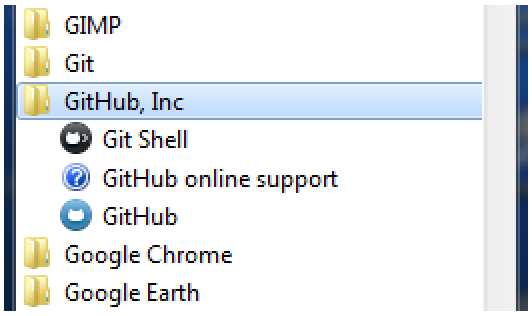
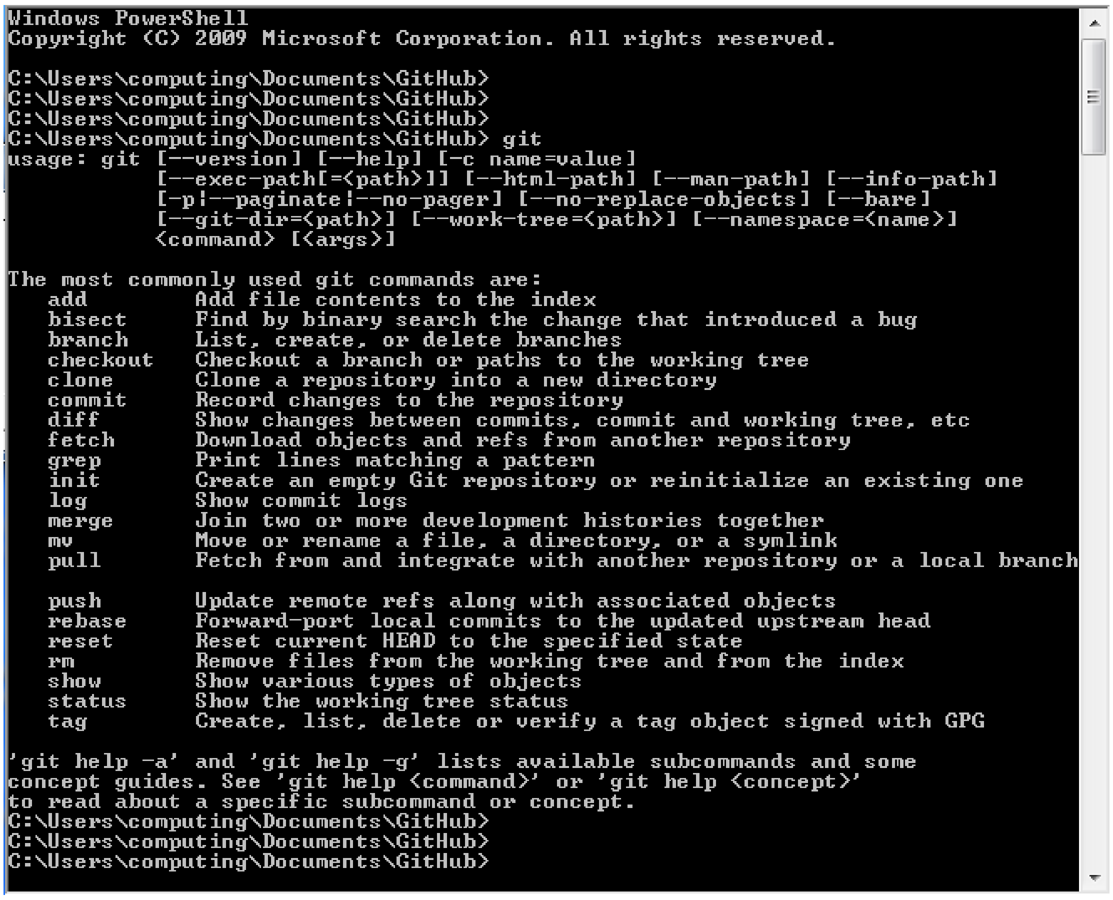
Introductory GitHub Tutorial

Git is a distributed revision control and source code management (SCM) system. The GitHub Application for Windows/Mac software is a application that wraps much of Git's functionality into an "easy to use" interface. You should note though that Git via the command line (e.g. clicking on "Git Shell") allows access to more powerful features, but you should probably read a good book on Git first before you attempt to use it (http://git-scm.com/book). This workshop will concentrate on using the GUI application.

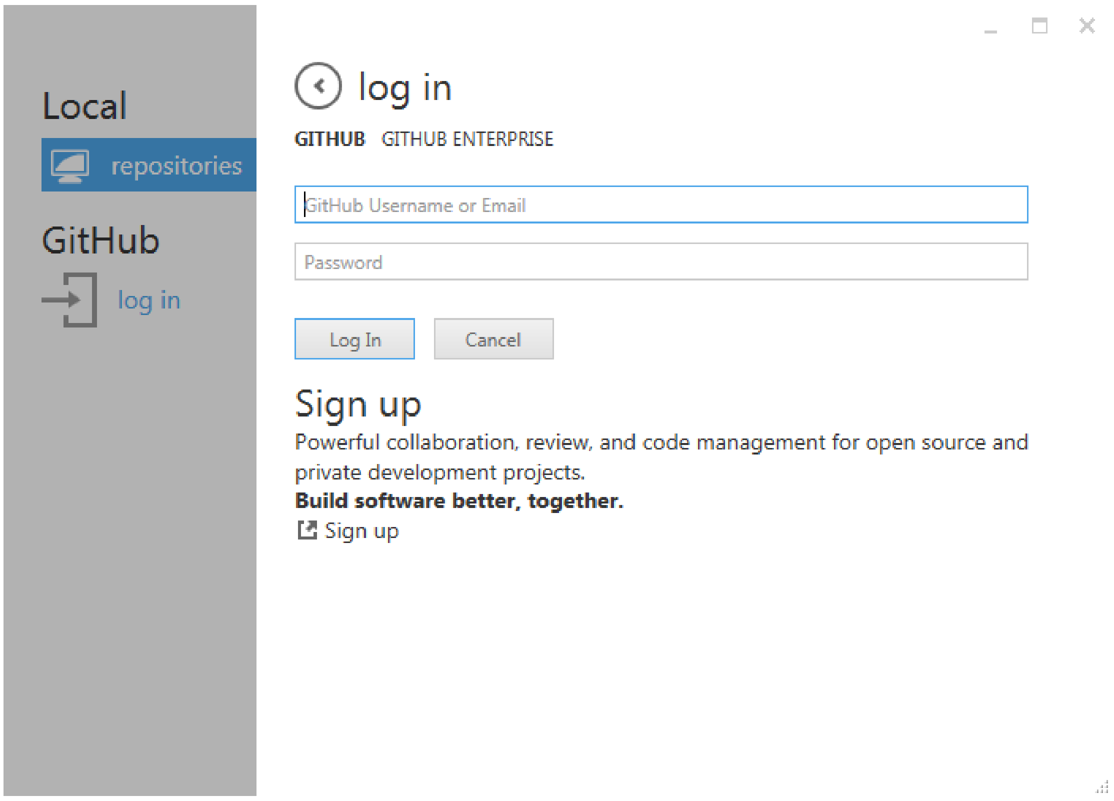


Locate the software as above



```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

C:\Users\computing\Documents\GitHub>
C:\Users\computing\Documents\GitHub>
C:\Users\computing\Documents\GitHub>
C:\Users\computing\Documents\GitHub> git
usage: git [--version] [--help] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

The most commonly used git commands are:
   add        Add file contents to the index
   bisect     Find by binary search the change that introduced a bug
   branch     List, create, or delete branches
   checkout   Checkout a branch or paths to the working tree
   clone      Clone a repository into a new directory
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   fetch      Download objects and refs from another repository
   grep       Print lines matching a pattern
   init       Create an empty Git repository or reinitialize an existing one
   log        Show commit logs
   merge      Join two or more development histories together
   mv         Move or rename a file, a directory, or a symlink
   pull       Fetch from and integrate with another repository or a local branch

   push       Update remote refs along with associated objects
   rebase     Forward-port local commits to the updated upstream head
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index
   show       Show various types of objects
   status     Show the working tree status
   tag        Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
C:\Users\computing\Documents\GitHub>
C:\Users\computing\Documents\GitHub>
C:\Users\computing\Documents\GitHub>
```

The command line is more powerful, but more complicated (some say).



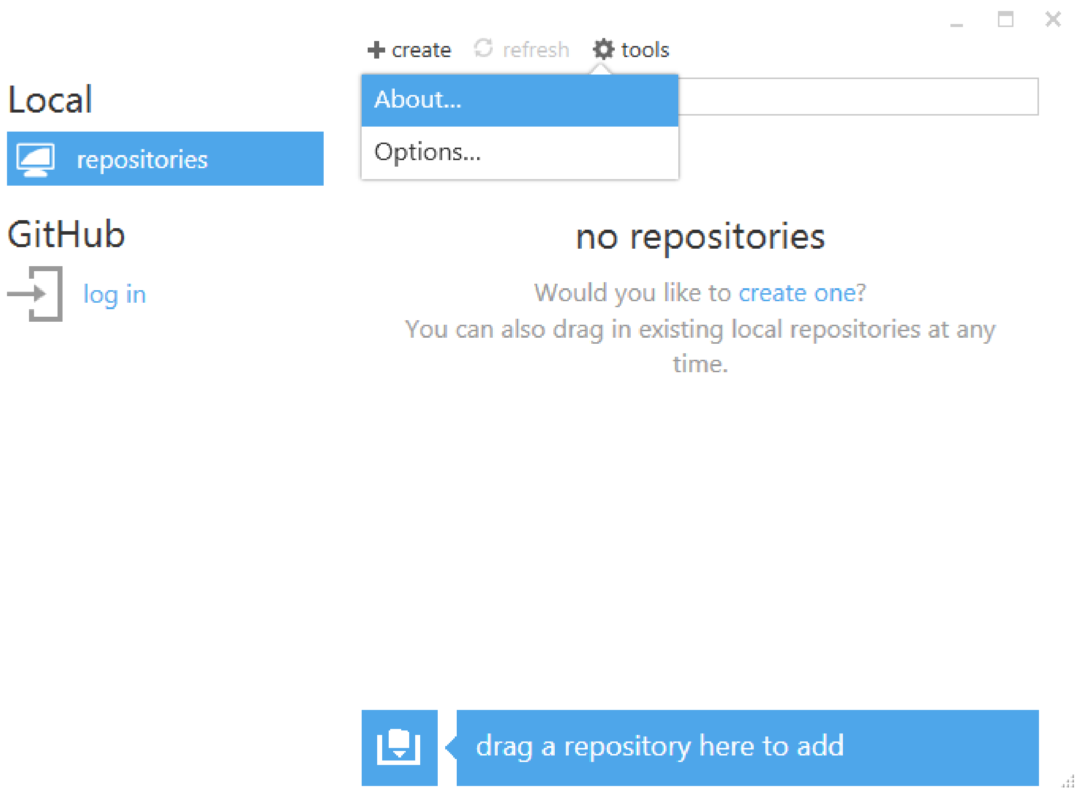
Sign up

Powerful collaboration, review, and code management for open source and private development projects.

Build software better, together.

