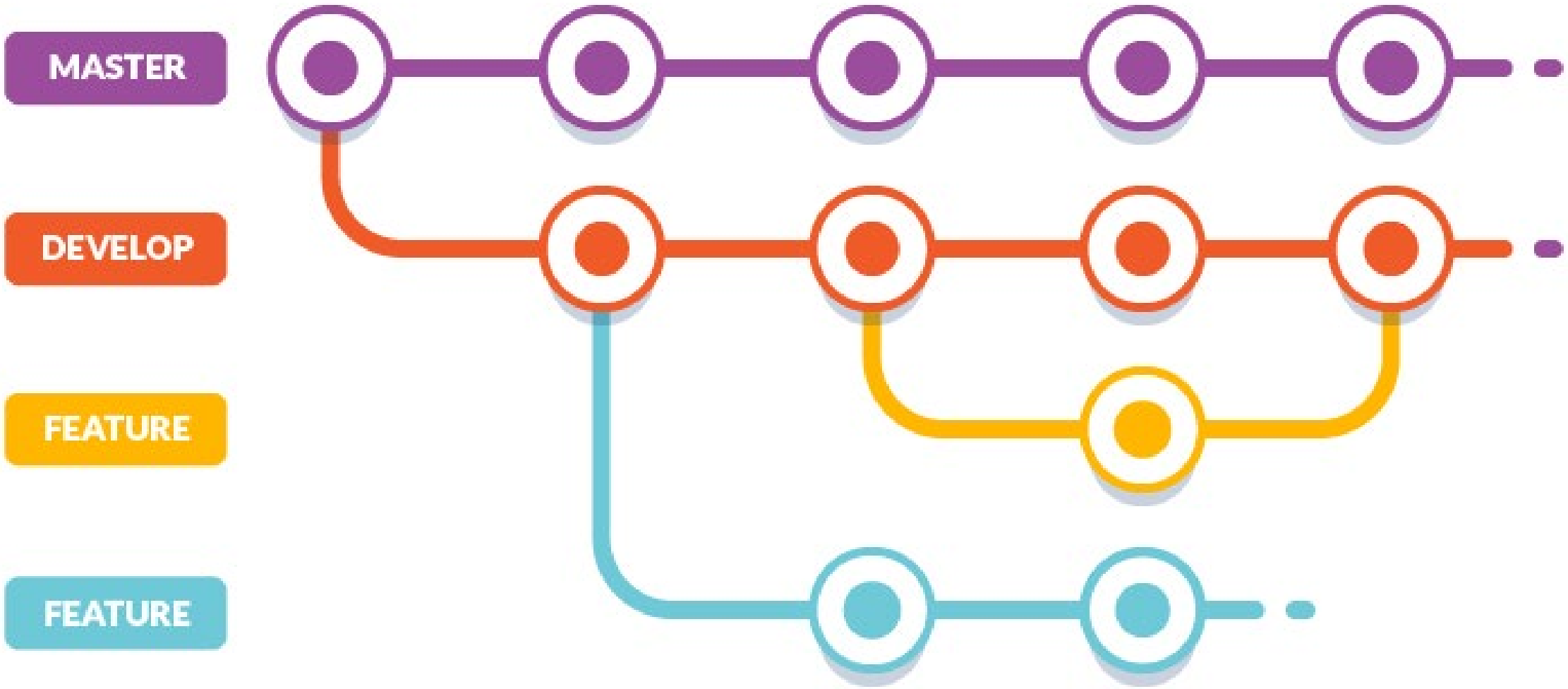
GIT

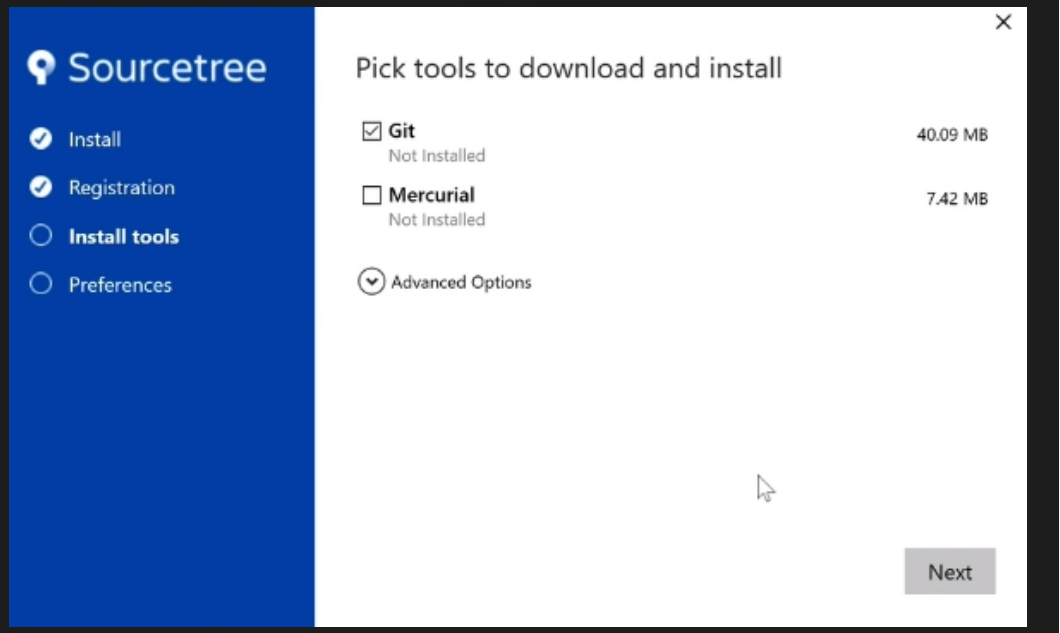


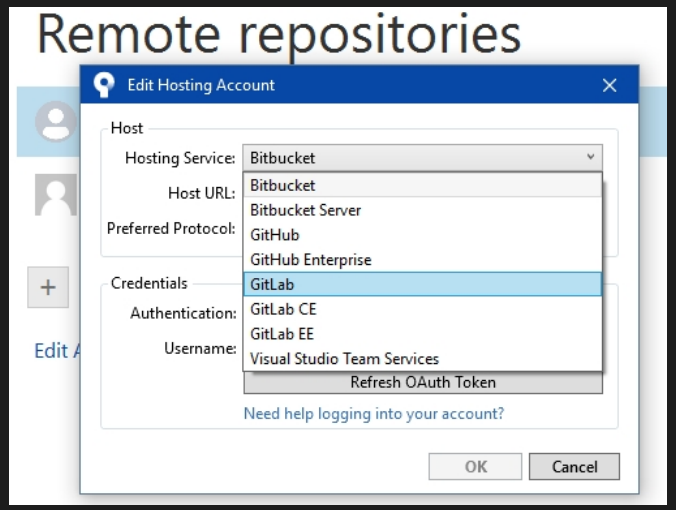
Intro

- Git is the VersionControlSoftware we're going to use. It was created by Linus Torvalds in 2005
- There are other systems out there Mercurial Perforce SVN, etc
  - It is a DB system that watches file changes on your system keeps track of them and stores all that in a big folder .git folder
- There are 3 major GUI client for Git: Sourcetree, GitHubDesktop, GitKraken
- You can put your content onto the internet to share with other people for offsite backup for collaboration:
  - Bitbucket (By Atlassian), GitHub, GitLab
- For the Setup, the best way is to follow a reverse order: Online repository / Local GUI Client / Git installation

