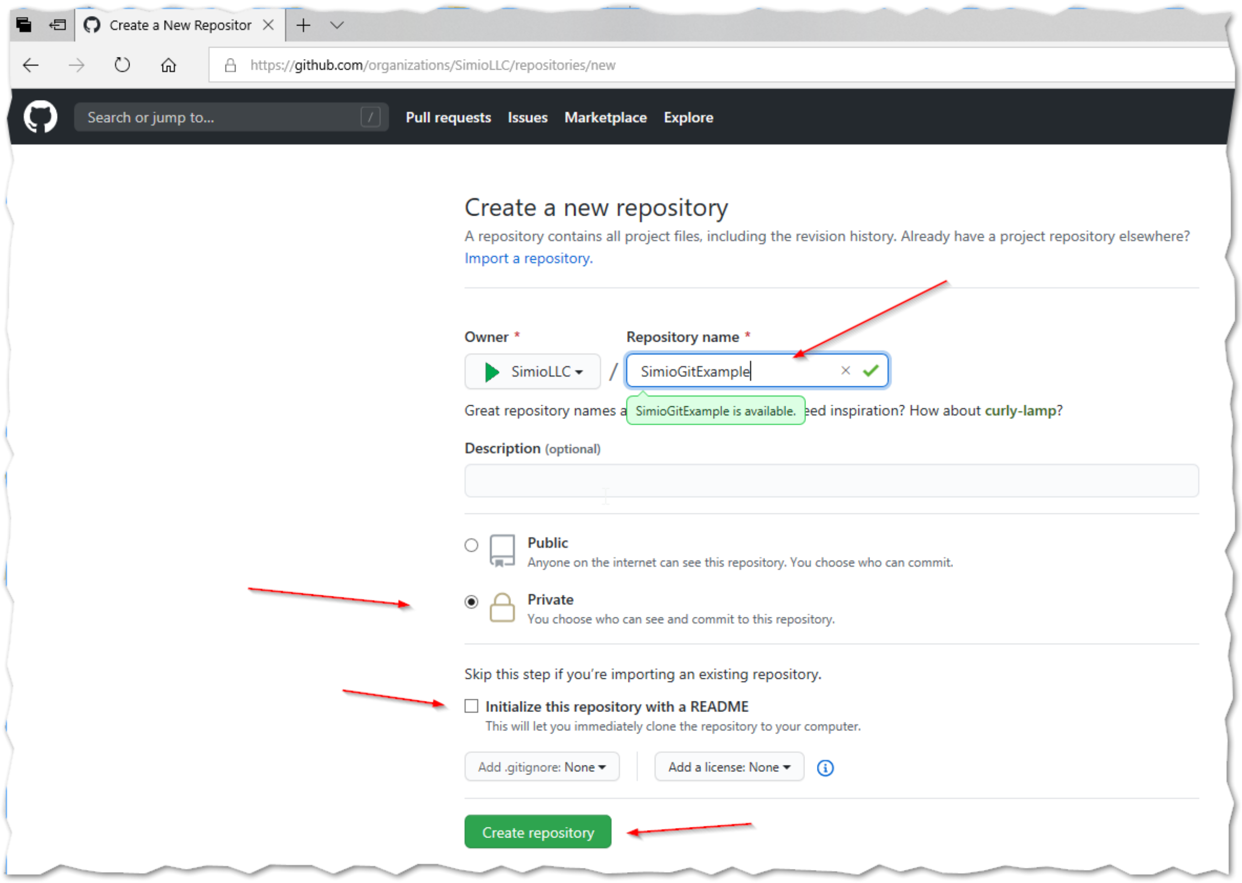


Your repository "SimioLLC/SimioGitExample" was successfully deleted.

batpox pushed to SimioLLC/MqttSteps 4 days ago
1 commit to `master`
`7395cf9` Added trace log to publish step

batpox pushed to SimioLLC/SimioSupportHelper 8 days ago
1 commit to `master`
`1bf1ccf` Show grids alpha-sorted by Name

At the "new" page, enter only the essential information. It is simpler if you do **not** Initialize with a README at this point.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * SimioLLC / Repository name * SimioGitExample

Great repository names a SimioGitExample is available. eed inspiration? How about curly-lamp?

Description (optional)

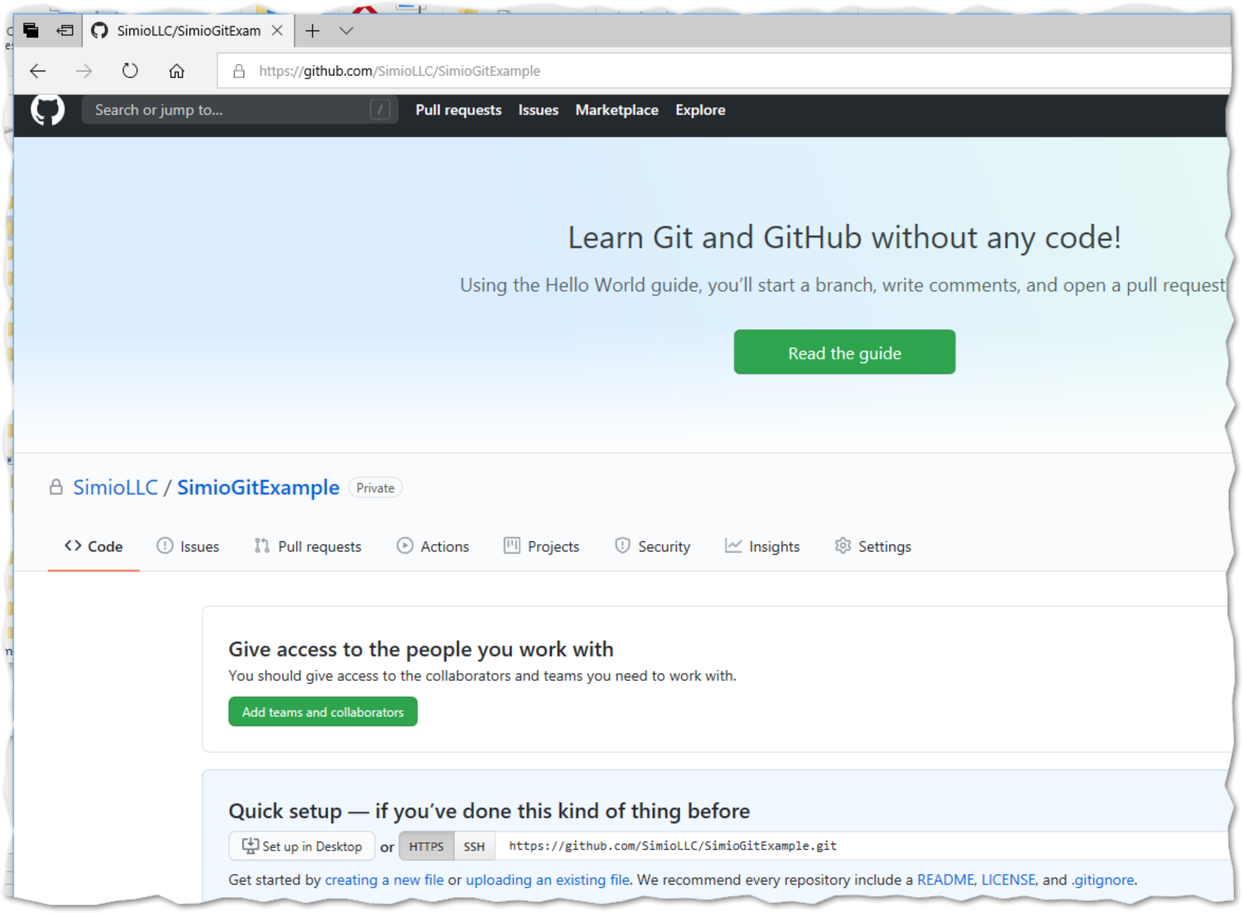
Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer.