Sign up

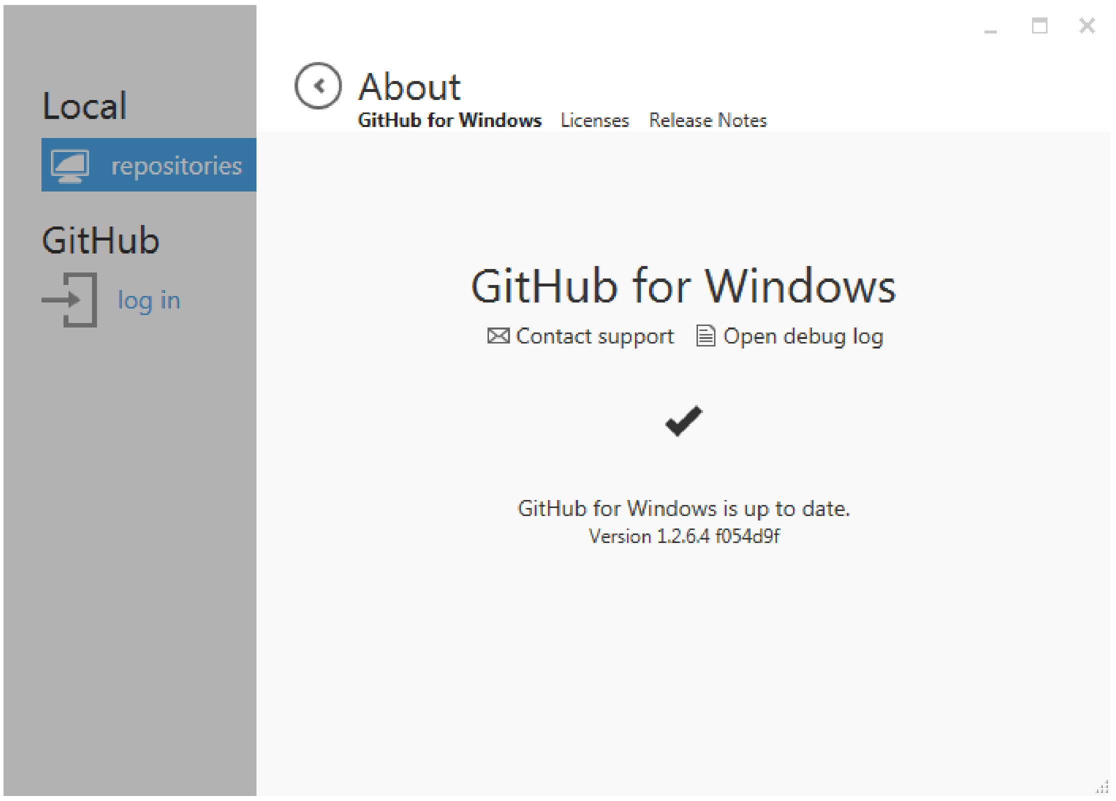
You do not need to login to github yet, the first part of this tutorial is about using Git locally (i.e. without a remote Git server). Just click the arrow next to "Log in" to skip.



no repositories

Would you like to create one?
You can also drag in existing local repositories at any time.

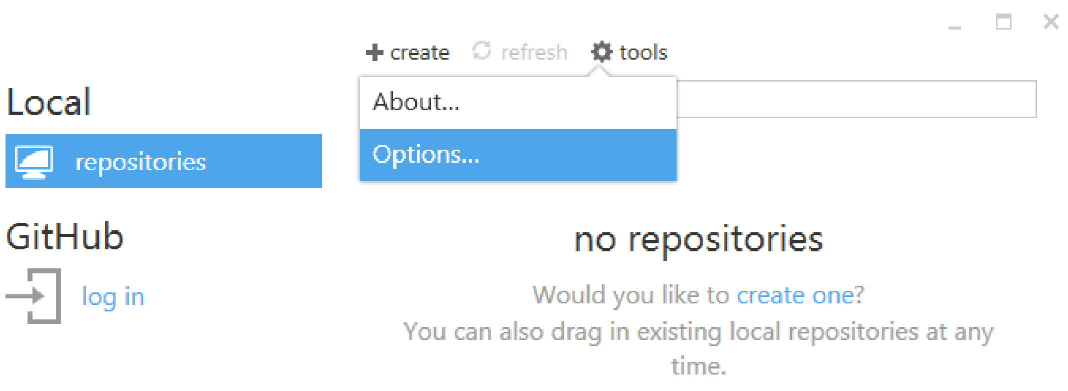
Click on “tools->About…” make sure your version is current.



GitHub for Windows

GitHub for Windows is up to date.
Version 1.2.6.4 f054d9f

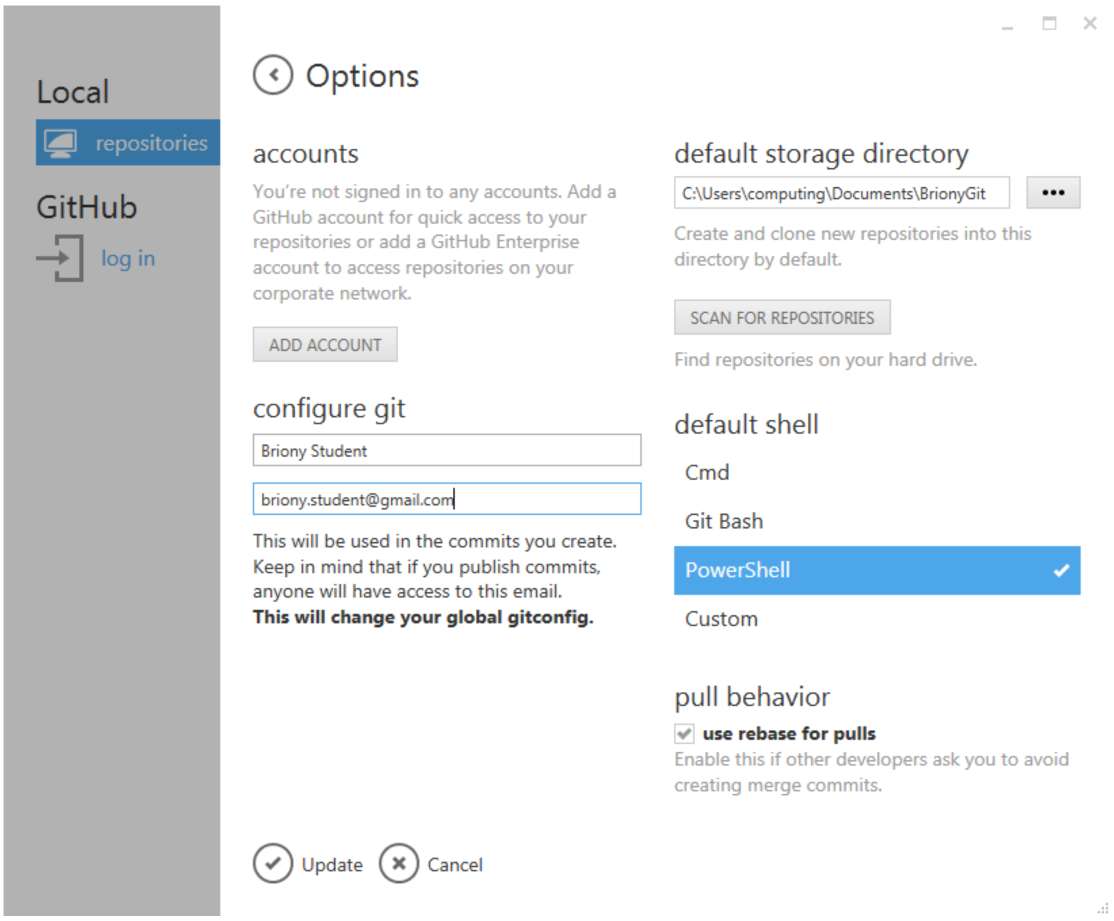
This tutorial was written using “Version 1.2.6.4 f054d9f”



no repositories

Would you like to create one?
You can also drag in existing local repositories at any time.

Each time you start using Git on a shared machine you will need to setup the options.



accounts

You're not signed in to any accounts. Add a GitHub account for quick access to your repositories or add a GitHub Enterprise account to access repositories on your corporate network.

configure git

Briony Student

briony.student@gmail.com

This will be used in the commits you create. Keep in mind that if you publish commits, anyone will have access to this email.
This will change your global gitconfig.

default storage directory

C:\Users\computing\Documents\BrionyGit

Create and clone new repositories into this directory by default.

Find repositories on your hard drive.

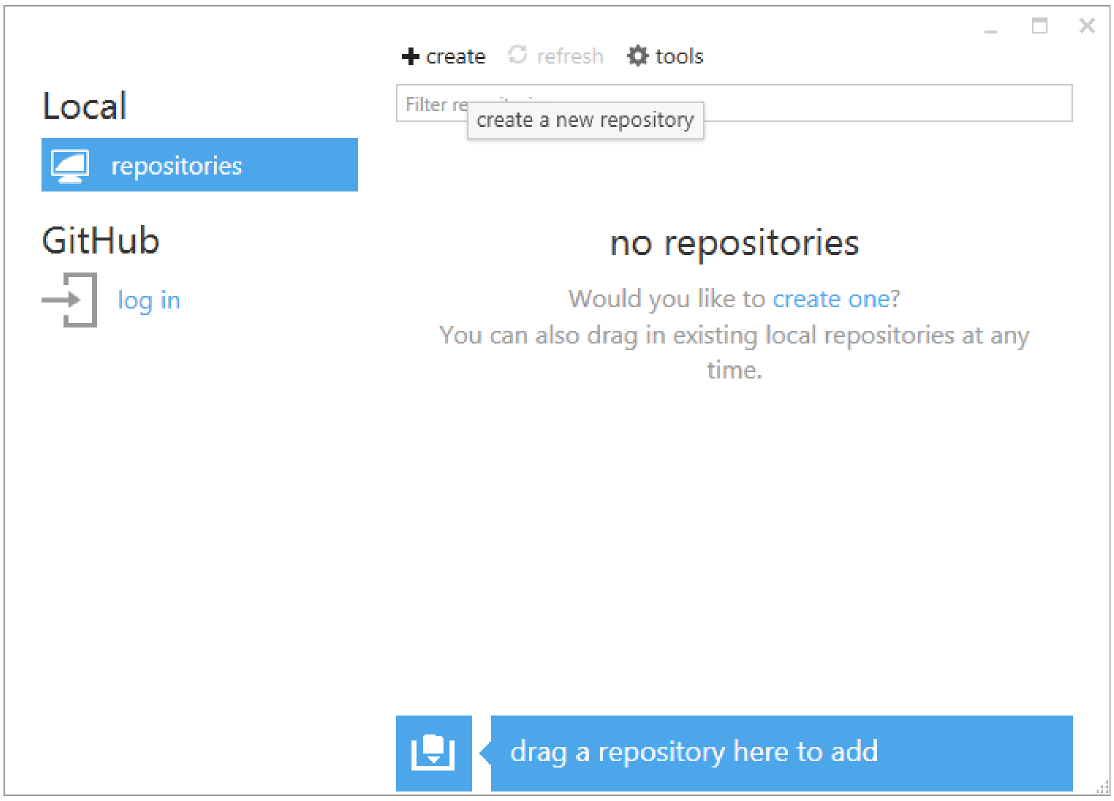
default shell

Cmd
Git Bash
[✓ PowerShell]
Custom

pull behavior

[✓] **use rebase for pulls**
Enable this if other developers ask you to avoid creating merge commits.

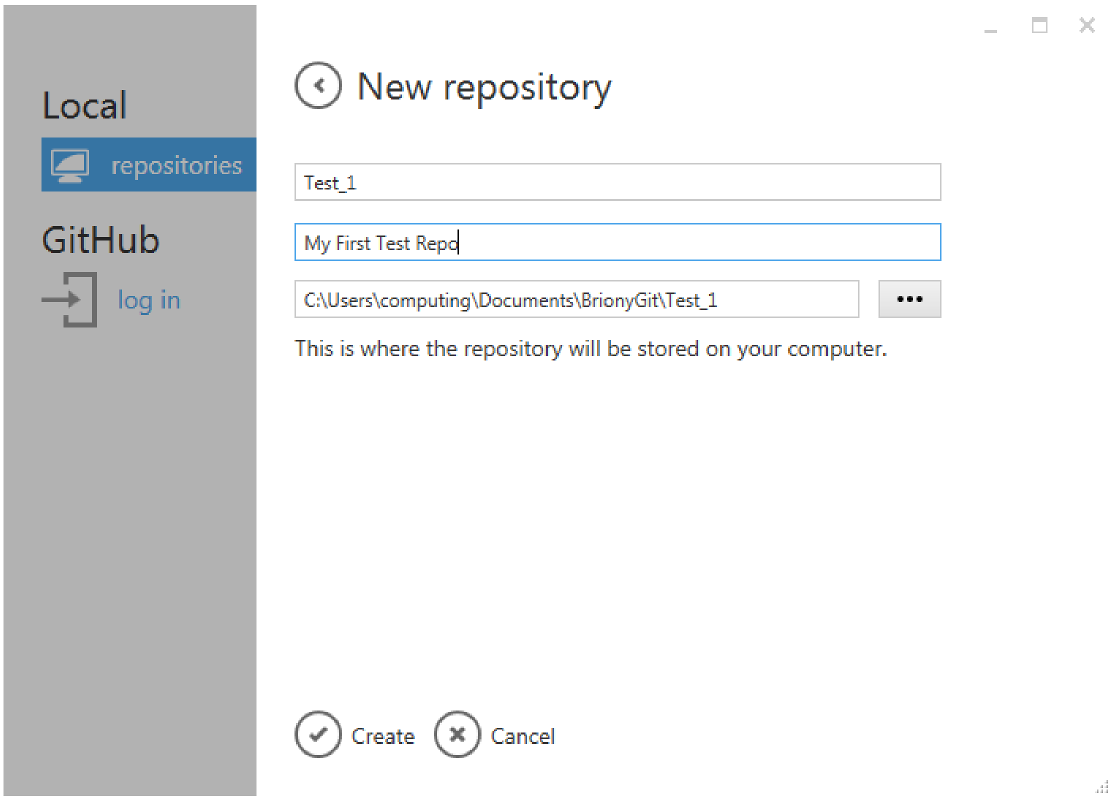
1) Enter your name, 2) Enter your email address, 3) Enter your storage directory. N.B. this **does not** log you into the github website, it just sets details that will be linked to any changes that you make, and says where these changes will be stored.



no repositories

Would you like to create one?
You can also drag in existing local repositories at any time.