Intro

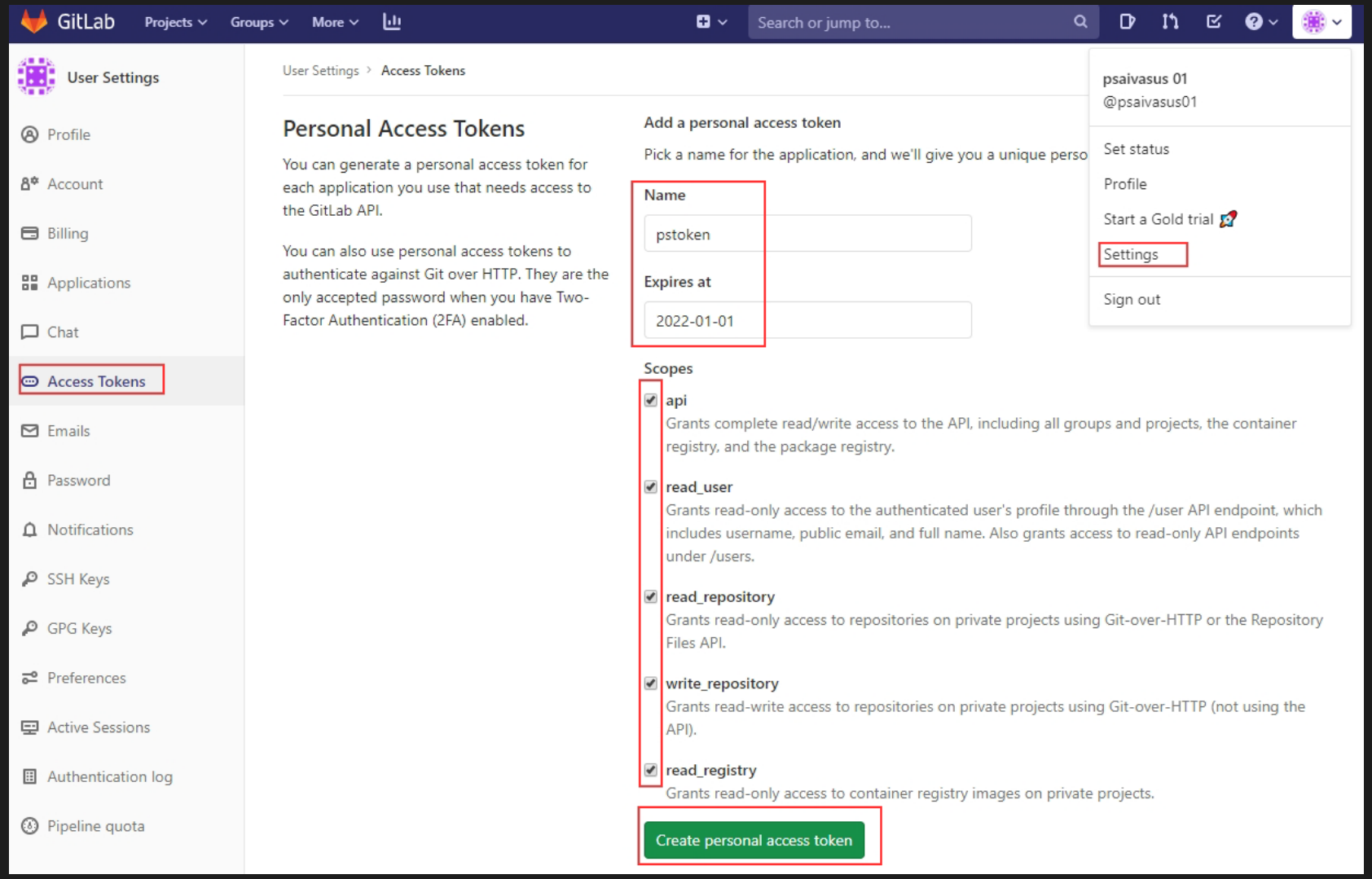
- Create an account on GitLab
- Install SourceTree
  - Registration – Bitbucket
  - Create an account
  - You'll be redirected to bitbucket account: we'll not use it, but we just need to get Atlassian free account for SourceTree
  - Go back to Sourcetree client
  - Install Git
  - Preferences/Set global author: use GitLab credential (username/password)
  - After: "Load SSHKey?" - choose No
  - Once Source tree open, Edit accounts/Authentication/Delete Bitbucket account
    - Remote/AddAccount/GitLab, Authentication PersonalAccessToken
    - We need to create a PAT on GitLab

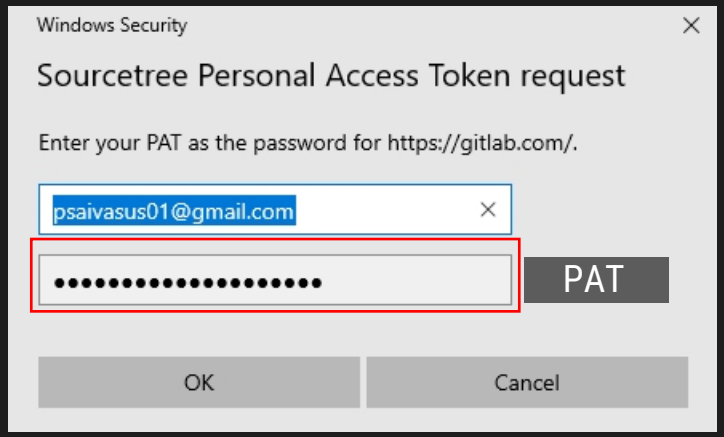


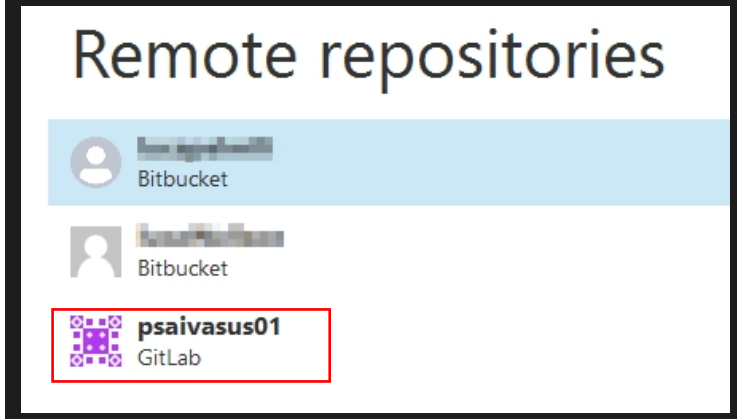


PAT Creation

- Create a Personal Access Token. Personal icon/Preferences





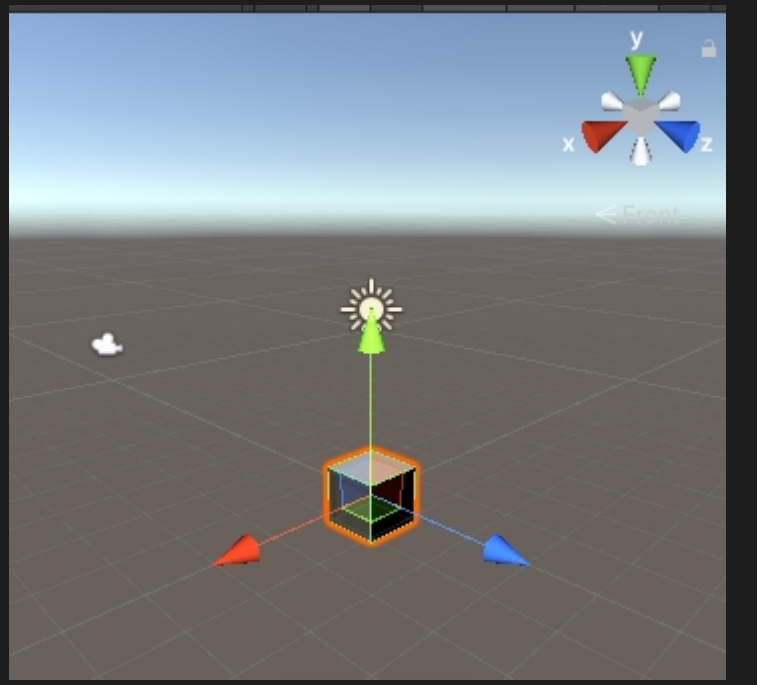


RoadMap

- Create a Unity project
- Setup our Project
- Setup our local repo (repository)
- Ignore the files that don’t matter
- Setup our remote repo on GitLab
- Push