It should quickly come back with something like this, asking if you want a quick setup. But we are going to Git Push using Tortoise Git to do our initial setup.



Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request

SimioLLC / SimioGitExample Private

Give access to the people you work with

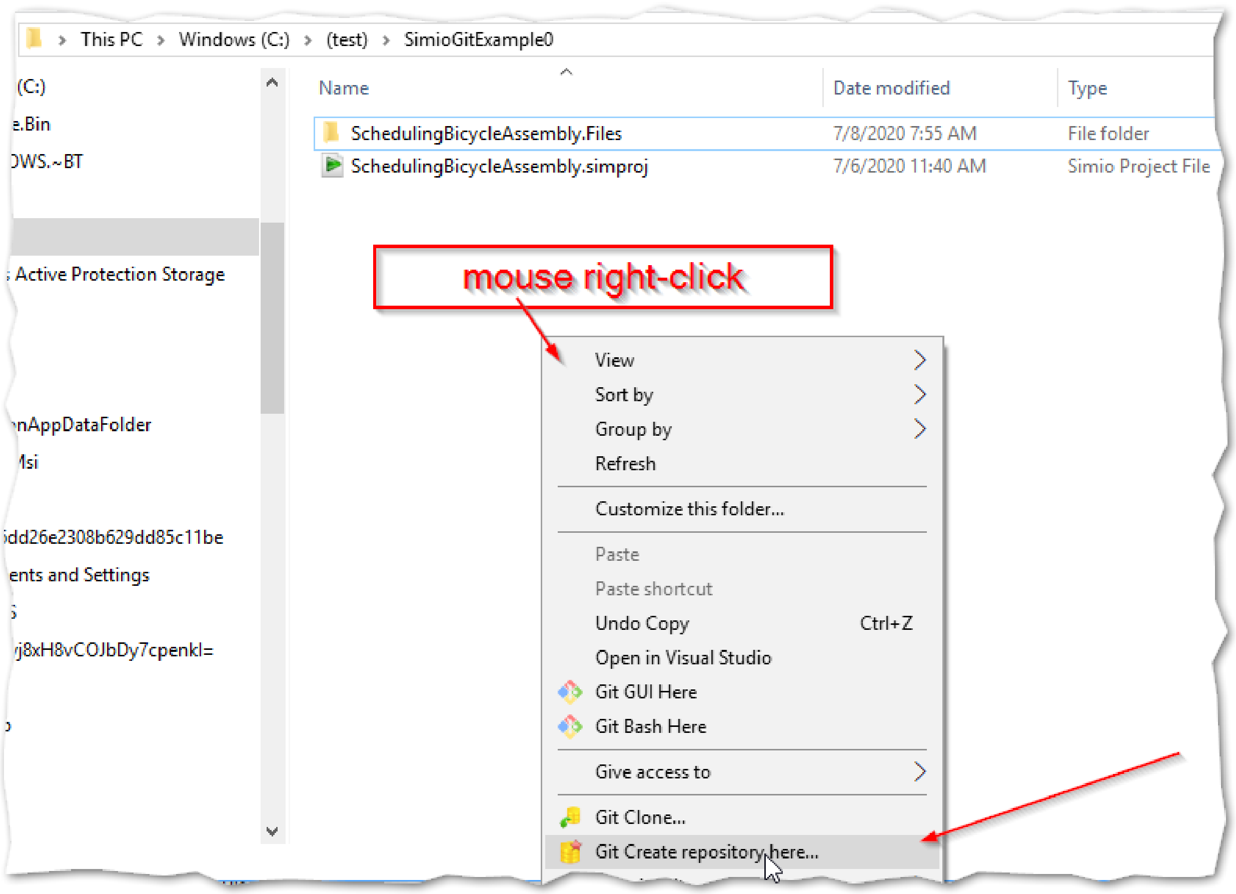
You should give access to the collaborators and teams you need to work with.

Quick setup — if you've done this kind of thing before

`https://github.com/SimioLLC/SimioGitExample.git`

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

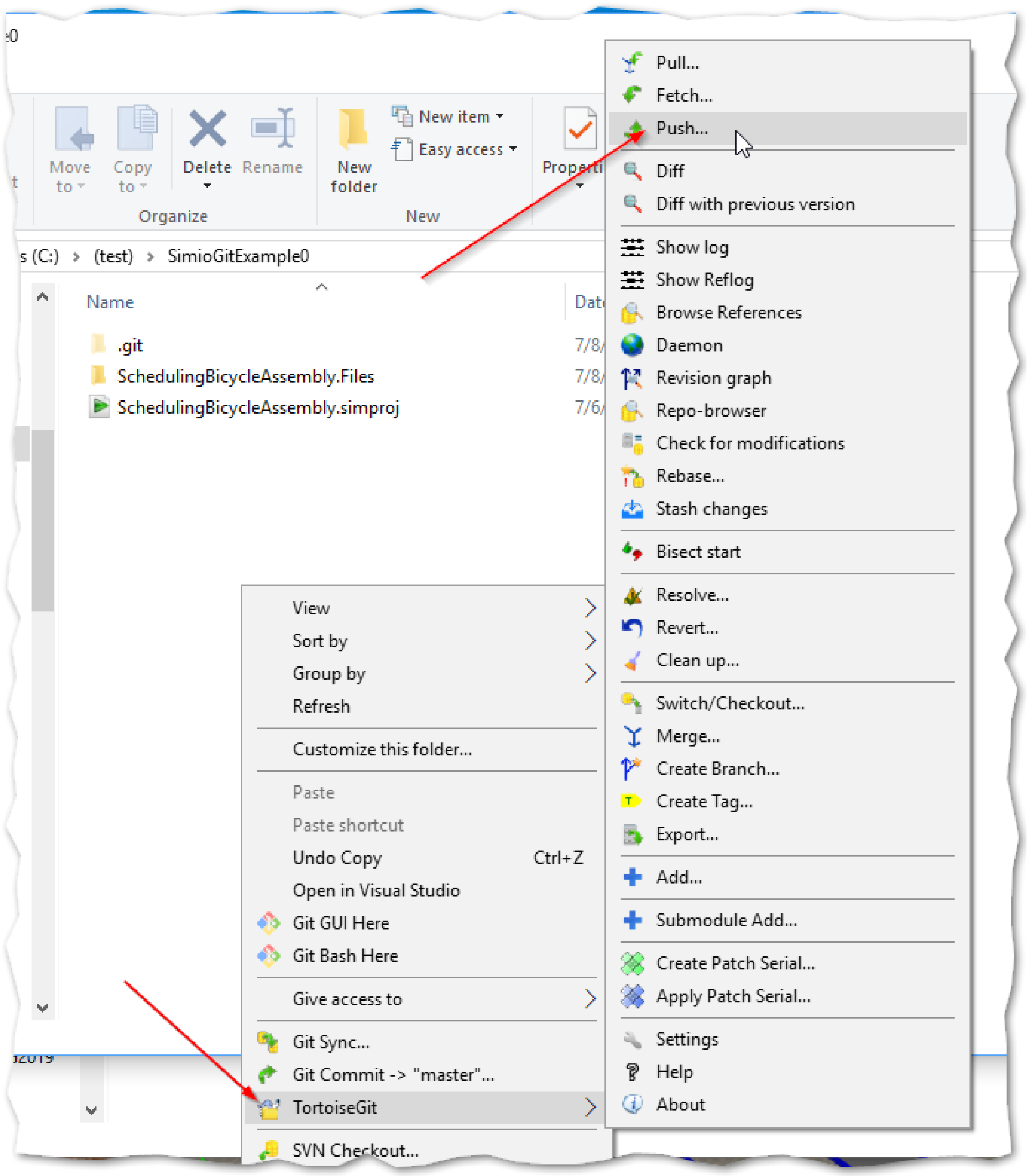
Using Windows File Explorer, hop back to the folder that contains our Simio Project and right-click to show the Tortoise Git menus.