Click "+ create" to start making your new repository.



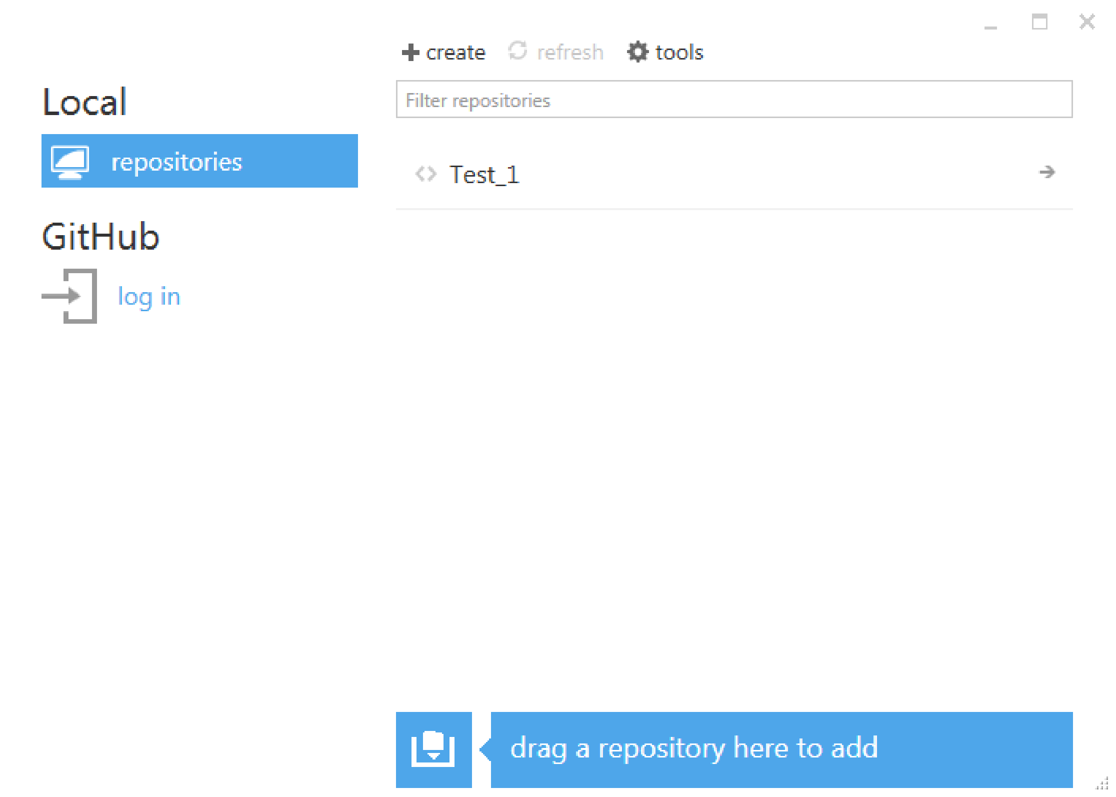
Test_1

My First Test Repo

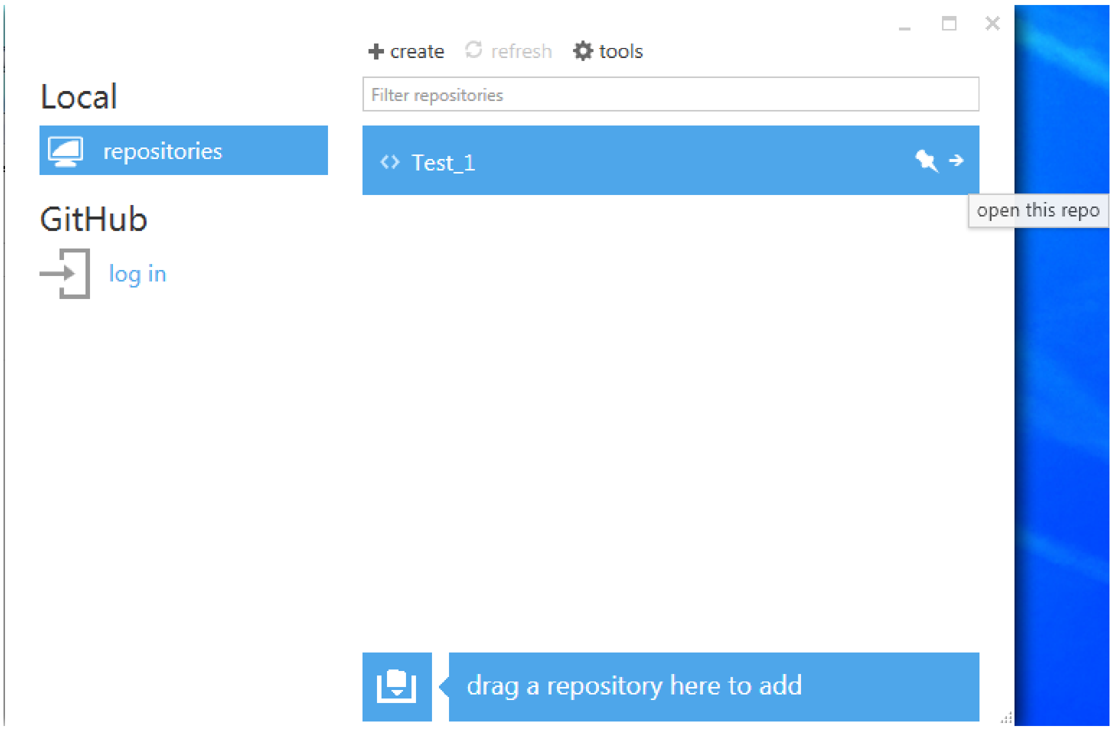
C:\Users\computing\Documents\BrionyGit\Test_1

This is where the repository will be stored on your computer.

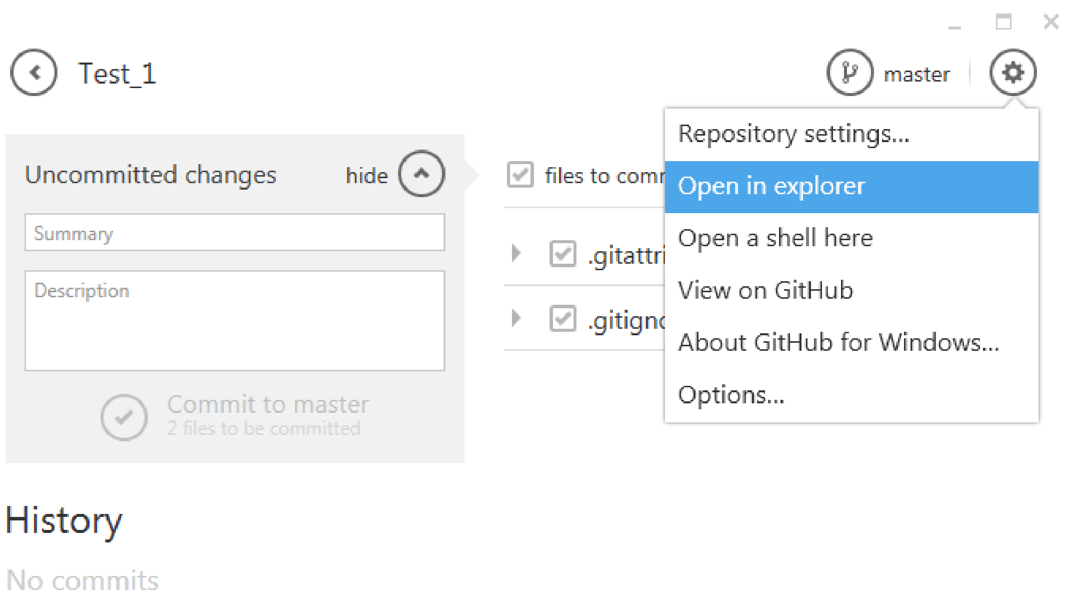
Type in a "Name" and a "Description" and click "Create"



Now your new repo has been created.

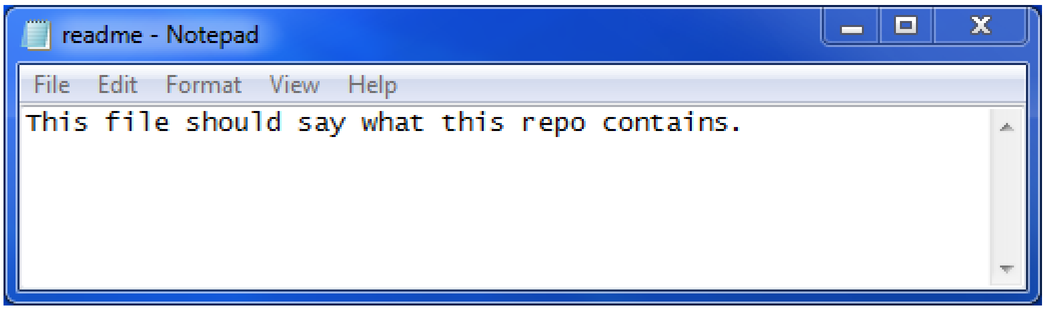


Click on the arrow and have a check to see the files that have been created.

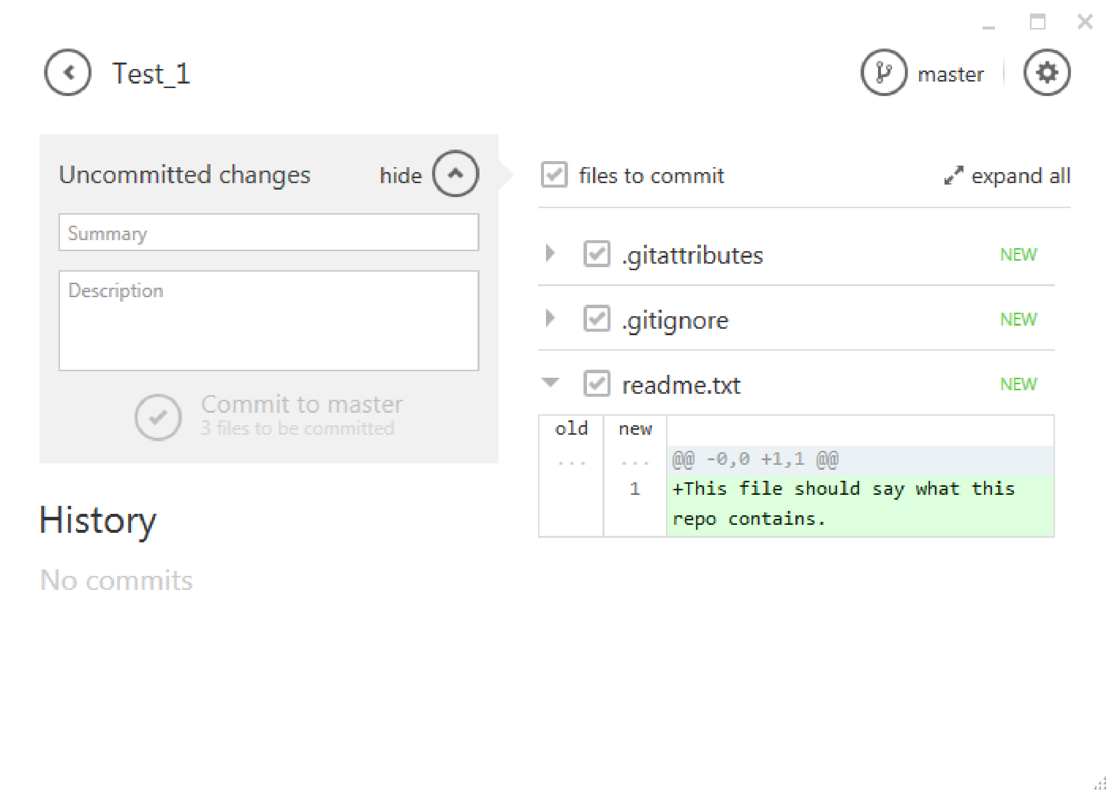


There should be a folder called “Test_1” this is where all the files in your repo will be stored. You can open this folder from the “tools and options” menu (as above). At the moment it should just contain two text files “.Gitattributes” and ‘Gitignore” (whose names will be hidden in explorer). These contain the “Repository settings…” for this repo (and you can ignore these for now, though they will need changing when you start using Unity, to stop Git trying to keep track of **everything**).