Unity project

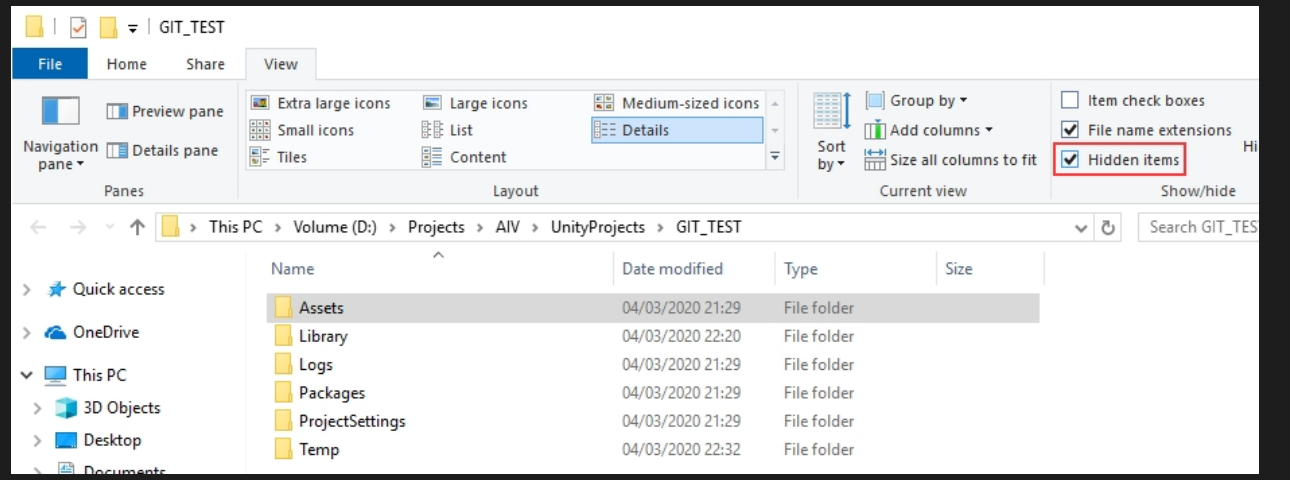
- Create a Unity project and a scene with a cube
- Locate it on the HD

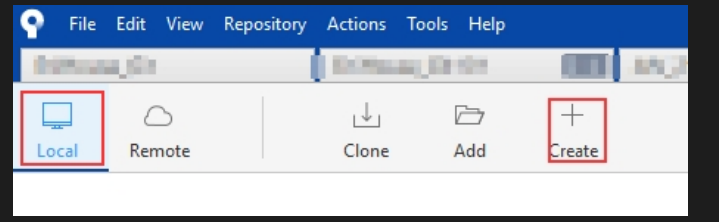


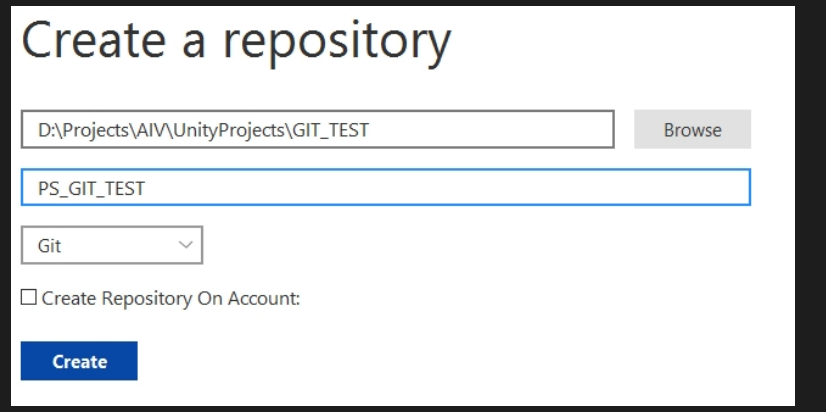
Setup Repo

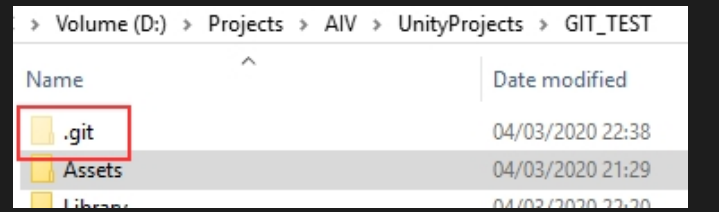
- Create a Repo means create a .git folder inside your project (dot because it's hidden – a Unix system convention)
- On the ProjectFolder, Show hidden files
- SourceTree
  - Create a new Local Repository
  - DestinationPath: Project Folder
  - Click on Create. You'll get a warning on existing path: continue and click YES

- On the ProjectFolder, there is a new folder: .git. Here SourceTree will store the differences of our project versions



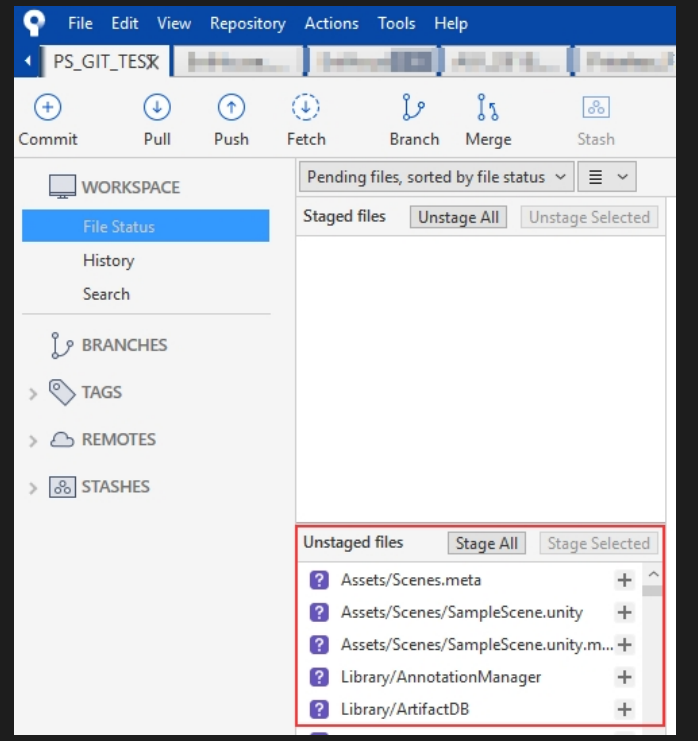


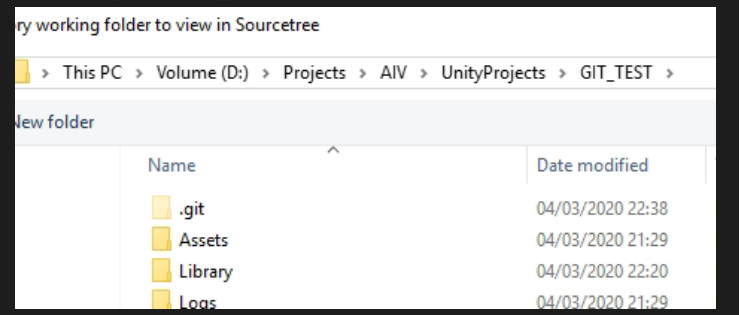


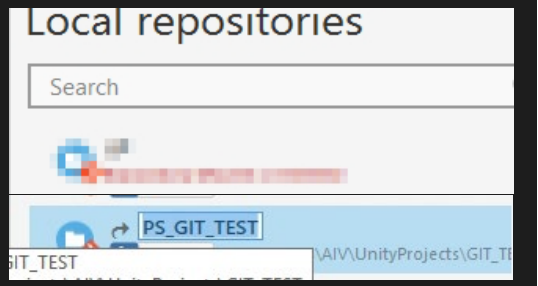


Setup Repo

- Inside Sourcetree you'll see A LOT of Unstaged files: we must ignore most of these files, we'll setup this later
- If, for some reason, your Sourcetree Project tab is not open (for example, if you relocate your project on another HD or another Path), you can open a Repo with File/Open, and pointing the Project folder (the folder with the hidden .git folder)
- You can Rename your Tab Sourcetree label in the View/ToggleBookmarksSideBar/Rename (Ctrl+B)