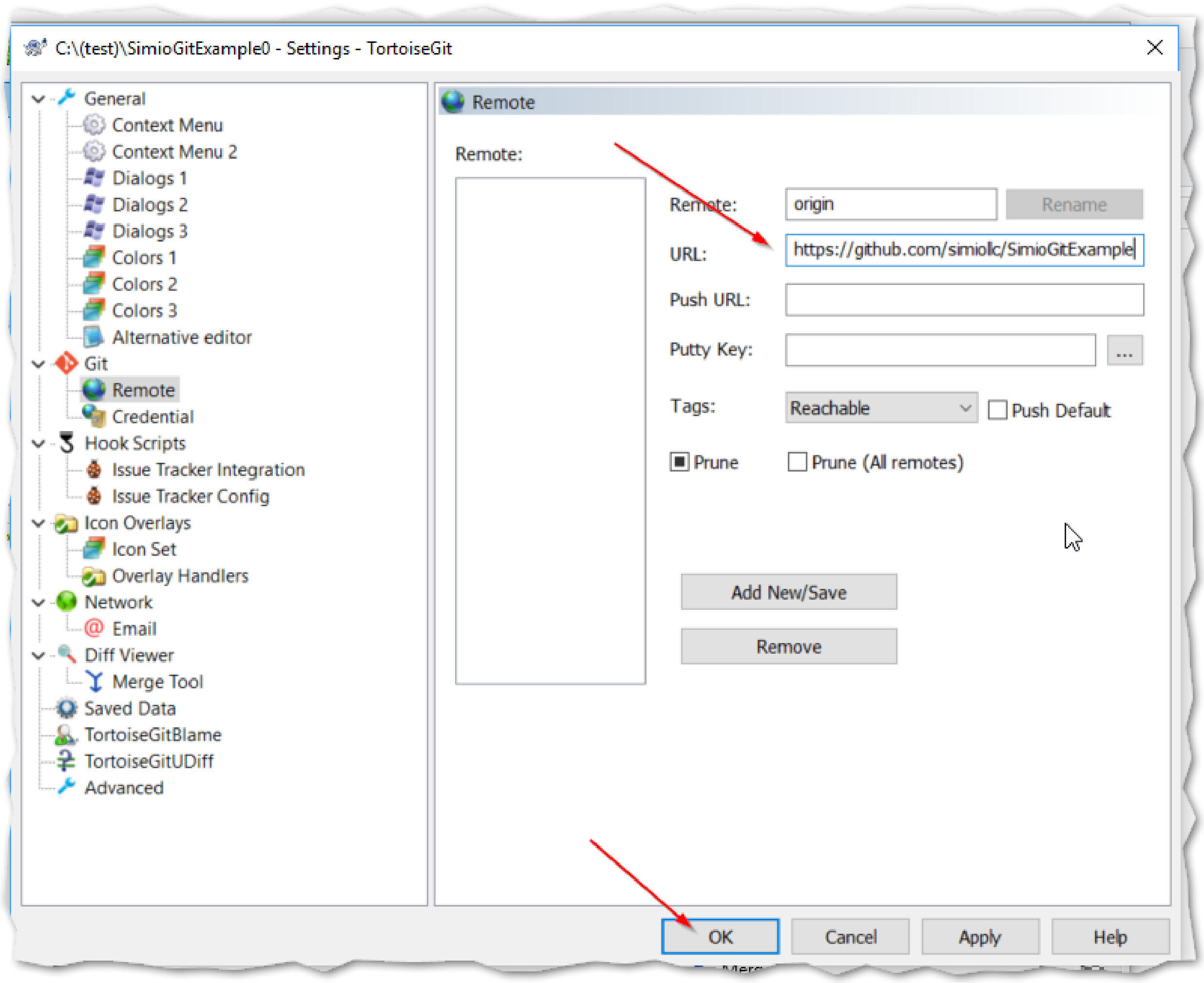

This PC > Windows (C:) > (test) > SimioGitExample0
Name
Date modified
Type
SchedulingBicycleAssembly.Files 7/8/2020 7:55 AM File folder
SchedulingBicycleAssembly.simproj 7/6/2020 11:40 AM Simio Project File
mouse right-click
View
Sort by
Group by
Refresh
Customize this folder...
Paste
Paste shortcut
Undo Copy Ctrl+Z
Open in Visual Studio
Git GUI Here
Git Bash Here
Give access to
Git Clone...
Git Create repository here...

This creates a Git folder which is your local repository.

Now when you again right-click, there is a different menu and you can select Tortoise Git and then Push…

… which brings up a dialog for your initial push.


Move to
Copy to
Delete
Rename
New item
Easy access
New folder
Properties
Organize
New
s (C:) › (test) › SimioGitExample0
Name
Date
.git
SchedulingBicycleAssembly.Files
SchedulingBicycleAssembly.simproj
View
Sort by
Group by
Refresh
Customize this folder...
Paste
Paste shortcut
Undo Copy Ctrl+Z
Open in Visual Studio
Git GUI Here
Git Bash Here
Give access to
Git Sync...
Git Commit -> "master"...
TortoiseGit
SVN Checkout...
Pull...
Fetch...
Push...
Diff
Diff with previous version
Show log
Show Reflog
Browse References
Daemon
Revision graph
Repo-browser
Check for modifications
Rebase...
Stash changes
Bisect start
Resolve...
Revert...
Clean up...
Switch/Checkout...
Merge...
Create Branch...
Create Tag...
Export...
Add...
Submodule Add...
Create Patch Serial...
Apply Patch Serial...
Settings
Help
About


C:\(test)\SimioGitExample0 - Settings - TortoiseGit

Remote

Remote:

Remote: origin

URL: https://github.com/simiollc/SimioGitExample

Push URL:

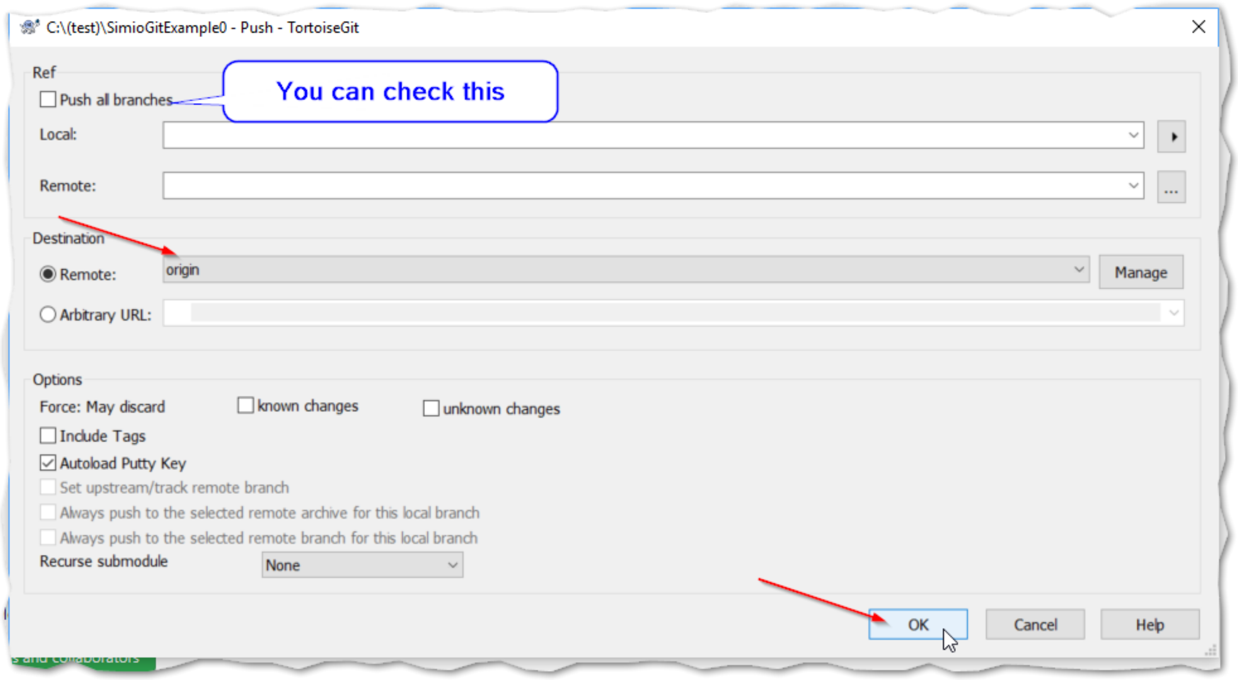
Putty Key:

Tags: Reachable

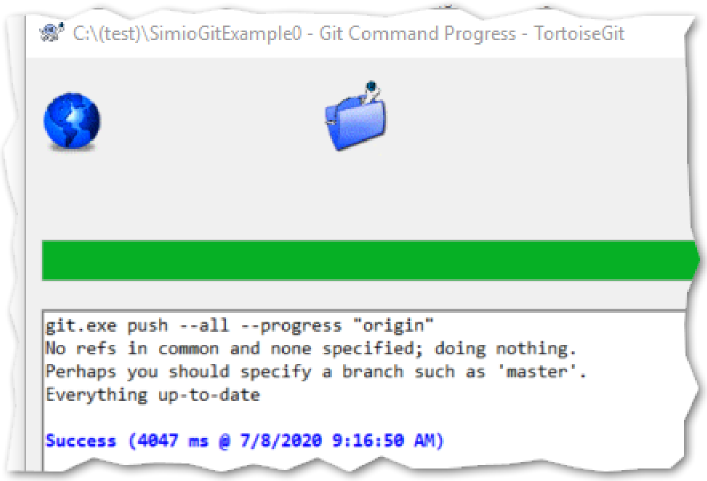
[ ] Push Default

[✓] Prune

[ ] Prune (All remotes)



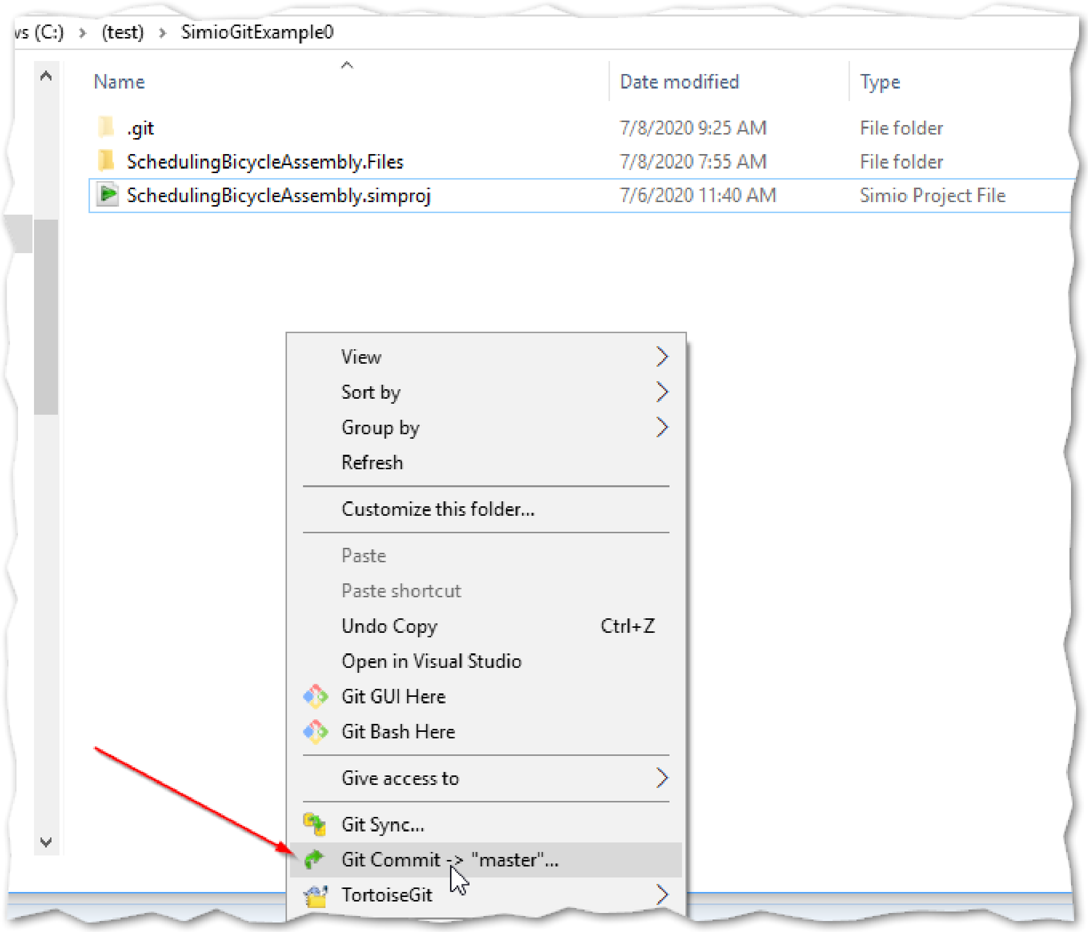
So finally, you should see this dialog from TortoiseGit


C:\(test)\SimioGitExample0 - Git Command Progress - TortoiseGit

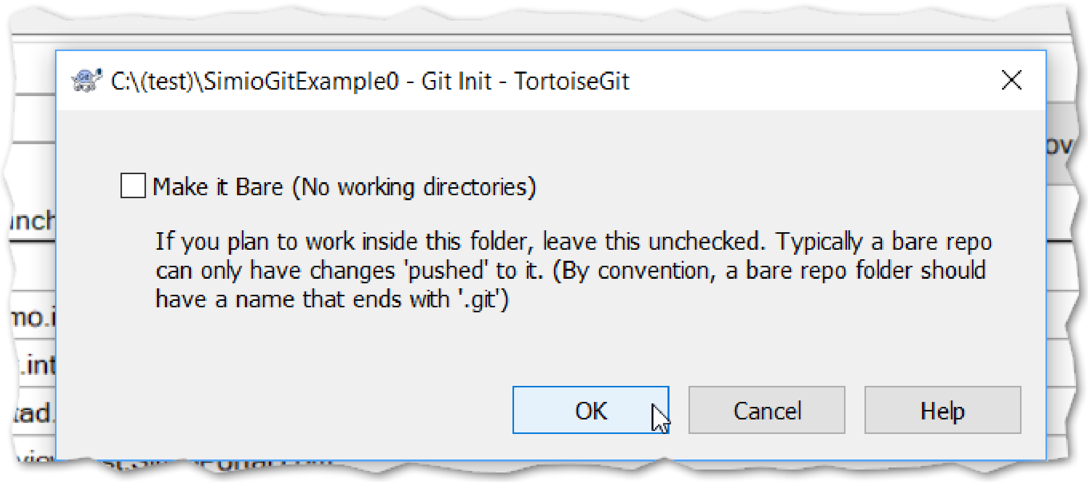
```
git.exe push --all --progress "origin"
No refs in common and none specified; doing nothing.
Perhaps you should specify a branch such as 'master'.
Everything up-to-date

Success (4047 ms @ 7/8/2020 9:16:50 AM)
```