Create a file that you want your repo to include. Use a text editor, e.g. Notepad to create a file in the “Test_1” folder called “readme.txt” and add the following content.

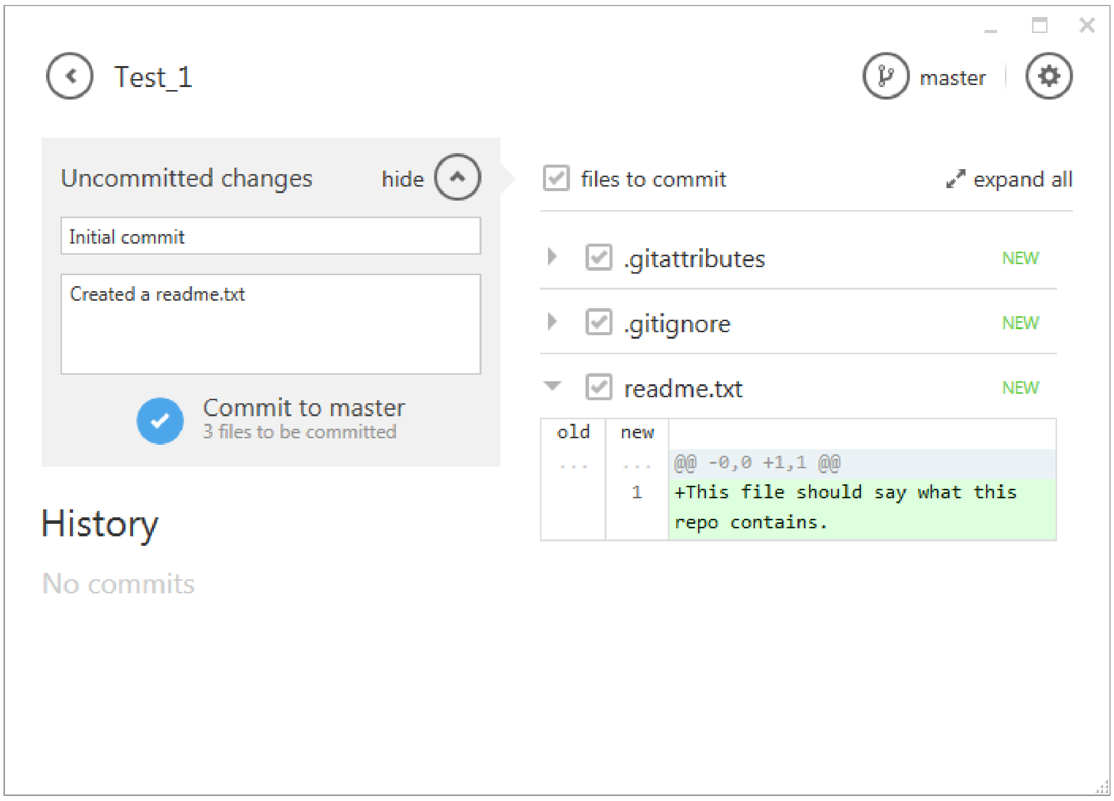


```
This file should say what this repo contains.
```



```
old  new
...  ...  @@ -0,0 +1,1 @@
     1    +This file should say what this
          repo contains.
```

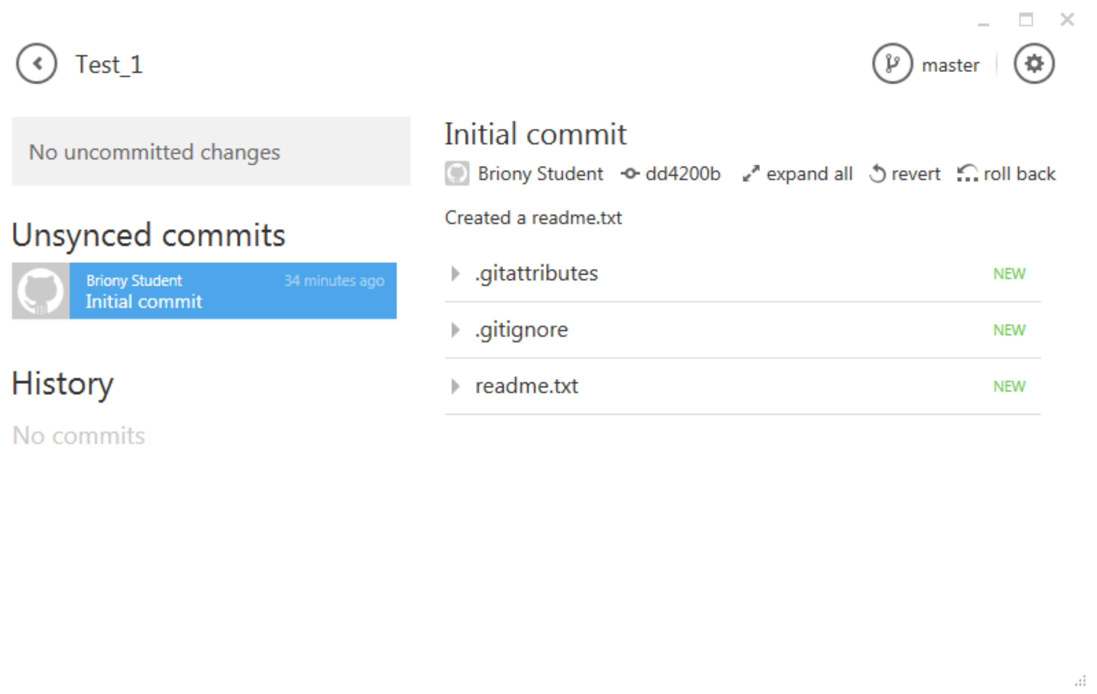
Go back to the repo and you should see the “readme.txt” file is now included. By clicking on the arrow near the file name you can see what changes have been made to the file. Even though this file is in the repo folder and it being tracked, it has not been “committed”, so if you deleted it, there would be no way to get it back. Let’s commit!



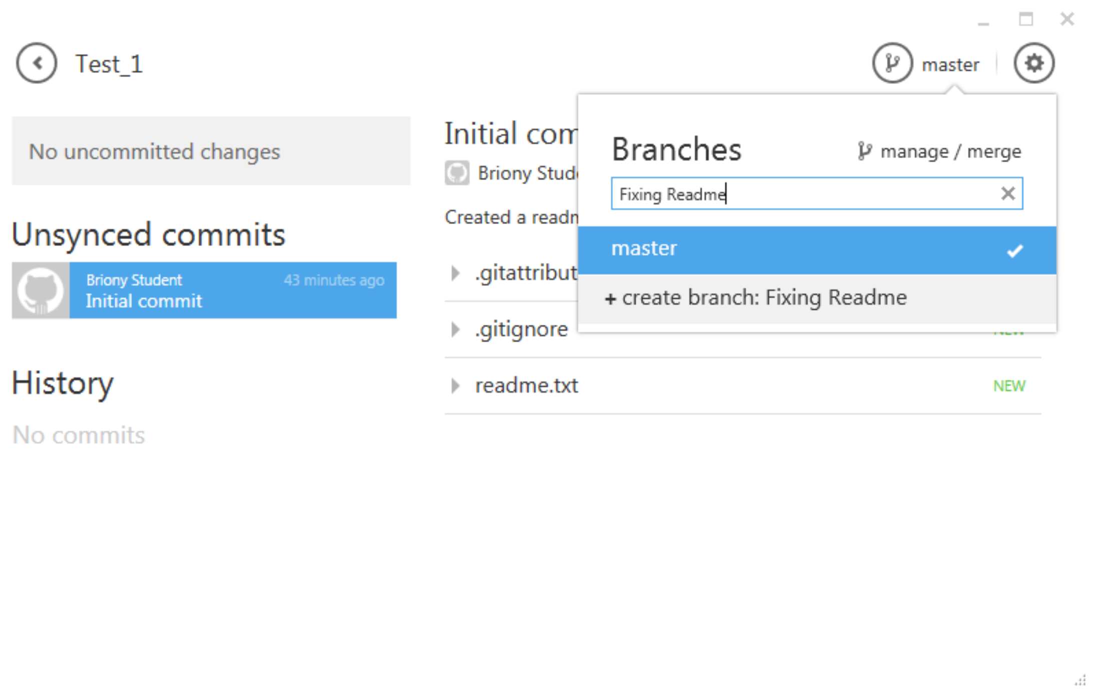
```
old  new
...  ...  @@ -0,0 +1,1 @@
     1    +This file should say what this
          repo contains.
```

Enter a “Summary” and “Description” and click “Commit to master”.

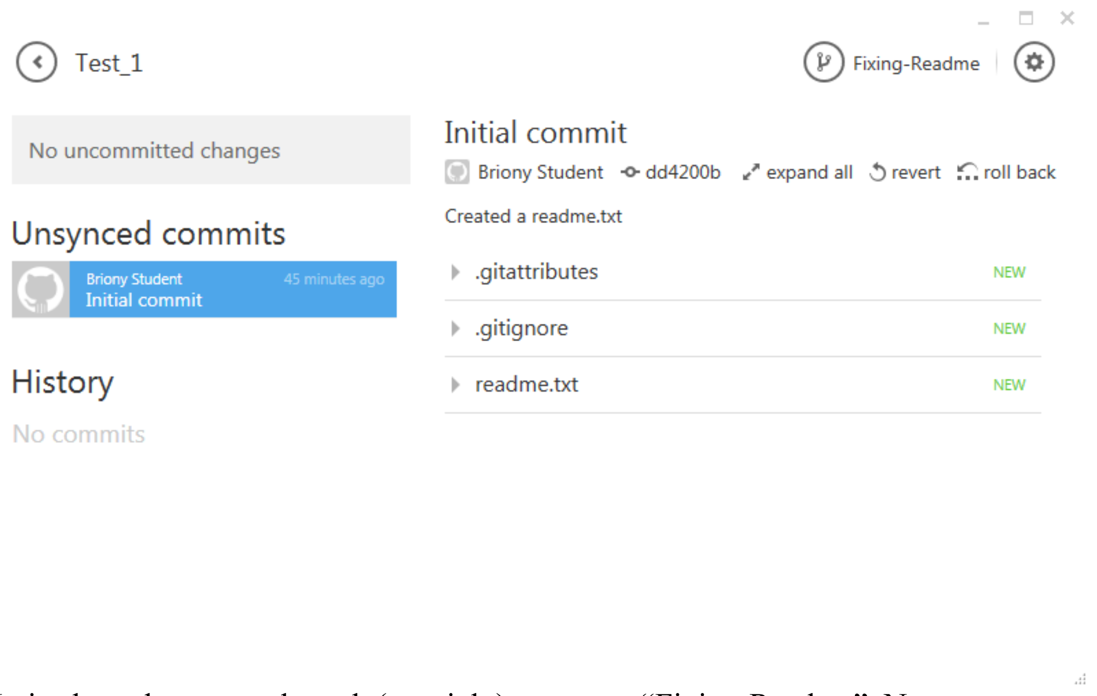
For interest, all of the repository files will go into a hidden subfolder called ".git", and the files you commit into the subfolders in ".git/objects/". You can have a look at this database in various ways (e.g. the shell command "Get-ChildItem –force –recurse" will show you everything), and for reference, the encoded filenames are created using SHA1. But for now, let's concentrate using the repo rather than how it works.



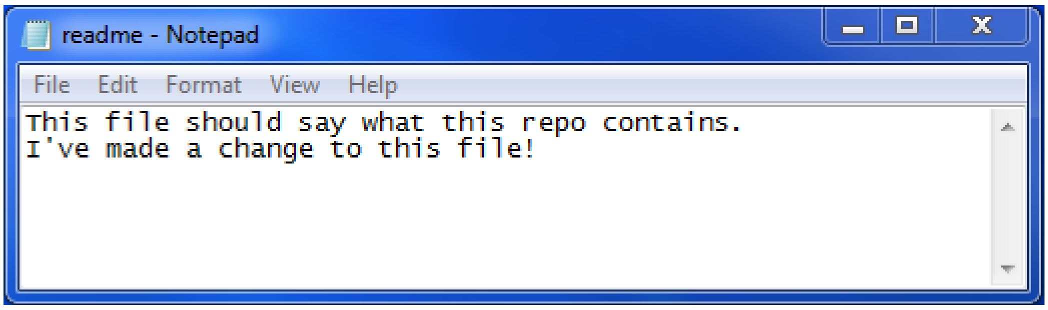
Note in the top-right corner is says "Master". This means we are working what is know as the **Master Branch** and this is generally a **bad idea**. The Master Branch is reserved for completed work, i.e. work that has been tested and agreed by the development team for inclusion in the current build. When you download the latest changes (e.g. so that you can include you own work and prevent yourself from working with outdated code), you do not want to be downloading a load of bugs that break your project. So it is essential that when you are developing you create your own branches to develop with. These are often just copies of the Master Branch, but you can mess them up without having to worry about breaking the project for everyone else.