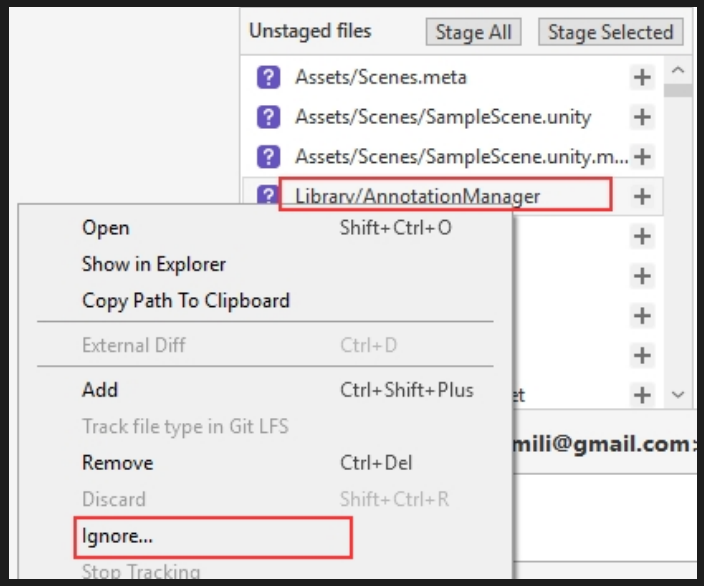


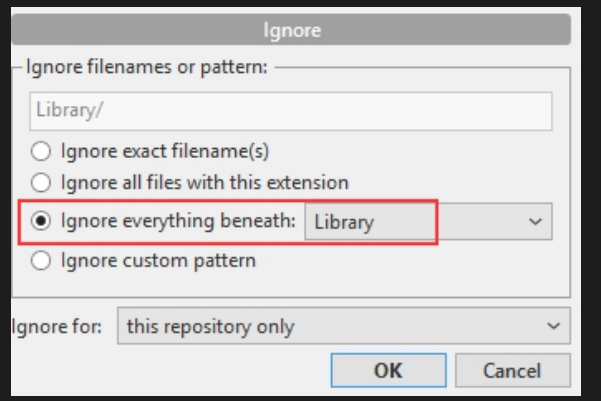


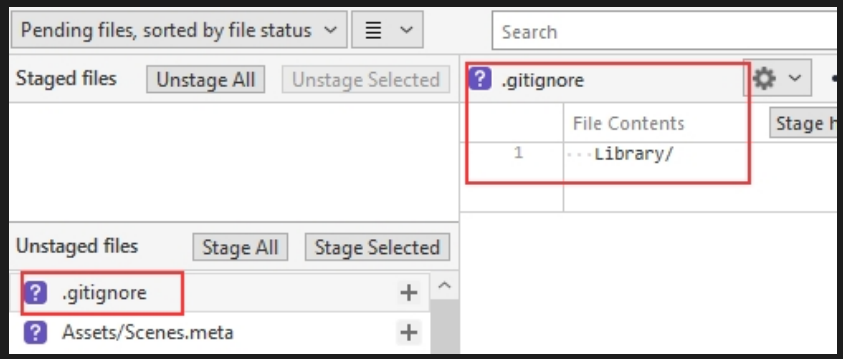


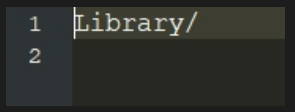
Ignore

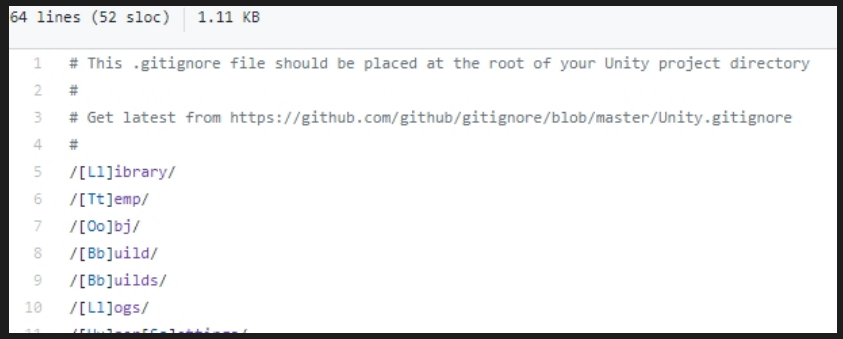
- We need to ignore the right files, so that we only store what matters
- Library folder is a cache, has derived information that can be recreated (if you delete it, it will be recreated next time you'll open the project)
- In SourceTree, select a Library file, right click, ignore
  - Ignore everything beneath: Library
  - Now a .gitignore file is created, with this information
- This is a good starting point for Unity gitignore: https://github.com/github/gitignore/blob/master/Unity.gitignore
- Copy its content and past into the existing .gitignore
- Now ST should show you only a few files to upload!
- On https://github.com/github/gitignore there are also other ready .gitignore files, for other software
- You should ignore /UserSettings and /Library folders, while /ProjectSettings is a folder that must be on the repo: it could be used to setup the Building pipeline for Continuous Delivery
- https://docs.unity3d.com/2020.1/Documentation/Manual/ExternalVersionControlSystemSupport.html











Ignore

ST

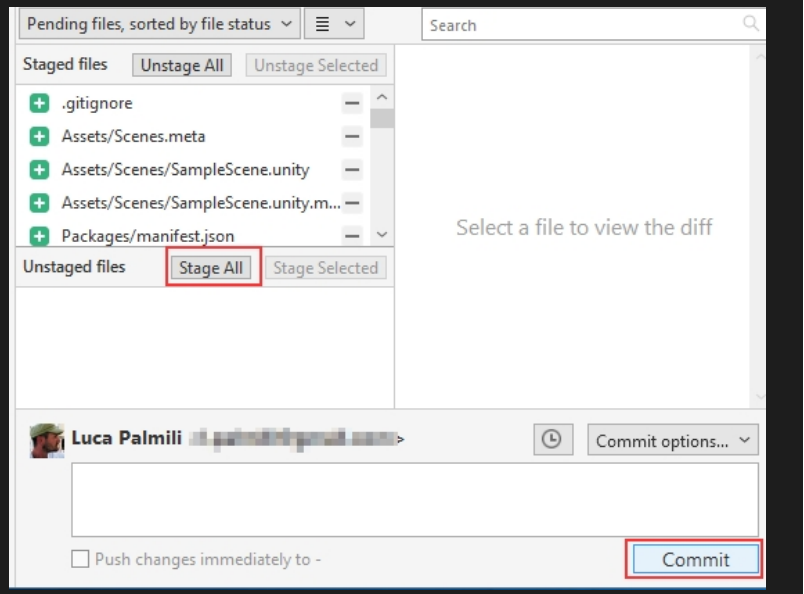
- Unstaged files are files that have changed from the last commit
- Click on StageAll
- Staging is the next stage towards committing: these files are ready to go, from now until I click commit, every other change will appear under UnstagedFiles
- Write something on Commit comment, and then click on Commit
- To see the commit history, click on History
  - For each commit, if it is a text difference, we'll see the changes

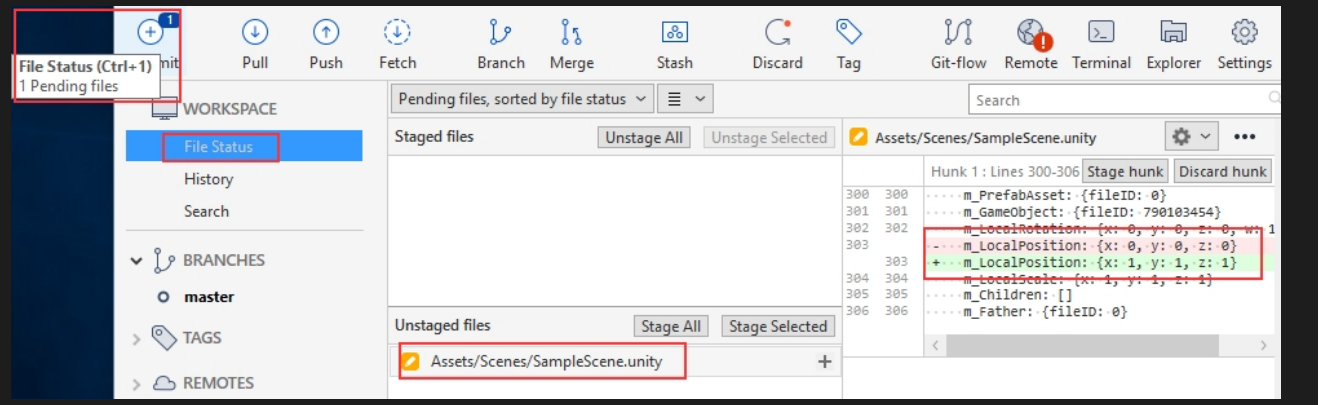
Unity

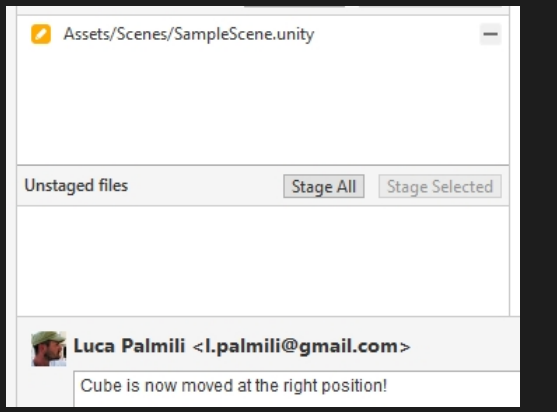
- Try to make a change on the Unity project
- Move your cube and save the scene

ST

- Back to SourceTree, you'll see in FileStatus that something is changed into you Scene file: LocalPosition!
  - RedLine: past version
  - GreenLine: new version
- Ensure that you saved the scene and the project in Unity
- Now we can perform the second commit!