Finally, again right-click on the folder and select

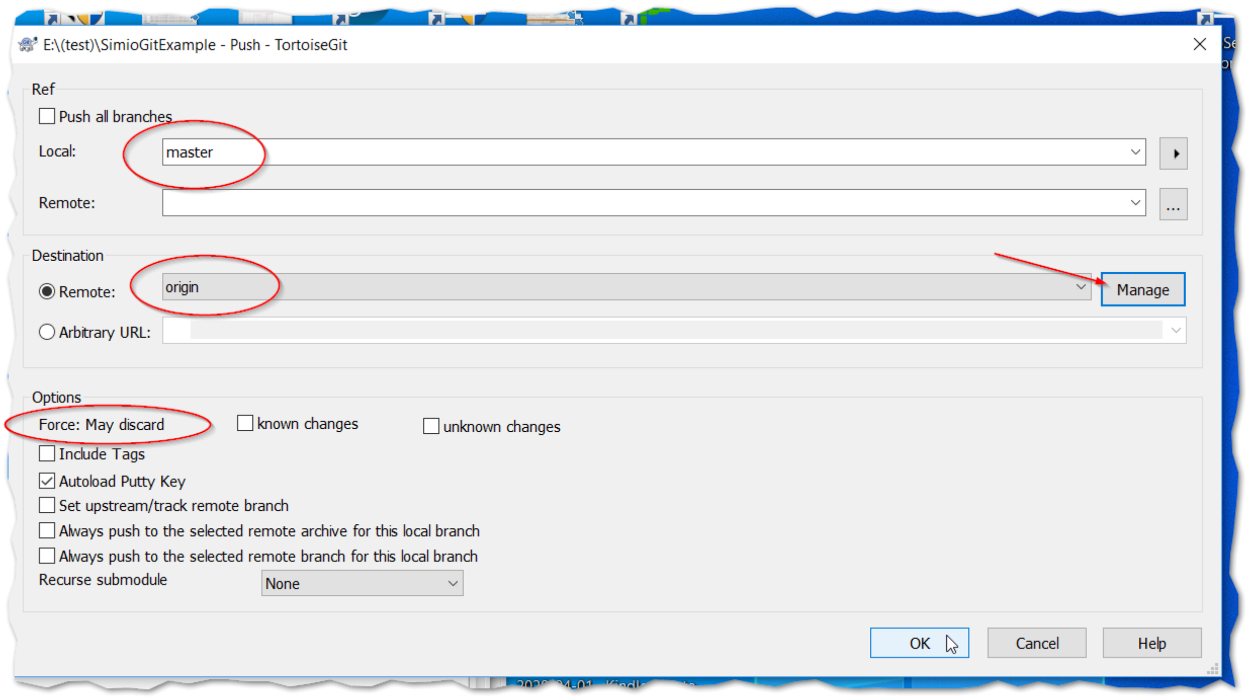


And then check "All" so recursively all files are pushed to GitHub.



Make it Bare (No working directories)

If you plan to work inside this folder, leave this unchecked. Typically a bare repo can only have changes 'pushed' to it. (By convention, a bare repo folder should have a name that ends with '.git')



E:\(test)\SimioGitExample - Push - TortoiseGit

Ref

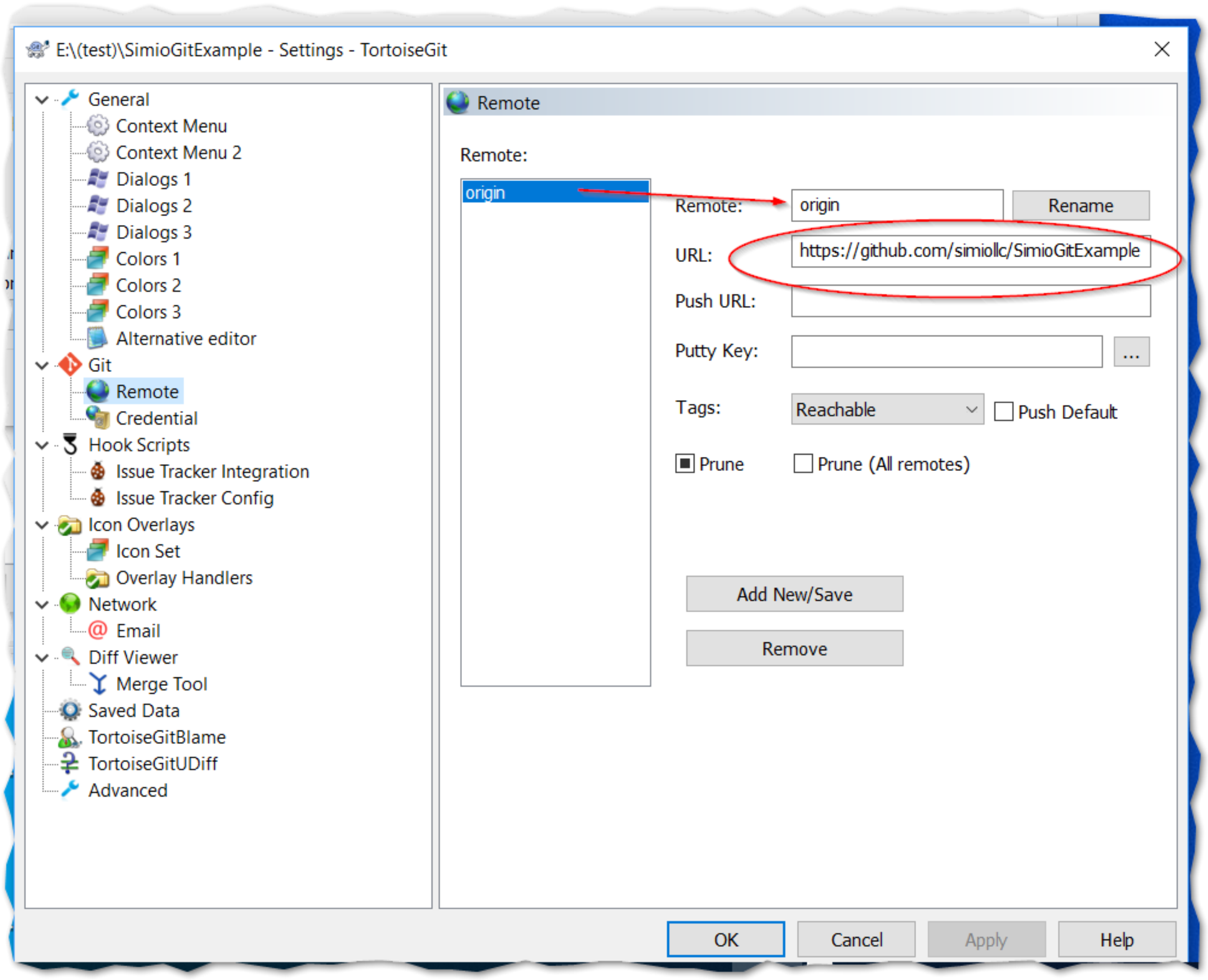
- [ ] Push all branches
- Local: master
- Remote:

Destination

- (●) Remote: origin
- ( ) Arbitrary URL:

Options

- Force: May discard
- [ ] known changes
- [ ] unknown changes
- [ ] Include Tags
- [✓] Autoload Putty Key
- [ ] Set upstream/track remote branch
- [ ] Always push to the selected remote archive for this local branch
- [ ] Always push to the selected remote branch for this local branch
- Recurse submodule: None



E:\(test)\SimioGitExample - Settings - TortoiseGit

Remote

Remote: origin

Remote: origin

URL: https://github.com/simiollc/SimioGitExample

Push URL:

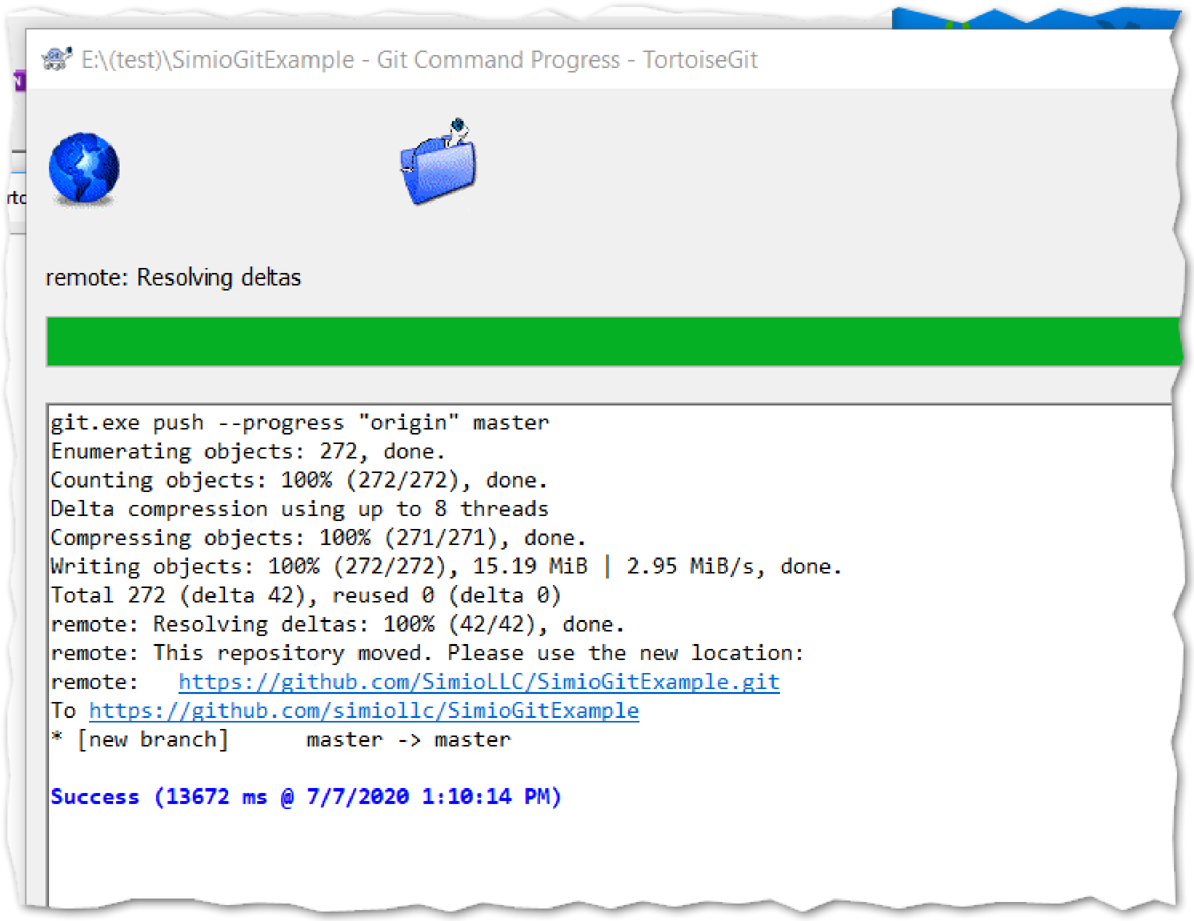
Putty Key:

Tags: Reachable

Push Default: ☐

Prune: ■

Prune (All remotes): ☐

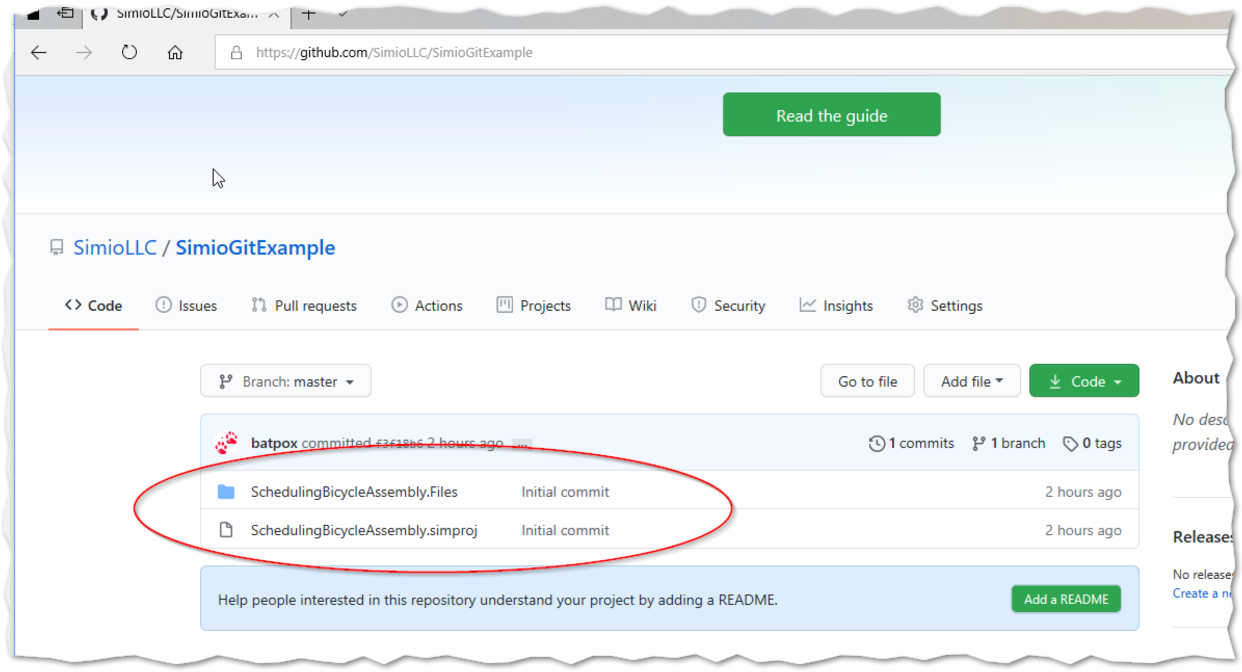

E:\(test)\SimioGitExample - Git Command Progress - TortoiseGit
remote: Resolving deltas

```
git.exe push --progress "origin" master
Enumerating objects: 272, done.
Counting objects: 100% (272/272), done.
Delta compression using up to 8 threads
Compressing objects: 100% (271/271), done.
Writing objects: 100% (272/272), 15.19 MiB | 2.95 MiB/s, done.
Total 272 (delta 42), reused 0 (delta 0)
remote: Resolving deltas: 100% (42/42), done.
remote: This repository moved. Please use the new location:
remote:    https://github.com/SimioLLC/SimioGitExample.git
To https://github.com/simiollc/SimioGitExample
* [new branch]      master -> master

Success (13672 ms @ 7/7/2020 1:10:14 PM)
```

And now you can go to the Repository on GitHub and examine it:

Simio
Forward Thinking

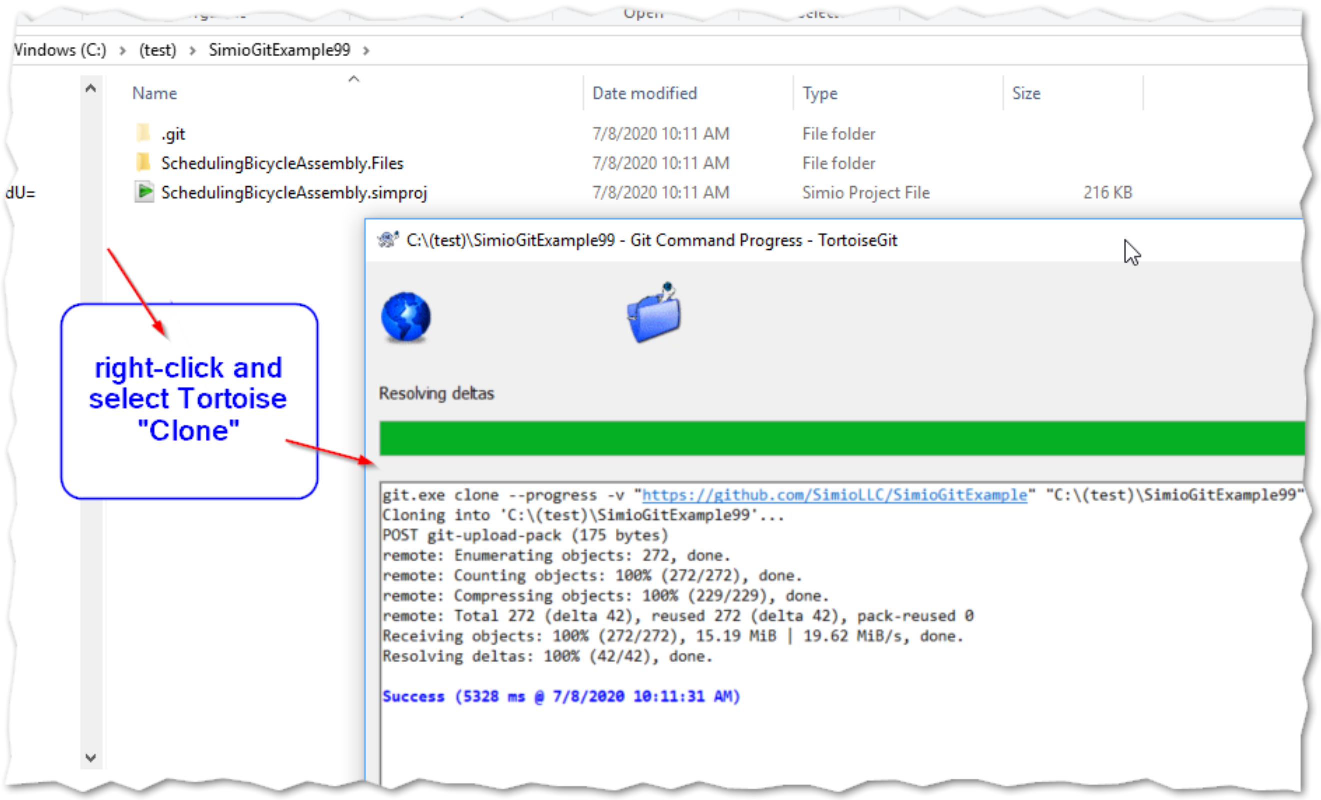


Setting up a New User

So now we have a global Git repository.

When you have a new user, they must clone a copy of the repository locally.

Fortunately, this is much simpler than building the initial GitHub repository.

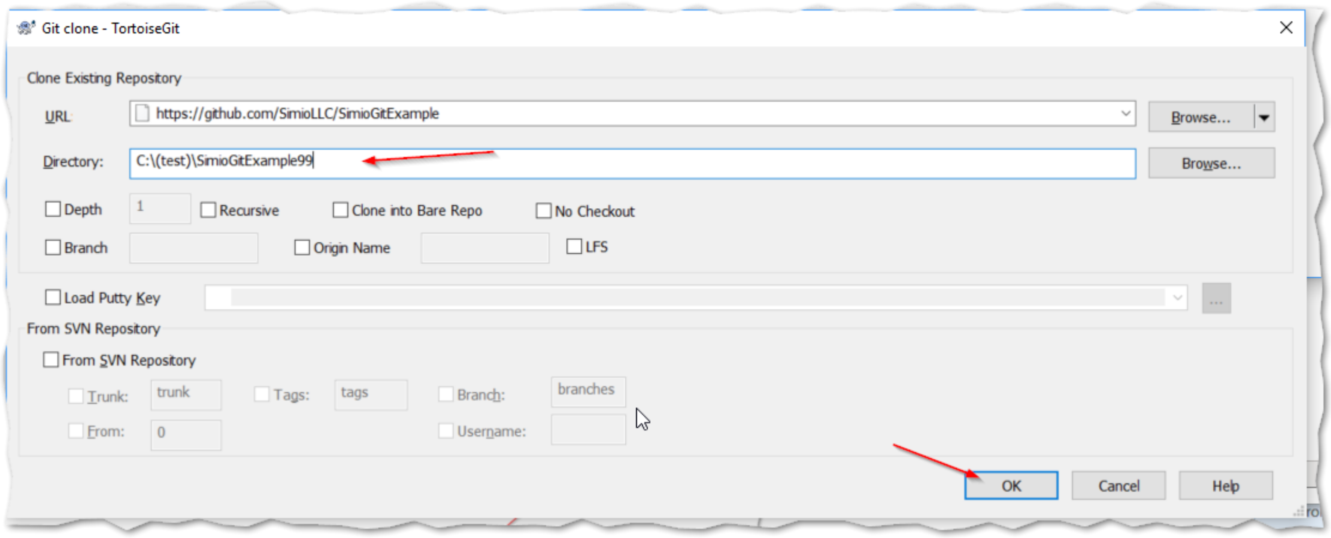

Windows (C:) > (test) > SimioGitExample99 >
Name
Date modified
Type
Size
.git
SchedulingBicycleAssembly.Files
SchedulingBicycleAssembly.simproj
right-click and select Tortoise "Clone"
C:\(test)\SimioGitExample99 - Git Command Progress - TortoiseGit
Resolving deltas

Name	Date modified	Type	Size
.git	7/8/2020 10:11 AM	File folder	
SchedulingBicycleAssembly.Files	7/8/2020 10:11 AM	File folder	
SchedulingBicycleAssembly.simproj	7/8/2020 10:11 AM	Simio Project File	216 KB

```
git.exe clone --progress -v "https://github.com/SimioLLC/SimioGitExample" "C:\(test)\SimioGitExample99"
Cloning into 'C:\(test)\SimioGitExample99'...
POST git-upload-pack (175 bytes)
remote: Enumerating objects: 272, done.
remote: Counting objects: 100% (272/272), done.
remote: Compressing objects: 100% (229/229), done.
remote: Total 272 (delta 42), reused 272 (delta 42), pack-reused 0
Receiving objects: 100% (272/272), 15.19 MiB | 19.62 MiB/s, done.
Resolving deltas: 100% (42/42), done.

Success (5328 ms @ 7/8/2020 10:11:31 AM)
```

This is the Tortoise Dialog when you selected “Clone”



URL: https://github.com/SimioLLC/SimioGitExample

Directory: C:\(test)\SimioGitExample99

Depth: 1 · Trunk: trunk · Tags: tags · Branch: branches · From: 0

And that is all there is to it. A full copy of the project is now placed in the folder and is ready for running by Simio.

Demonstrations

This section contains several demonstrations of using GitHub.

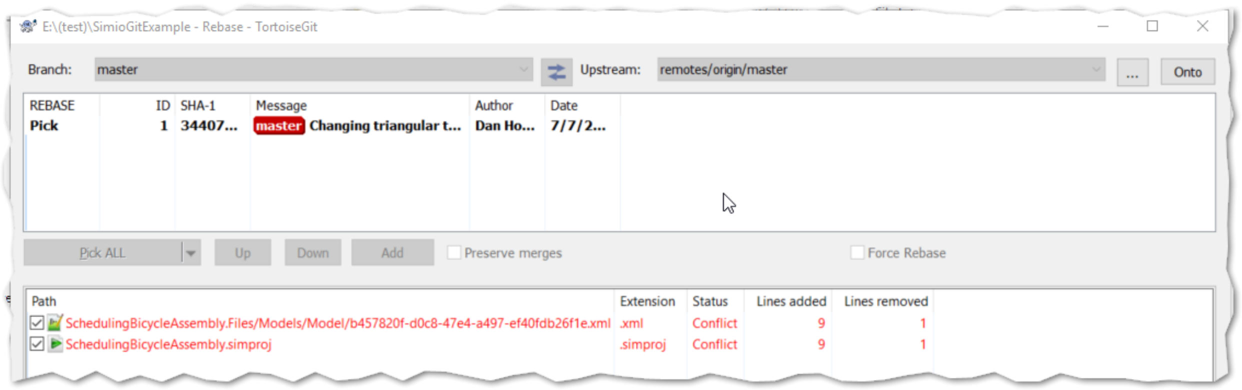
Demonstration 1: Make a Change in Simio

This is a demonstration of a single user making changes in Simio and committing or "Pushing" those changes to the GitHub repository using Tortoise Git.