Create a new branch called “Fixing Readme”.

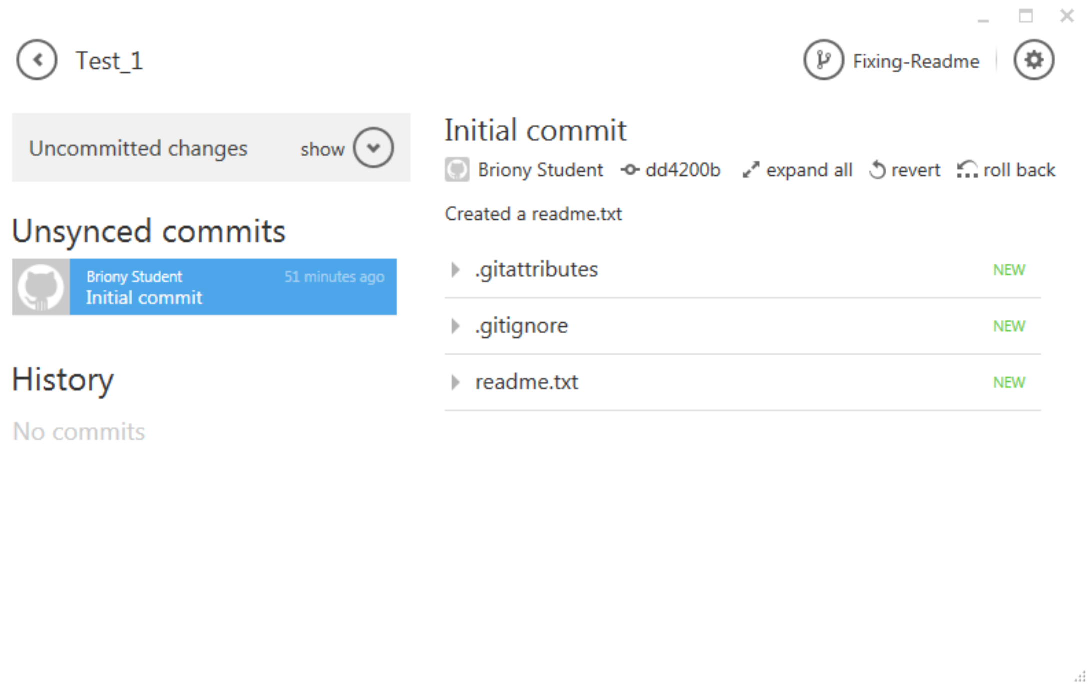


Notice how the current branch (top right) now says “Fixing-Readme”. Now we can make changes without worrying about breaking anything in the Master Branch

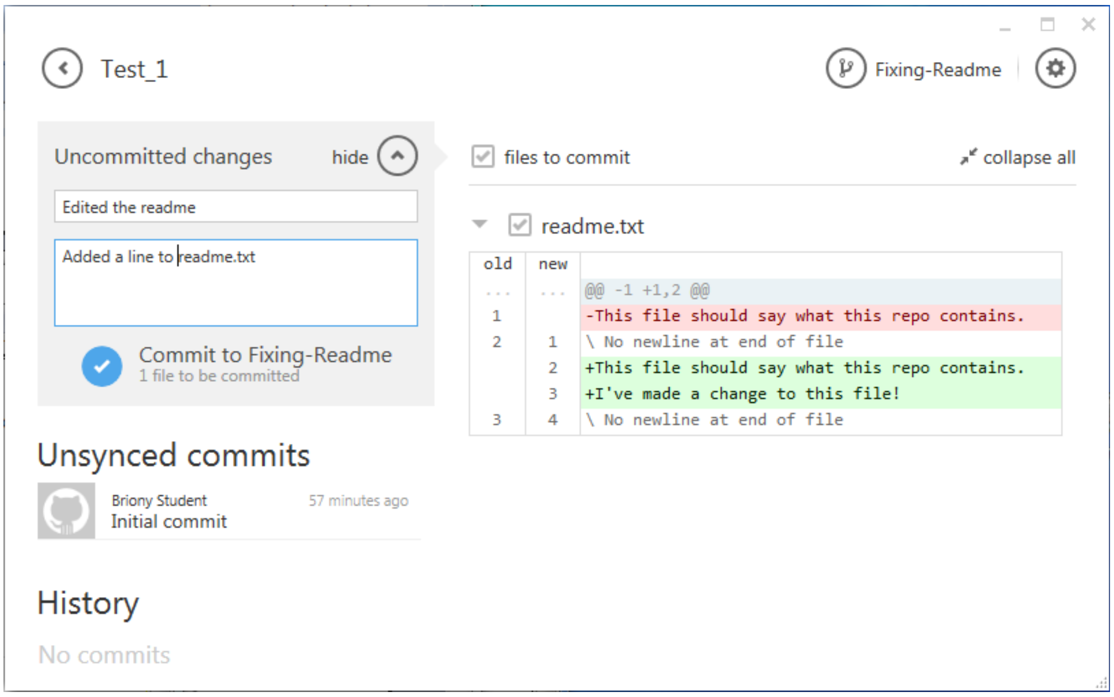


```
This file should say what this repo contains.
I've made a change to this file!
```

Open the file in Notepad and edit the text and save.



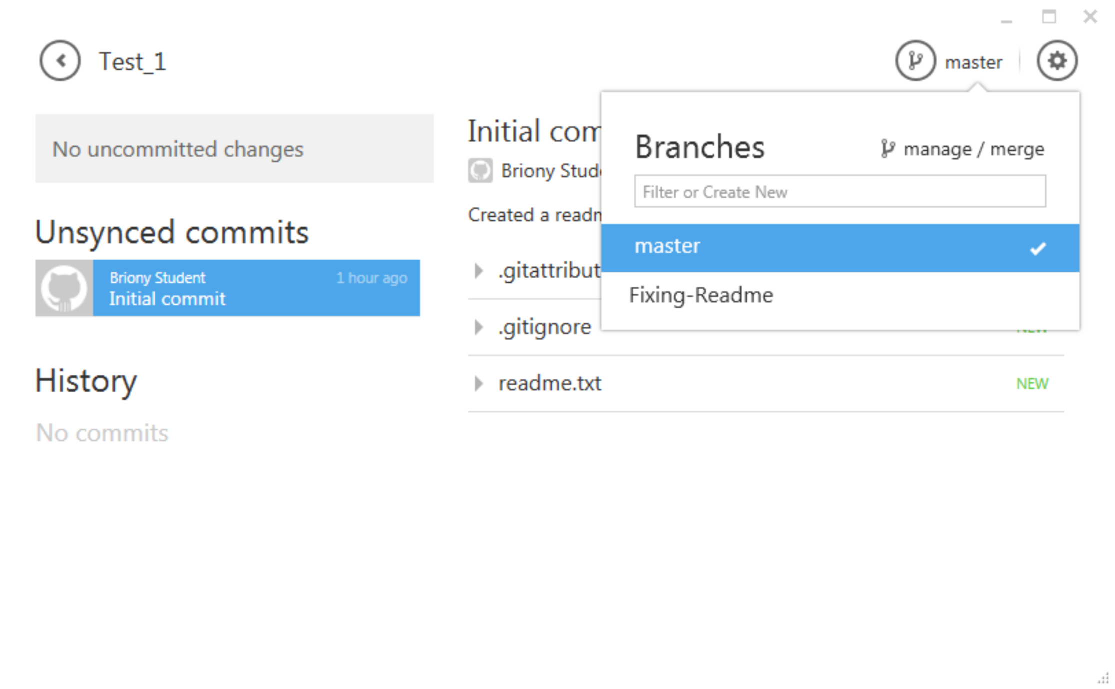
See that the application has now detected that some of your files have changed (it says, “Uncommitted changed in the top left). For interest, were you to now go into Notepad and undo your changes and re-save the file, then the application would detect that your file had gone back to its original state. It knows this because Git calculates the SHA1 from the contents of the file and uses it for the object’s filename. So, the same filename means the same contents (or at least it’s vanishingly unlikely that two files with the same SHA1 will contain something different to each other).



```
old  new
...  ...  @@ -1 +1,2 @@
1         -This file should say what this repo contains.
2    1    \ No newline at end of file
     2    +This file should say what this repo contains.
     3    +I've made a change to this file!
3    4    \ No newline at end of file
```

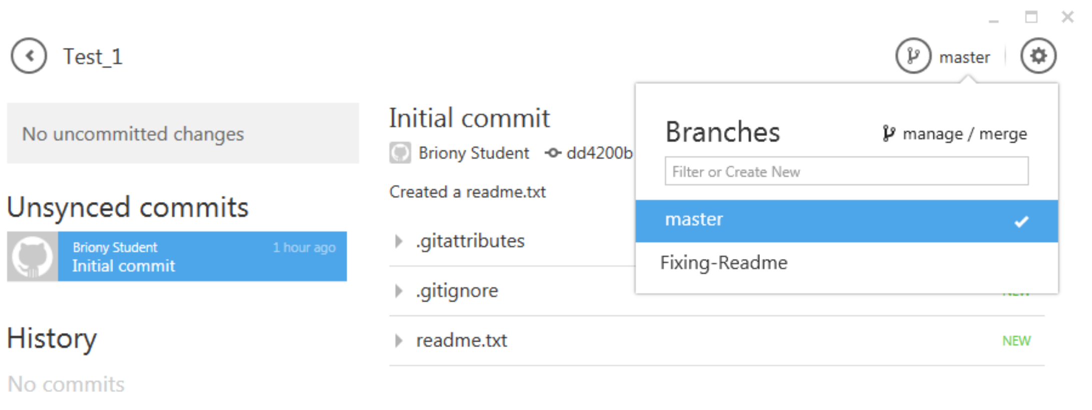
Enter a “Summary” and “Description” and click “Commit to Fixing-Readme”.
N.B. As before, if you do not **commit your changes**, they can be lost.

We should now have two committed branches, an original “Master Branch” and an edited “Fixing Readme” branch. You can change which branch the Test_1 folder currently contains (and which branch you would be working on if you made changes in that folder) by selecting the “switch branches” icon in the top right.

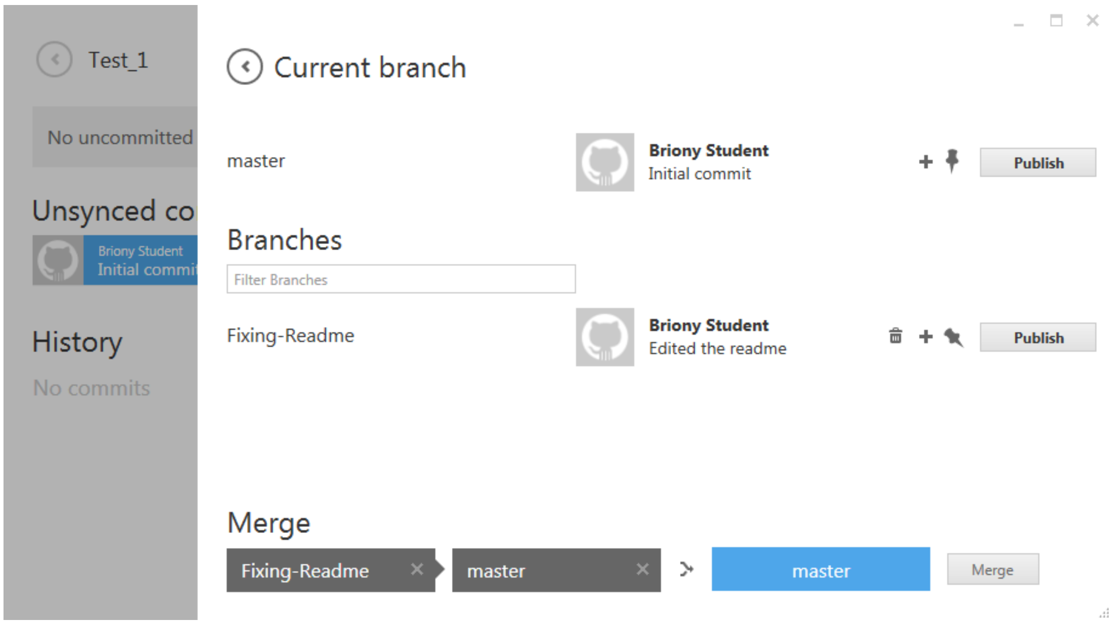


Try this:

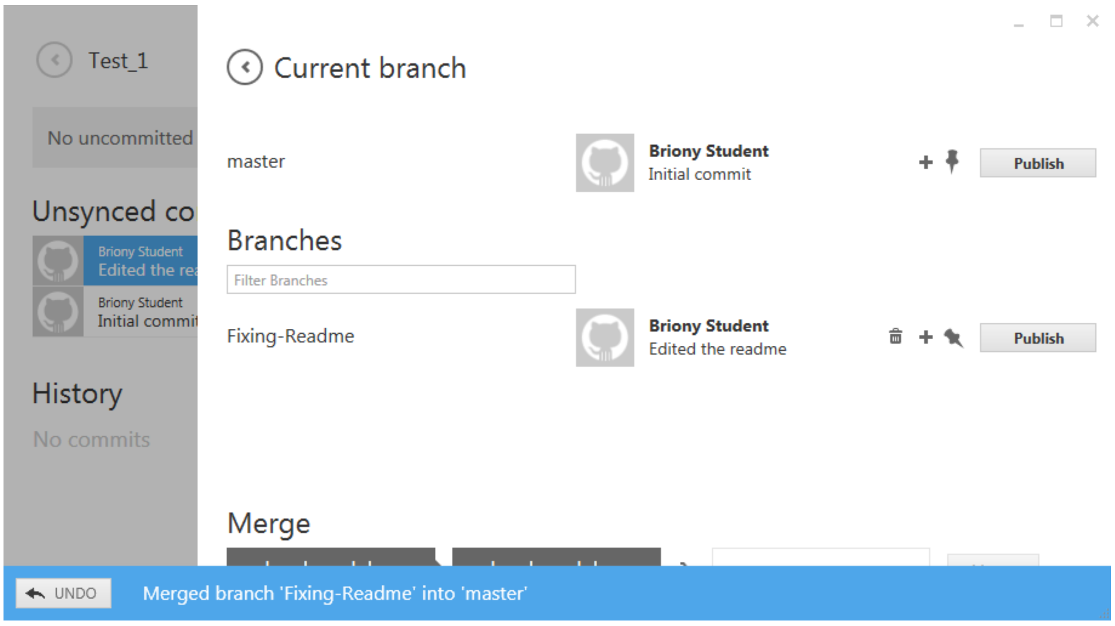
1) Select the master branch and open the readme.txt in Notepad
You should see the original (one line) contents.
2) Close Notepad, select the “Fixing-Readme” branch and open the readme
You should see the edited (two line) version.



If you are happy with the changes you have made in your branch and would like to incorporate them into the master branch then you need to merge them. In the “switch branches” menu (top right), click on “manage / merge”.



At the bottom, under where it says “Merge” drag the “Fixing-Readme” branch into the first box (i.e. the branch with the changes you want to add), and the “master” branch into the second box (i.e. the branch you want to make the changes to) and click “Merge”.



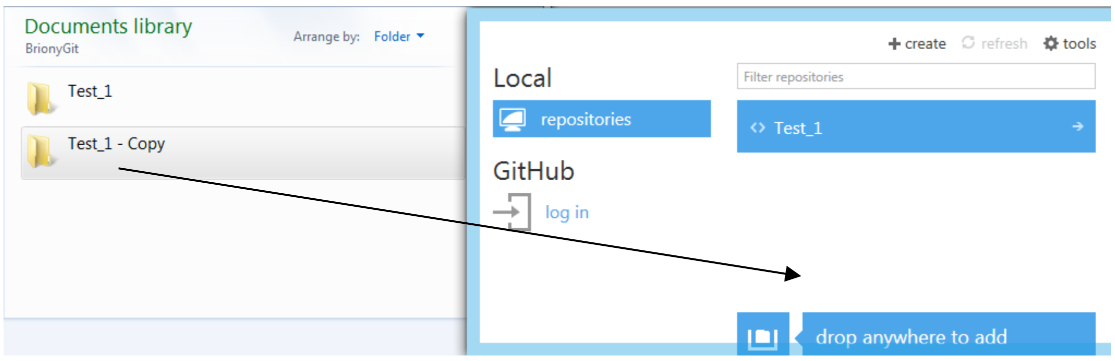
You should now find that the changes you made in your Fixing-Readme branch have been incorporated into the Master branch. You can check this by toggling between then two branches as before and checking the readme fine. This time there should be no difference between the branches.

Try this:

1) Select the master branch and open the readme.txt in Notepad
You should now see the merged (two line) contents.
2) Close Notepad, select the "Fixing-Readme" branch and open the readme
You should see the edited (two line) version.

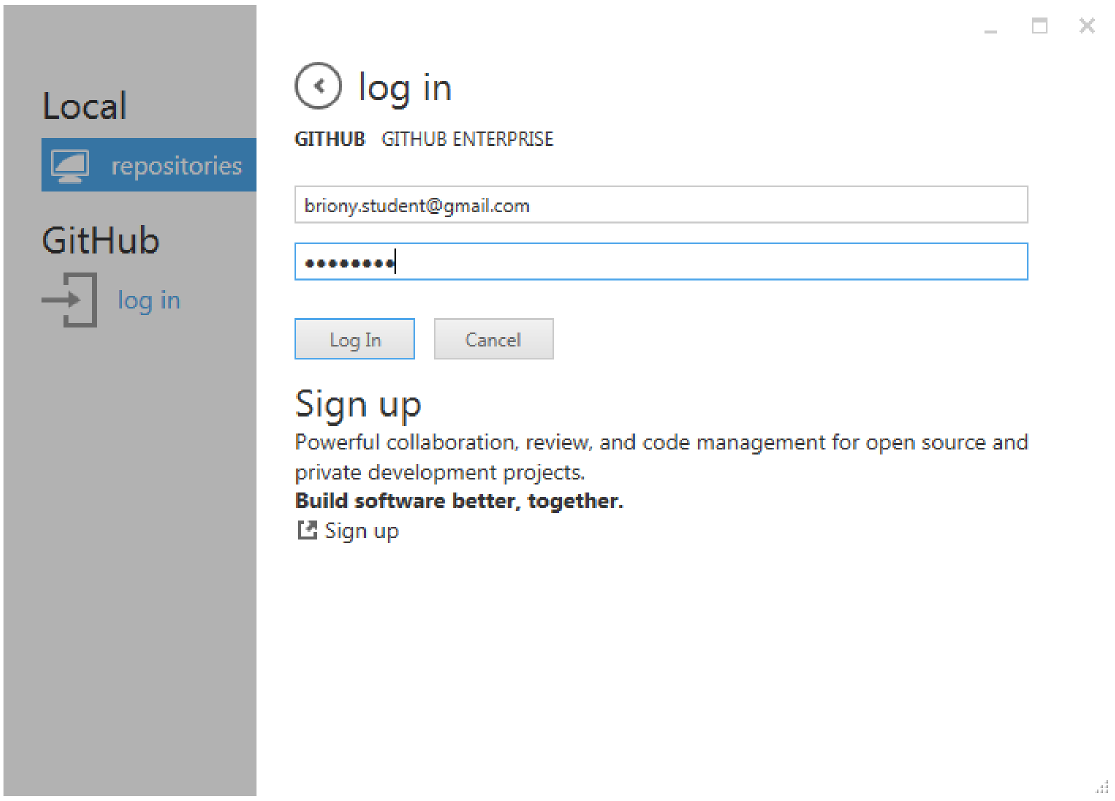
You might now feel that you no longer need the "Fixing-Readme" branch. If you are confident that your changes have now been successfully made to the Master branch you can delete your temporary working branch by clicking the trashcan icon in the "manage / merge" panel. Just make sure that the temporary branch is not selected as the current branch, since you cannot delete that.

You can also try copying Test_1 folder (e.g. drag and drop in windows explorer to make "Test_1 – Copy"). This is an entire copy of your whole repository, i.e. includes all the changes that you have committed, as well as the current working branch. You can add this so it is tracked in the application by dragging in the copy.



This is key to understanding how to work online with Git repositories. You can have multiple copies of the entire archive (each developer has one, and there is one at the github website). To collaborate then, you make your own copy of the repo, make some private branches, edit these, then merge these changes into a public branch (e.g. the master branch if everyone is happy with them, although other branches can be shared too) and then push the changes to shared branches up to the server where the rest of the team can get hold of them.[1]

Lets add our Test_1 repo to the github server (remember, in effect this just means copying all the files from the Test_1 folder to github, though we let the application take care of the actual work).

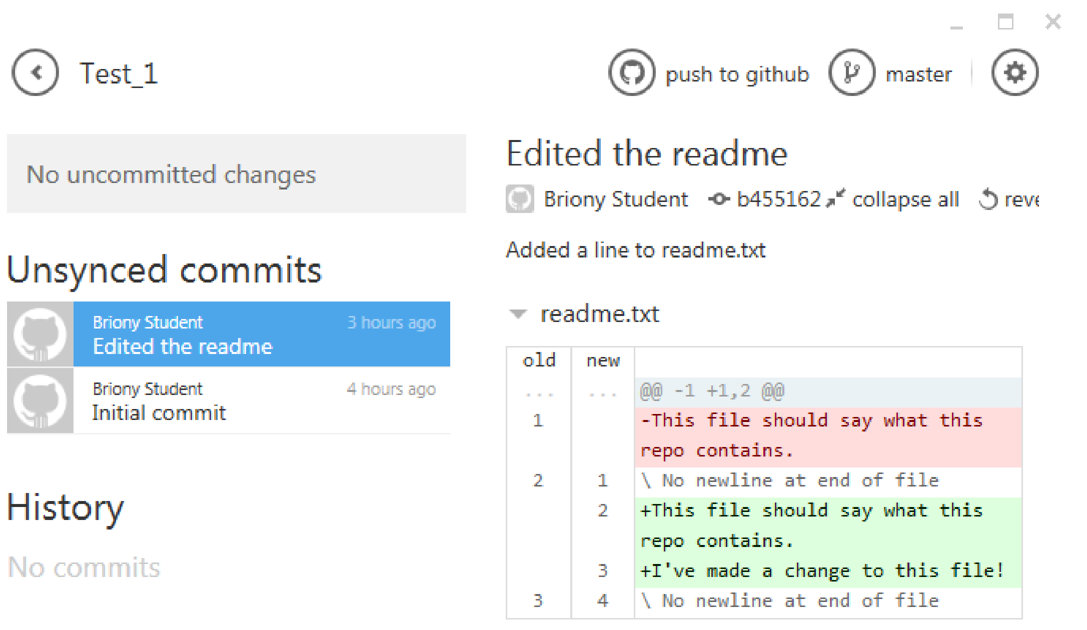


briony.student@gmail.com
••••••••

Sign up
Powerful collaboration, review, and code management for open source and private development projects.
Build software better, together.

First, log into **your own** github account through the application. If you have not already made an account you will need to do so now.

[1] You can stop the application from tracking the copy of Test_1 ("Test_1 – Copy") by right clicking on it's title and selecting ("Stop tracking repository"). This will leave the files intact, so you can also delete the "Test_1 – Copy" folder if you want too, leaving just the original "Test_1" files.