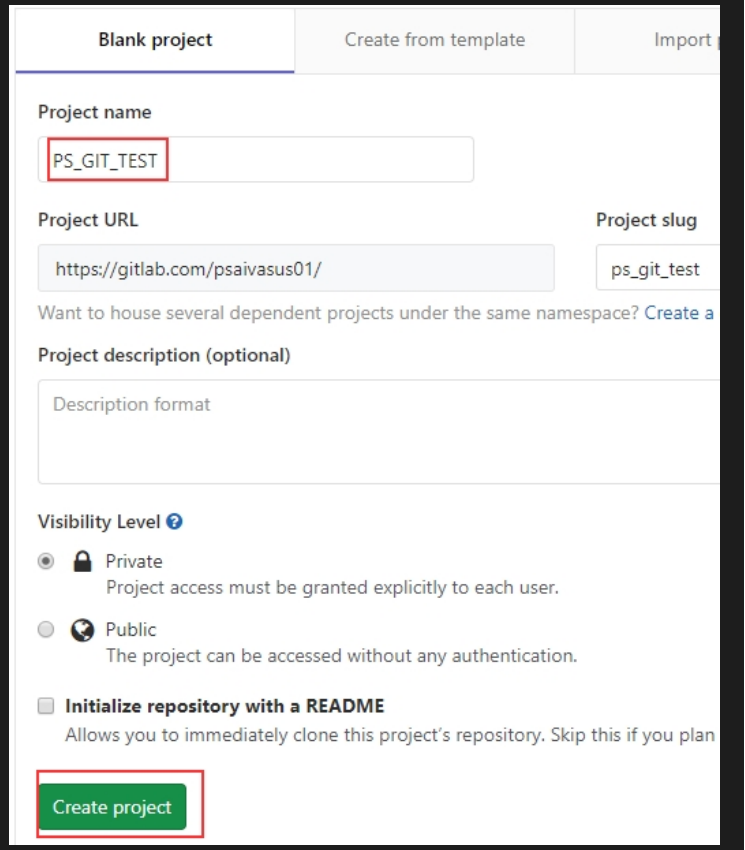


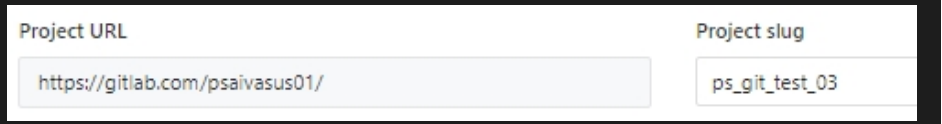


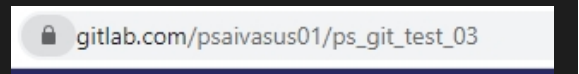


Remote

- At this point, we know that
  - Changes on Disc are Staged in Repo
  - Changes on Disc are Committed in Repo: this will encode changes in the .git folder
- Now we have to Push the information that we have locally on GitLab. This will allow us to:
  - Share with other people
  - Backup
  - Showcase our work
- On GitLab, click on the home icon on the upper left/Create a Project
- Choose a Project Name
  - NB: there are 3 names
    - Local project folder name
    - SourceTree Project Tab name
    - GitLab Project name
  - Project Slug is the end part of the project URL on gitlab domain
- Don't create a README file: since we already have a local project folder, this would create conflicts between remote and local project versions
- Click on CreateProject

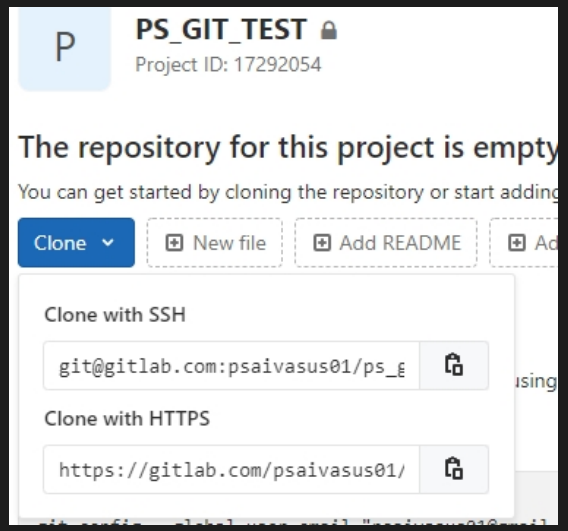


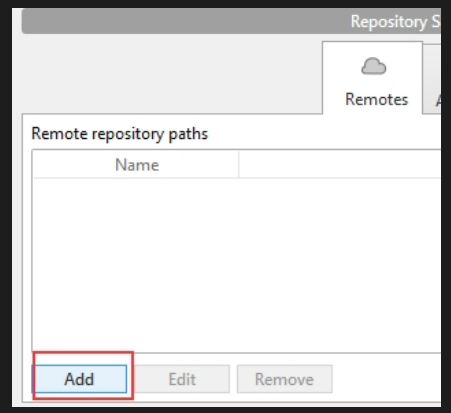


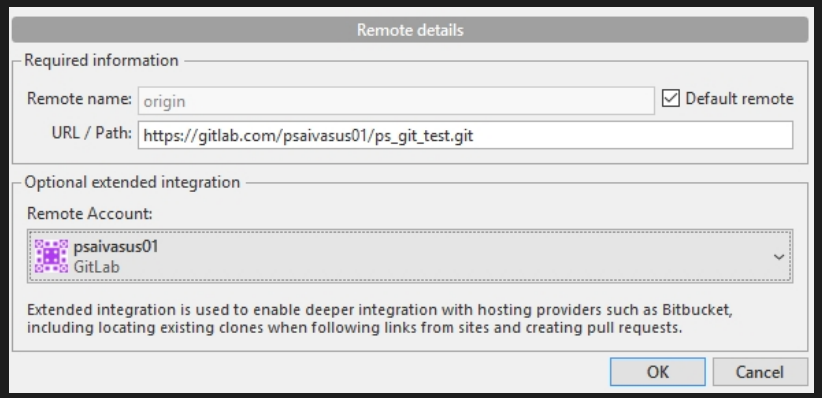


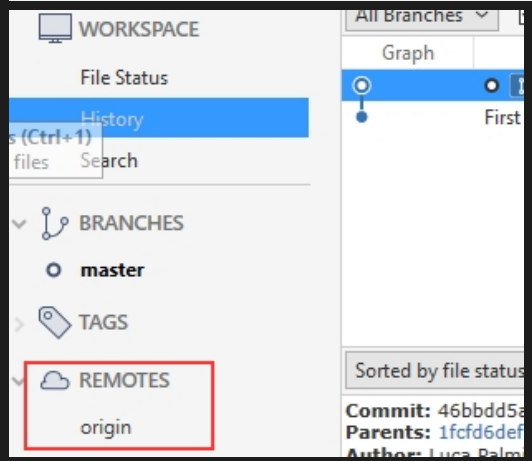
Remote

- We created an online Repo
- Under Clone, we have an HTTPS URL: copy it
- On SourceTree, go to Repository/AddRemote/Add
- On URL paste your Remote Clone HTTPS URL
- Choose your GitLab account, and then OK
- Now under Remotes we have our origin remote Repo!