The workflow for committing a change is:

1. Run Simio
2. Make changes and press "Save". This updates the Simio files on disk.
3. Go to File Explorer, right-click and

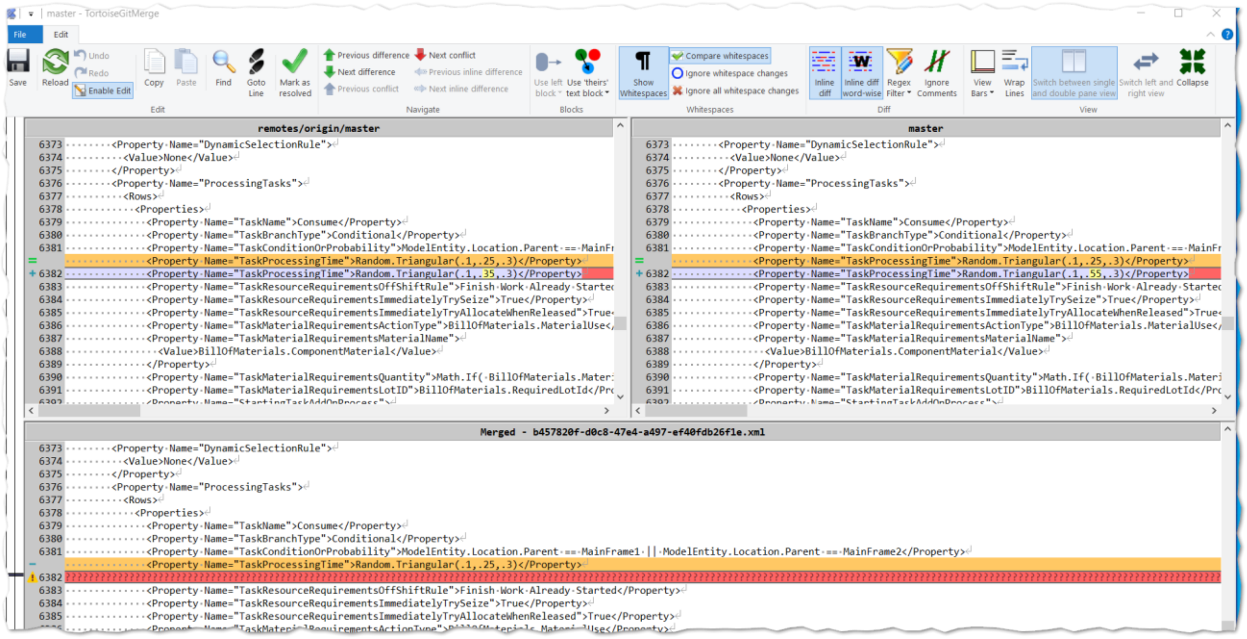
Demonstration 2: Making Conflicting Changes



Branch: master
Upstream: remotes/origin/master

REBASE	ID	SHA-1	Message	Author	Date
Pick	**1**	**34407...**	master **Changing triangular t...**	**Dan Ho...**	**7/7/2...**

Path	Extension	Status	Lines added	Lines removed
☑ SchedulingBicycleAssembly.Files/Models/Model/b457820f-d0c8-47e4-a497-ef40fdb26f1e.xml	.xml	Conflict	9	1
☑ SchedulingBicycleAssembly.simproj	.simproj	Conflict	9	1



remotes/origin/master

```
        <Property Name="DynamicSelectionRule">
          <Value>None</Value>
        </Property>
        <Property Name="ProcessingTasks">
          <Rows>
            <Properties>
              <Property Name="TaskName">Consume</Property>
              <Property Name="TaskBranchType">Conditional</Property>
              <Property Name="TaskConditionOrProbability">ModelEntity.Location.Parent == MainFr
              <Property Name="TaskProcessingTime">Random.Triangular(.1,.25,.3)</Property>
              <Property Name="TaskProcessingTime">Random.Triangular(.1,.35,.3)</Property>
              <Property Name="TaskResourceRequirementsOffShiftRule">Finish Work Already Startec
              <Property Name="TaskResourceRequirementsImmediatelyTrySeize">True</Property>
              <Property Name="TaskResourceRequirementsImmediatelyTryAllocateWhenReleased">True<
              <Property Name="TaskMaterialRequirementsActionType">BillOfMaterials.MaterialUse</
              <Property Name="TaskMaterialRequirementsMaterialName">
                <Value>BillOfMaterials.ComponentMaterial</Value>
              </Property>
              <Property Name="TaskMaterialRequirementsQuantity">Math.If( BillOfMaterials.Materi
              <Property Name="TaskMaterialRequirementsLotID">BillOfMaterials.RequiredLotId</Prc
```

master

```
        <Property Name="DynamicSelectionRule">
          <Value>None</Value>
        </Property>
        <Property Name="ProcessingTasks">
          <Rows>
            <Properties>
              <Property Name="TaskName">Consume</Property>
              <Property Name="TaskBranchType">Conditional</Property>
              <Property Name="TaskConditionOrProbability">ModelEntity.Location.Parent == MainFr
              <Property Name="TaskProcessingTime">Random.Triangular(.1,.25,.3)</Property>
              <Property Name="TaskProcessingTime">Random.Triangular(.1,.55,.3)</Property>
              <Property Name="TaskResourceRequirementsOffShiftRule">Finish Work Already Startec
              <Property Name="TaskResourceRequirementsImmediatelyTrySeize">True</Property>
              <Property Name="TaskResourceRequirementsImmediatelyTryAllocateWhenReleased">True<
              <Property Name="TaskMaterialRequirementsActionType">BillOfMaterials.MaterialUse</
              <Property Name="TaskMaterialRequirementsMaterialName">
                <Value>BillOfMaterials.ComponentMaterial</Value>
              </Property>
              <Property Name="TaskMaterialRequirementsQuantity">Math.If( BillOfMaterials.Materi
              <Property Name="TaskMaterialRequirementsLotID">BillOfMaterials.RequiredLotId</Prc
```

Merged - b457820f-d0c8-47e4-a497-ef40fdb26f1e.xml

```
        <Property Name="DynamicSelectionRule">
          <Value>None</Value>
        </Property>
        <Property Name="ProcessingTasks">
          <Rows>
            <Properties>
              <Property Name="TaskName">Consume</Property>
              <Property Name="TaskBranchType">Conditional</Property>
              <Property Name="TaskConditionOrProbability">ModelEntity.Location.Parent == MainFrame1 || ModelEntity.Location.Parent == MainFrame2</Property>
              <Property Name="TaskProcessingTime">Random.Triangular(.1,.25,.3)</Property>
????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????
              <Property Name="TaskResourceRequirementsOffShiftRule">Finish Work Already Started</Property>
              <Property Name="TaskResourceRequirementsImmediatelyTrySeize">True</Property>
              <Property Name="TaskResourceRequirementsImmediatelyTryAllocateWhenReleased">True</Property>
```

There are two conflicts:

1. One arising from the change of the distribution parameters, and
2. The other from the main .SimProj file itself.

The second will happen every time two or more developers are working on the project simultaneously and then do a Simio Save followed by a Git Push. It happens because the date and hashcode are updated. For this you can always resolve by selecting "Mine"

The first requires a bit more thought. In this case, both developers made a change to the second argument of the triangle distribution. A decision must be made for which is correct.

Video:

Summary and the Road Ahead

As mentioned before, Simio is actively working internally on how to address the multi-developer issue and recognizes the importance of this feature.

Simio

Forward Thinking