Test_1

push to github | master

No uncommitted changes

Unsynced commits

Briony Student — 3 hours ago — Edited the readme

Briony Student — 4 hours ago — Initial commit

History

No commits

Edited the readme

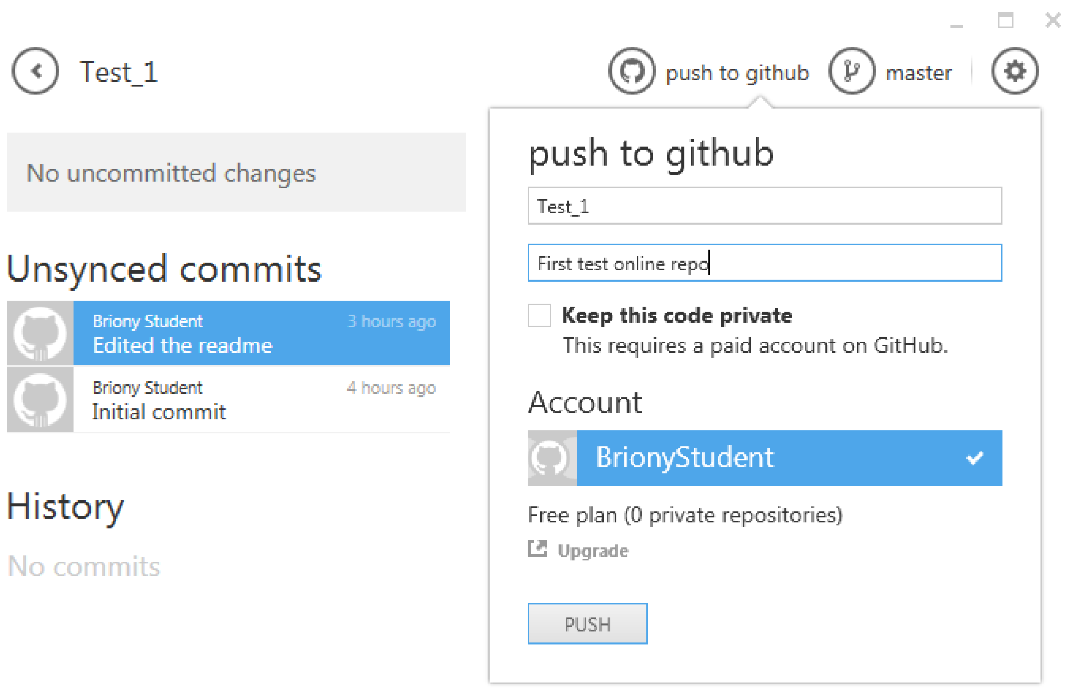
Briony Student b455162 collapse all reve

Added a line to readme.txt

readme.txt

```
old  new
...  ...  @@ -1 +1,2 @@
1         -This file should say what this
          repo contains.
2    1    \ No newline at end of file
     2    +This file should say what this
          repo contains.
     3    +I've made a change to this file!
3    4    \ No newline at end of file
```

Open the local Test_1 repo and the click “push to github”


Test_1

push to github | master

No uncommitted changes

Unsynced commits

Briony Student — 3 hours ago — Edited the readme

Briony Student — 4 hours ago — Initial commit

History

No commits

push to github

Test_1

First test online repo

Keep this code private

This requires a paid account on GitHub.

Account

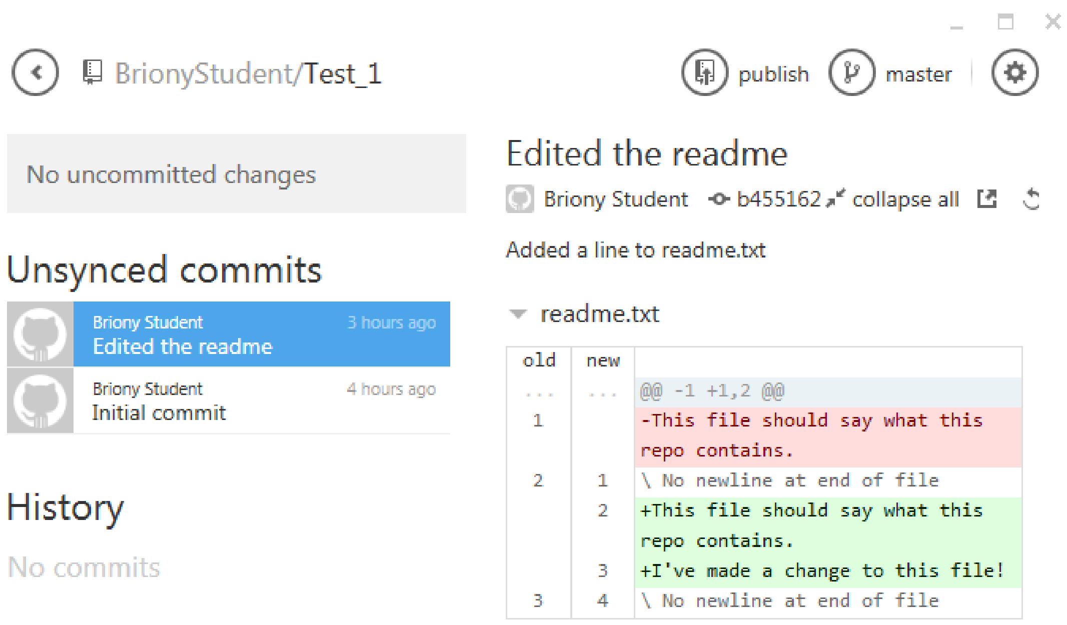
BrionyStudent ✓

Free plan (0 private repositories)

Upgrade

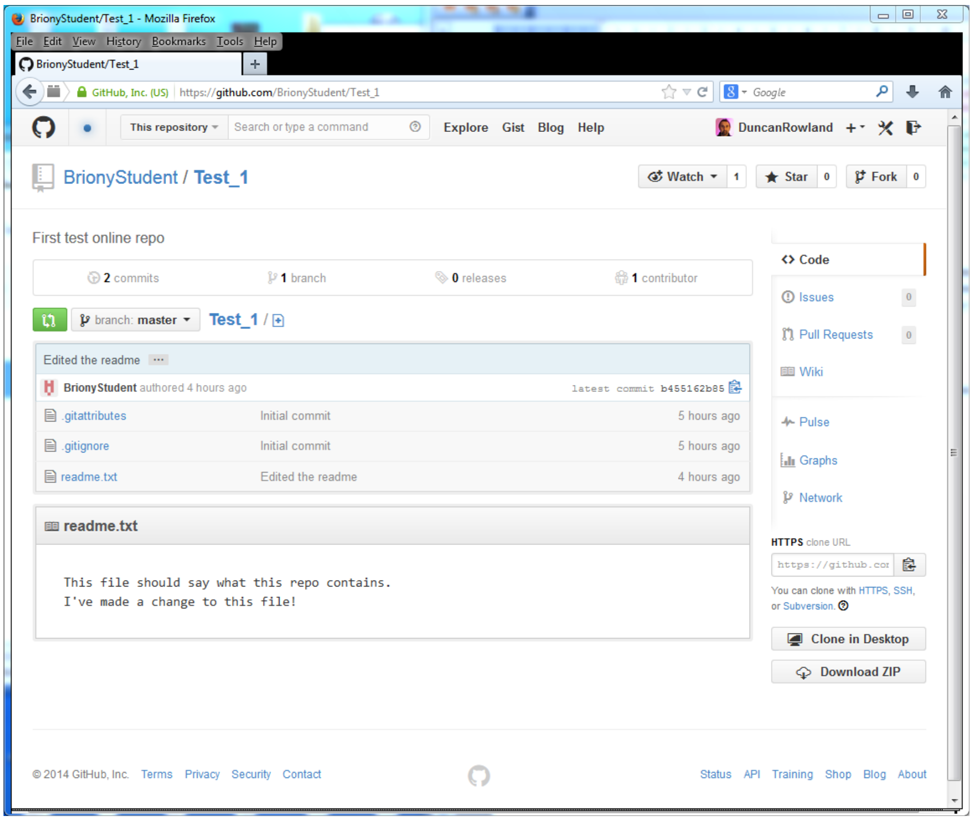
PUSH

Enter a description and the click “PUSH”. This will create the repo on your github account, but in order to add the actual files, there is one more step…



```
old  new
...  ...  @@ -1 +1,2 @@
1         -This file should say what this
          repo contains.
2    1    \ No newline at end of file
     2    +This file should say what this
          repo contains.
     3    +I've made a change to this file!
3    4    \ No newline at end of file
```

You need to "publish" the branches that you want to share. Some branches will just be for your own use, so there's no need to publicly upload those, though you may want to keep a copy on Git for your own convenience. N.B. you should clearly always **make a proper backup of your work somewhere else too, e.g. copies of the repo folder**).


BrionyStudent/Test_1 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
BrionyStudent/Test_1
GitHub, Inc. (US) https://github.com/BrionyStudent/Test_1
This repository Search or type a command Explore Gist Blog Help DuncanRowland
BrionyStudent / Test_1
Watch 1 Star 0 Fork 0
Code
Issues 0
Pull Requests 0
Wiki
Pulse
Graphs
Network
HTTPS clone URL
https://github.com
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP
© 2014 GitHub, Inc. Terms Privacy Security Contact
Status API Training Shop Blog About

First test online repo

2 commits 1 branch 0 releases 1 contributor

branch: master Test_1 /

Edited the readme
Briony Student authored 4 hours ago latest commit b455162b85

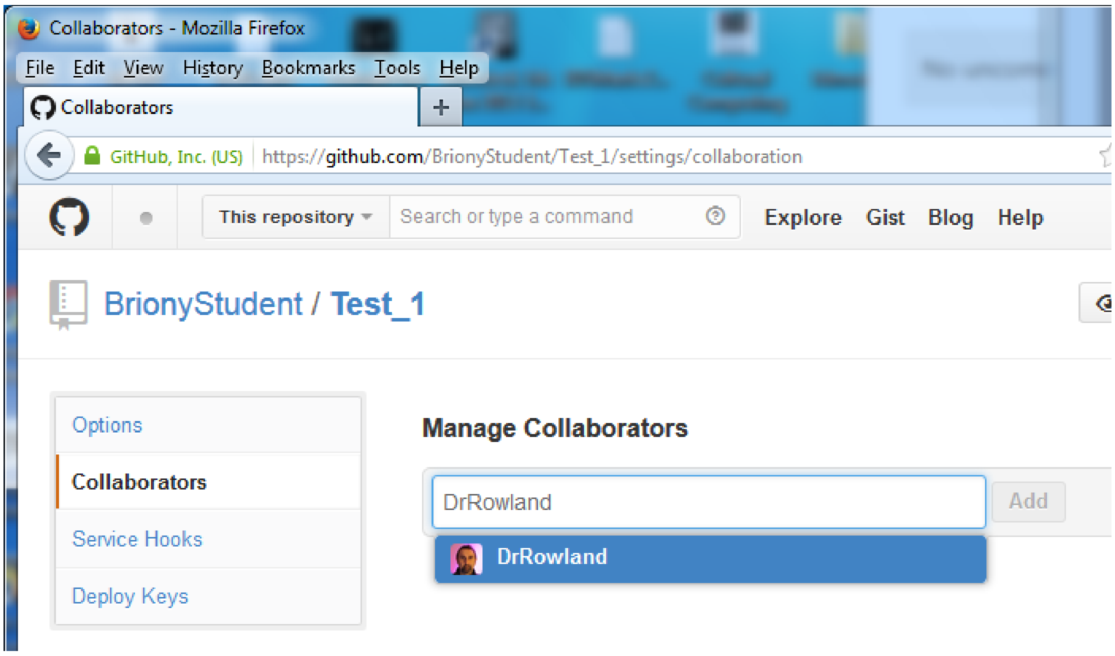
.gitattributes	Initial commit	5 hours ago
.gitignore	Initial commit	5 hours ago
readme.txt	Edited the readme	4 hours ago

readme.txt

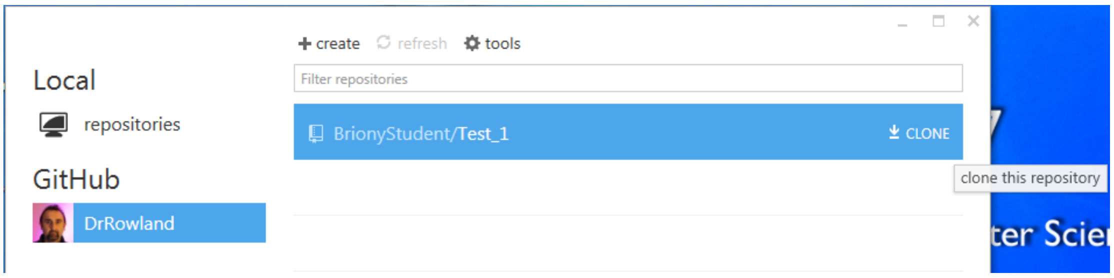
```
This file should say what this repo contains.
I've made a change to this file!
```

And that's it. Your repo is up on github and other people can access it!
Now, log into the github website and take a look at your repo there.