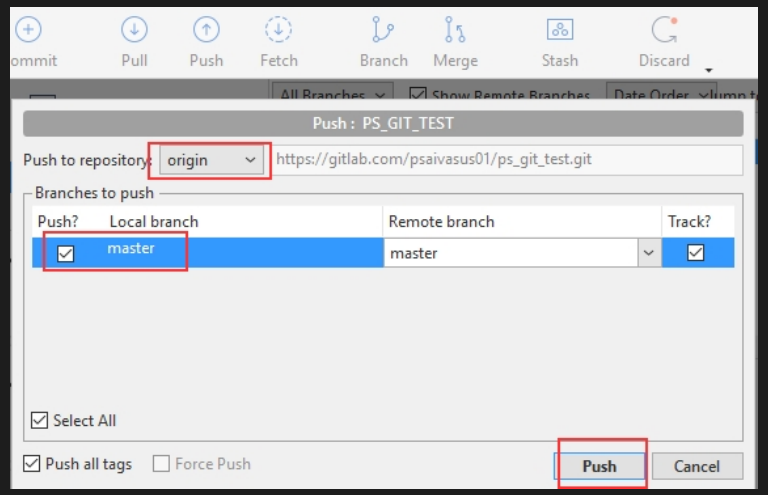


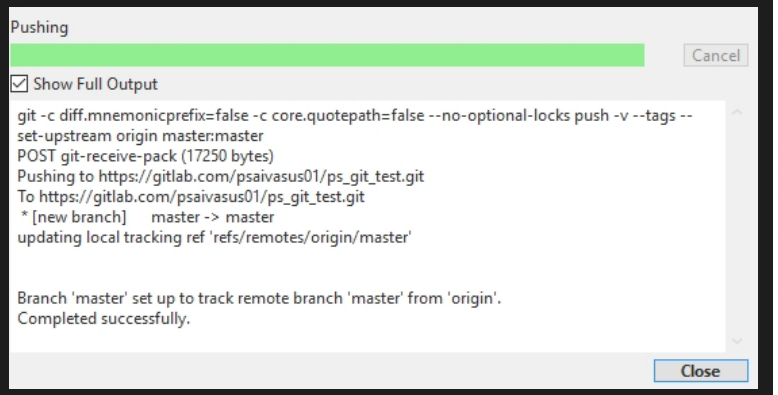


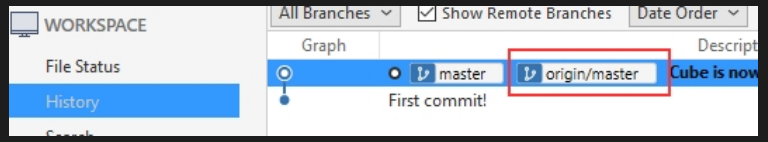


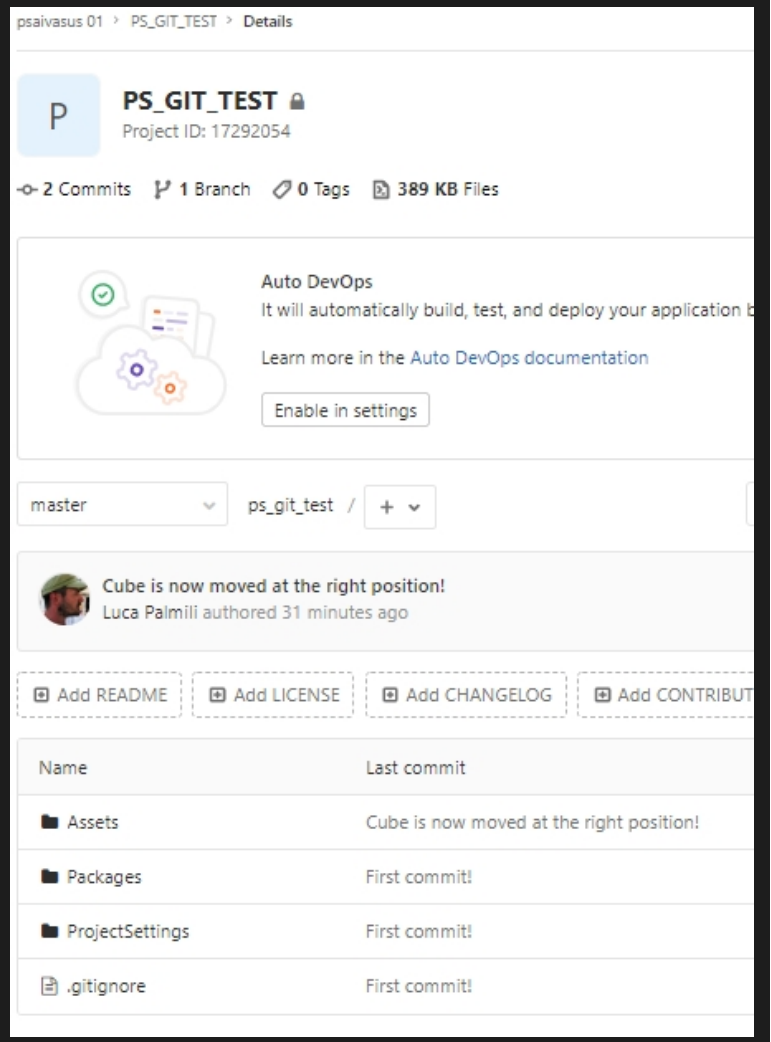
Push

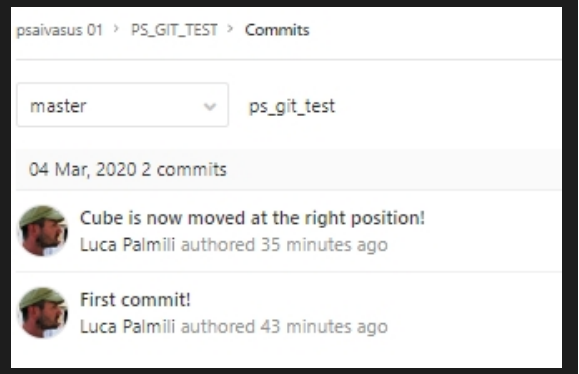
- We connected our Remote Repo to SourceTree, but we haven't pushed yet
- Click on Push button
- Tick master branch
- Once finished, you'll see that in History, an origin/master branch appeared: this basically means that this information is both stored locally and remotely
- On GitLab, go to your project details, and you'll see an online version of your project!
  - Note that there is no Library folder, along with some other missing folders!
- Online there is also the same history of commits you see in SourceTree, under Repository/Commits
- There are also a lot of other features
  - Issues
  - Pull/MergeRequest: when someone wants to contribute to your project
  - Wiki: sort of Wikipedia style project guide











Limitations

- GitHub has a 100MB limit on files
- Git System has no way to lock assets
  - If you’re on a team, if you want to lock an asset while you’re working on it, you can’t
  - There is a Unity plugin for GitHub that allows to do that

Stashing

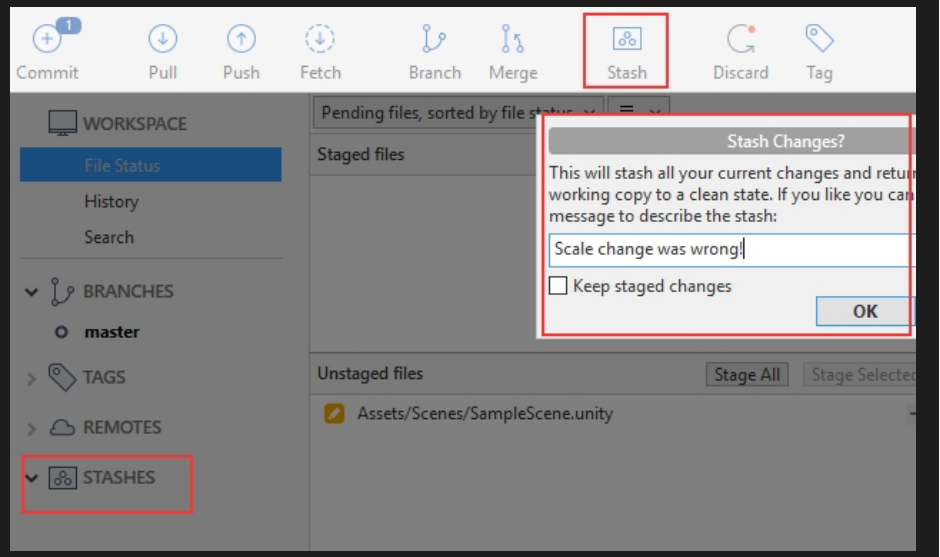
- Imagine that you are working on a part of a project and it starts getting messy. There has been an urgent bug that needs your immediate attention. It is time to save your changes and switch branches. The problem is, you don't want to do a commit of half-done work. The solution is git stash