Although anyone can download your work, this does not mean they can edit it. You must add people as “Collaborators” to your project so they can uploaded changes to it. In the following example, Briony will add me (“DrRowland”) as a collaborator and I will edit her readme.txt and upload it. **Find a collaborator in the room and add them to your project (and have them add you to theirs) and follow along.**

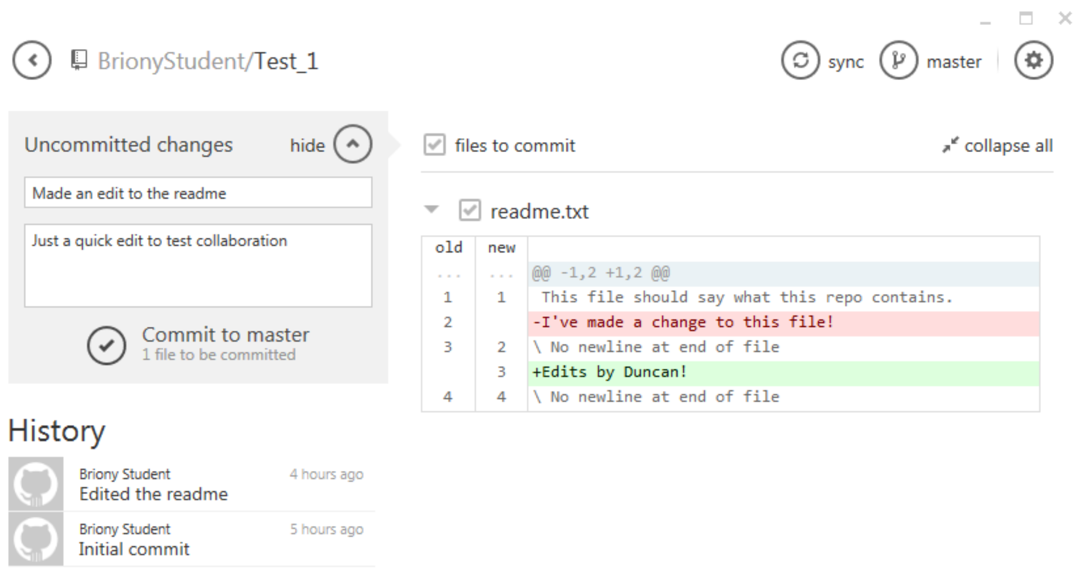


Open your own Test_1 repo on the github website and go to settings. On the left hand side click “Collaborators” and enter **your collaborator’s details** (not mine as above) in the box and click “Add”.



Back in github, it’s might be a good idea to go into “tools-Options” and change your “default storage directory” to something new (e.g. /BrionysRemoteGit). This is because in this tutorial BOTH of you will have a Test_1 repo so there is likely to be some confusion unless you keep them in separate folders.

Your collaborator should now clone your repository, and you should clone theirs.



BrionyStudent/Test_1

Made an edit to the readme

Just a quick edit to test collaboration

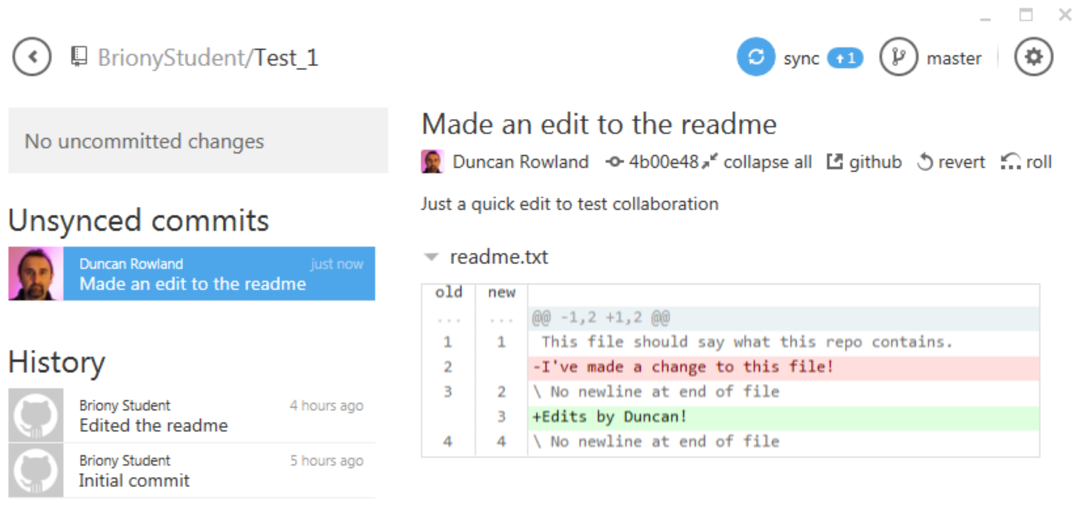
Briony Student 4 hours ago
Edited the readme

Briony Student 5 hours ago
Initial commit

readme.txt

```
old  new
...  ...  @@ -1,2 +1,2 @@
1    1     This file should say what this repo contains.
2         -I've made a change to this file!
3    2    \ No newline at end of file
     3    +Edits by Duncan!
4    4    \ No newline at end of file
```

Once you have cloned your collaborators Test_1 repo you can make changes and commit them just like we did with our own repository.



BrionyStudent/Test_1

Duncan Rowland just now
Made an edit to the readme

Briony Student 4 hours ago
Edited the readme

Briony Student 5 hours ago
Initial commit

Made an edit to the readme

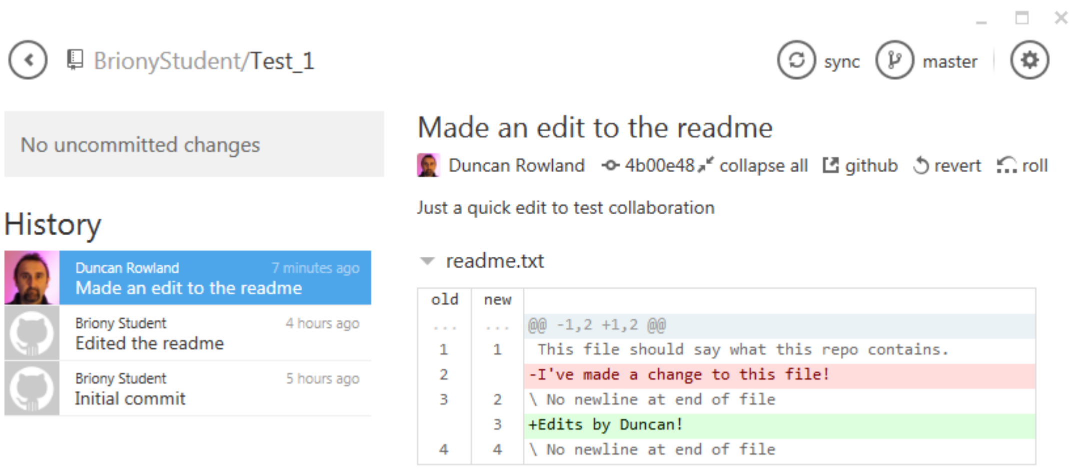
Duncan Rowland 4b00e48

Just a quick edit to test collaboration

readme.txt

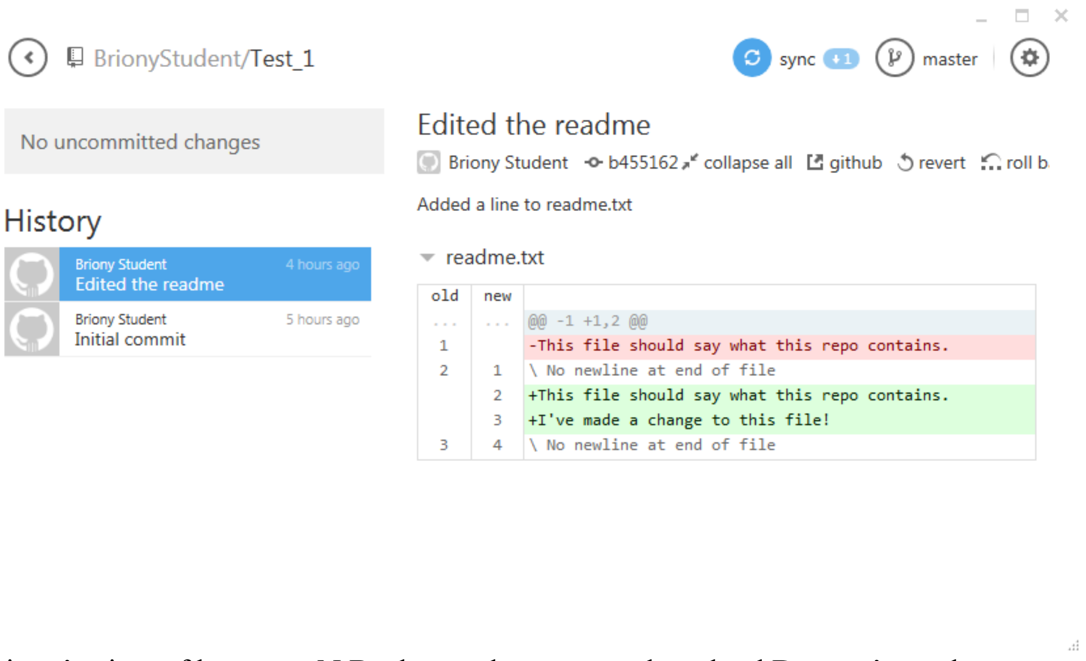
```
old  new
...  ...  @@ -1,2 +1,2 @@
1    1     This file should say what this repo contains.
2         -I've made a change to this file!
3    2    \ No newline at end of file
     3    +Edits by Duncan!
4    4    \ No newline at end of file
```

These will be stored locally until you sync the local changes with the server, at which point your changes will be uploaded. Make some changes to your collaborator’s readme file now and upload the changes as described.


BrionyStudent/Test_1
sync
master
No uncommitted changes
History
Duncan Rowland 7 minutes ago
Made an edit to the readme
Briony Student 4 hours ago
Edited the readme
Briony Student 5 hours ago
Initial commit
Made an edit to the readme
Duncan Rowland 4b00e48 collapse all github revert roll
Just a quick edit to test collaboration
readme.txt

```
old  new
...  ...  @@ -1,2 +1,2 @@
1    1     This file should say what this repo contains.
2         -I've made a change to this file!
3    2    \ No newline at end of file
     3    +Edits by Duncan!
4    4    \ No newline at end of file
```

Duncan’s view of Briony’s repo (with all his changes uploaded).


BrionyStudent/Test_1
sync +1
master
No uncommitted changes
History
Briony Student 4 hours ago
Edited the readme
Briony Student 5 hours ago
Initial commit
Edited the readme
Briony Student b455162 collapse all github revert roll b
Added a line to readme.txt
readme.txt

```
old  new
...  ...  @@ -1 +1,2 @@
1         -This file should say what this repo contains.
2    1    \ No newline at end of file
     2    +This file should say what this repo contains.
     3    +I've made a change to this file!
3    4    \ No newline at end of file
```

Briony’s view of her repo. N.B. she need to sync to download Duncan’s work.

It is important to keep your master branch synced so that you are not working with old code. You can always merge changes from the master branch into your local working branches (using a method similar way to that which merged your own changes into the master branch). N.B. When you’ve merged your changes with the master branch make sure it’s properly tested before you sync and uploaded it (otherwise you will break the master branch for everyone[2]).

[2]Note, there are other ways to collaborate via Git. You can 'fork' (create your own version of the repo on the server) and then request changes you make to be included in the main branch (issue a 'pull request'). This works well when one person looks after the repo but wants to allow non-collaborators to suggest changes to the project. Since we are working in small groups, where everyone knows each other, to keep things simpler, and uniform across teams, we will not be using this forking method.

Your task now, until the end of the workshop, is to try using Git with your collaborator until you are confident with it. This will be your last opportunity to become familiar with it before we use it in the project.

N.B. Excuses along the lines of "I did loads of work but then I tried checking it into git and I somehow managed to loose everything" are NOT ACCEPTABLE! You should ALWAYS BACKUP YOUR WORK. Git is a collaboration and versioning tool and it not an alternative method for creating backups.

Try the following:

1) **Choose a repo to work with (either yours or your collaborators)**
2) **Have one of you add another text file to it (i.e. commit and sync)**
3) **Make sure you're collaborator also syncs (so they receive the new file)**
4) **Now, both make some changes to the new file at the same time**
5) **Finally both try and check in your changes at the same time**

What happens? - how can you fix this?

5) **Try adding a new image file and doing the same thing (steps 3-5 again)**

What happens now? - and how can you fix this?

6) **Mouse over 'revert' and 'rollback'**

Can you see the different between the two? (it's not quite as obvious as it looks)

7) **Test using 'revert'**
8) **Test using 'rollback'**
9) **Add BrionyStudent as a collaborator to one of your repos**

Adding Briony as a collaborator is important, I will use this account to login and check your work. I will also use this to make a list of your github accounts.

10) **Continue to test git for the remainder of the session (finish 1pm)**

---

p.s. Last year there was some trouble using Unity with git (with people attempting to upload and download gigabytes of files that Unity created). In general, you should not be uploading files that get created for you (i.e. 'automatically' by your development system), rather you should be uploading just your actual 'work'. To cause this to happen you need to to do two things, tell git to ignore some files (i.e. not copy them to the repo) and to configure Unity so that it does not make others in the first place. Andy and Oliver have been looking into this and apparently the solution appears here:

http://www.dannygoodayle.com/2013/02/14/using-unity-with-git

If you have time at the end of the session and are confident with git and Unity I recommend creating a test Unity repo in preparation for the development work.