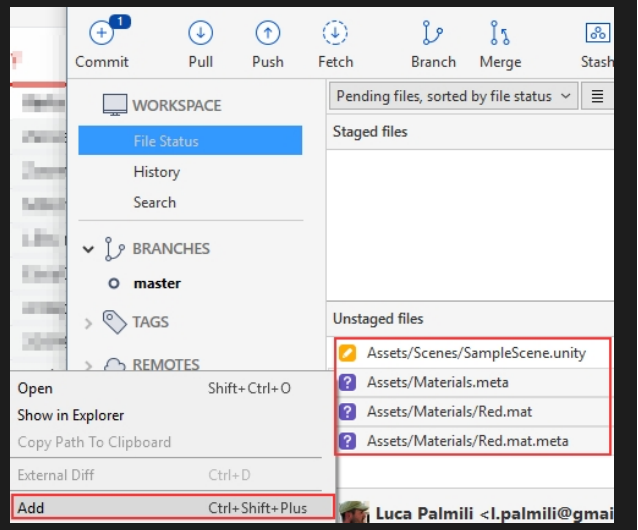
Try it

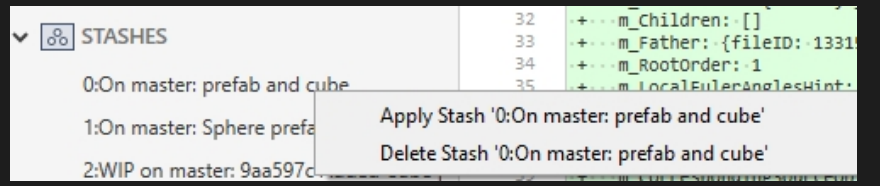
- In Unity, ensure that you have 2 scenes and one c# script file. If not, create them and commit
- Change both scenes
- There is a bug notification: you have to leave your current work, but you don't want to commit an half-done work

ST

- You have a notification: something is changed in the scenes
- Leave a message. Now you can choose to
  - KeepStagedChanges: the staged files will remain, and you can commit them
  - Not KeepStagedChanges
- History line has gone back to where we were
- Now we want to work on that bug: simulate it by making a change, for example changing the C# script. Well done, the bug is fixed!
- Commit it
- Now, we would like to come back to our previous half-done work
- Open Stashes on the left, and click Apply Stash
- Go back to Unity and reload the scenes: the changes saved into the stash are now applied again!

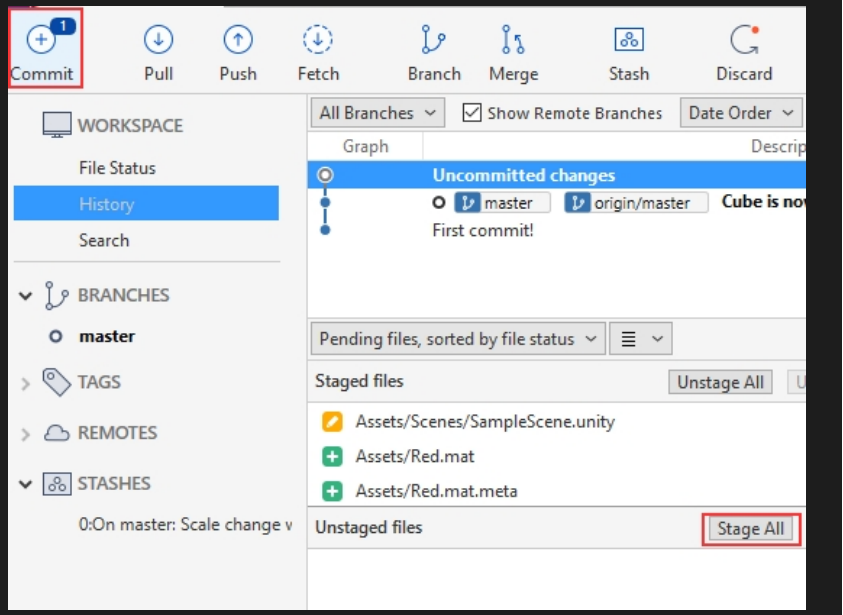


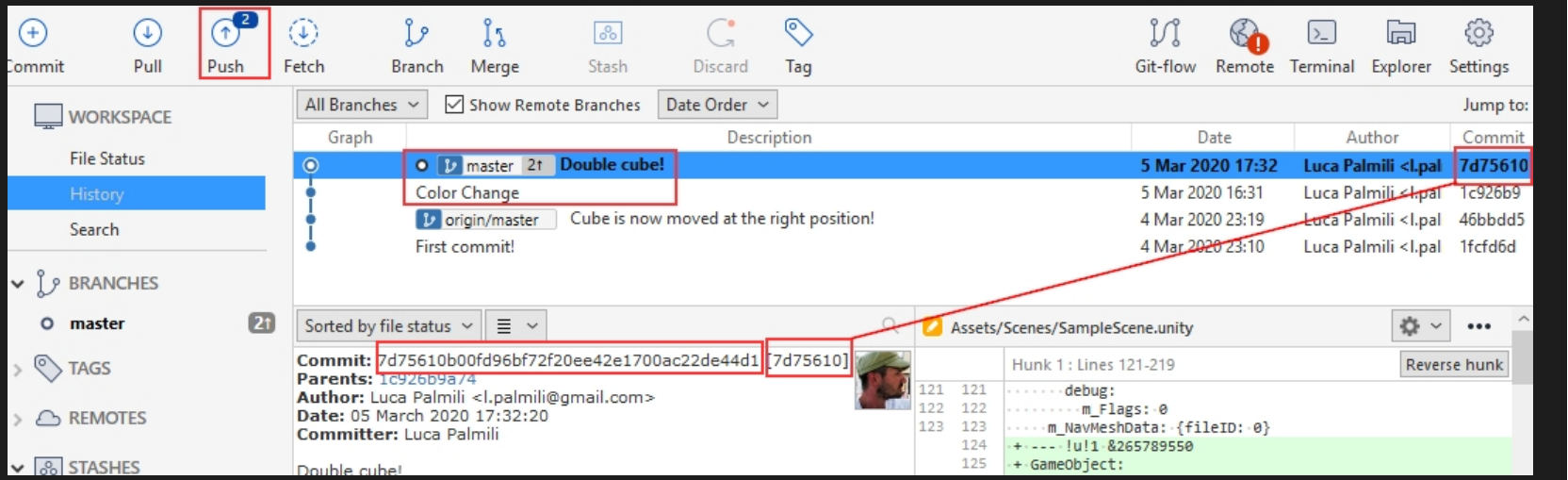




Commit

- In Unity, create a new Material color, assign it to the cube, and save
- In SourceTree, StageAll files and click Commit, leaving a Commit message
- Now Push button says that we have 1 local change that is not pushed on the remote server yet
- In Unity, duplicate the cube, and save
- In SourceTree, StageAll files and click Commit, leaving a Commit message
- Now Push button says that the local repo is 2 commits ahead of the remote one
- Each commit has a special cryptographic SHA (followed by a sum of these digits, tipically the first 7), that represent changes of that commit and the previous one: it creates a sort of chain
- Click Push: we are pushing the master branch (there is a label “master” at the beginning of the last commit) to the remote master branch





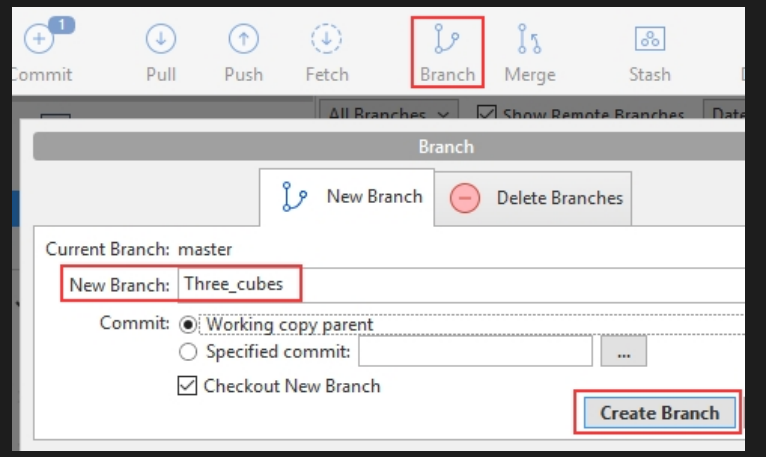
Branches

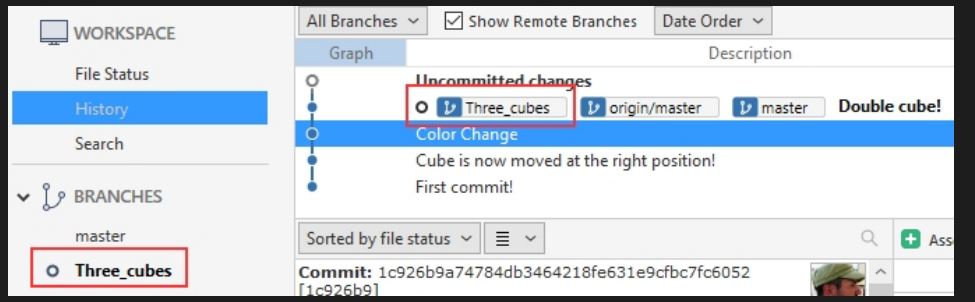
- A Branch is a pointer to a commit

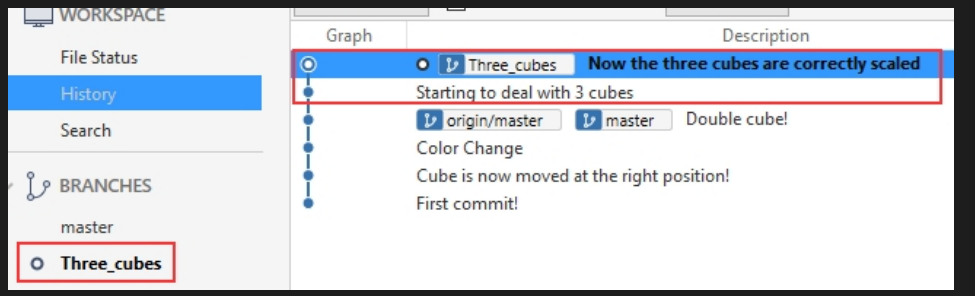
- Remove all the objects inside the scene, and create 3 cubes
- Let's pretend that I'm doing these changes as a prototype, something that I want to develop further while maintaining the original version untouched

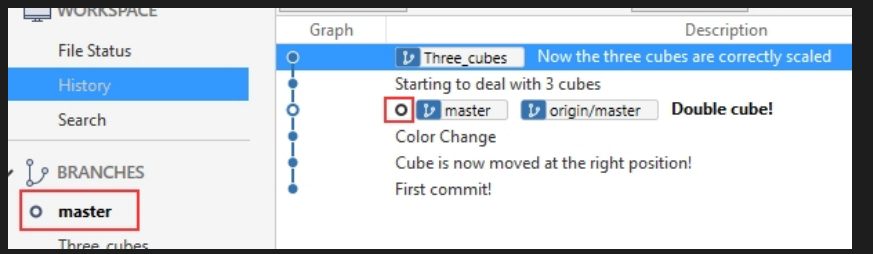
- In SourceTree, create a branch and name it Three_cubes
  - The History shows that there are 3 pointers to the same Commit "Double Cube!": master, origin/master, and Three_cubes
  - Under branches, there is a new Three_cubes branch
- Stage files and Commit changes

- In Unity, rotate the cubes and make them bigger
- In ST, Stage files and Commit changes
- The history says that Three_cubes branch has 2 commits after the master branch
- What if we don't want this Three_cubes prototype, and want to go back to the original version?
  - Double click on "master" branch
    - Branches says that we are now on the master branch
    - There is a little circle that says we go back
  - Go back to Unity: everything is restored!



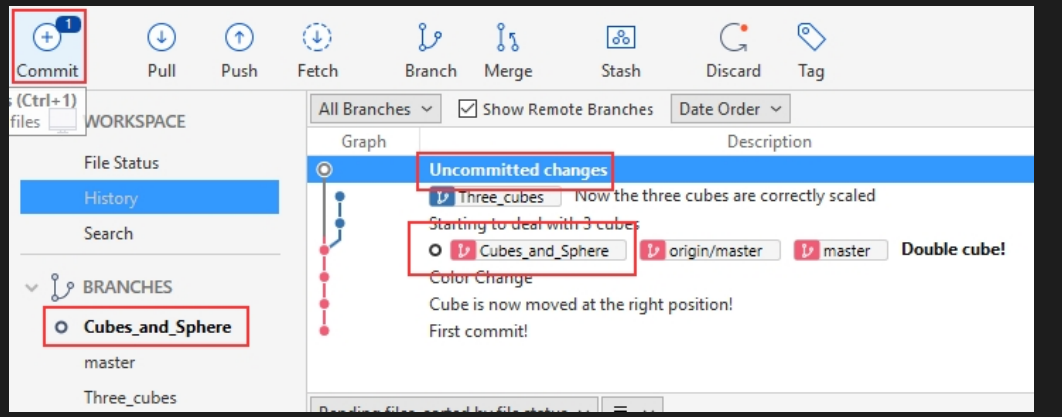


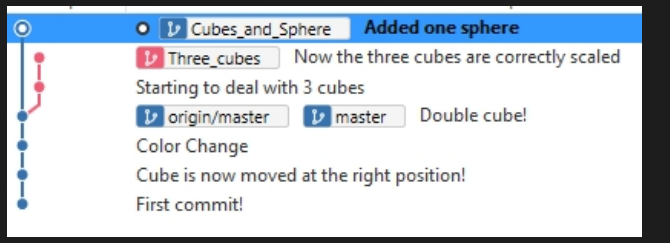


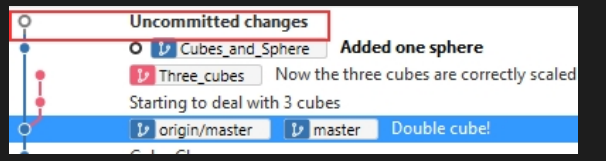


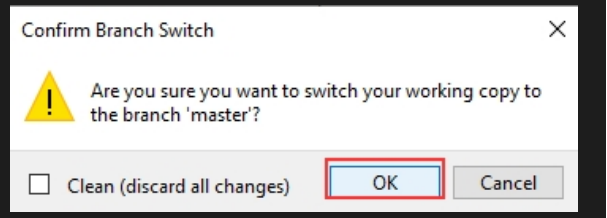
Branches

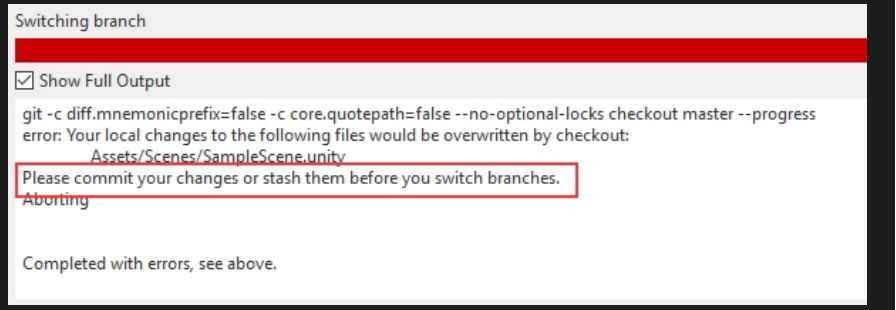
- Now that we are back to master branch, try to create another branch Cubes_and_Sphere
- In Unity, delete all the cubes and create 3 spheres
- In sourceTree, there is now a Cubes_and_Sphere branch that is pointing where everything started, and some uncommitted changes
- Commit these changes
- Go back to history: now we can clearly see that there is a Three_cubes branch that is stopped, and a Cubes_and_Sphere branch that has a new commit
- Now, we can (By click on the row in the history OR under Branches on the left column):
  - Go back to the last Three_and_cubes branch
  - Go back to the last master version
  - Restore the last Cubes_and_Sphere branch
- NB: we can switch branch only if there are no pending commits
- Try it:
  - Switch to Cubes_and_Sphere branch
  - Change Unity scene and save
  - In SourceTree we have a pending commit to perform
  - Try to switch to master branch
  - An error occurs: before to do that, you have to stash (or discard) your changes!
  - Discard (or Stash) your last pending commit
  - Now you can switch branch without problems!







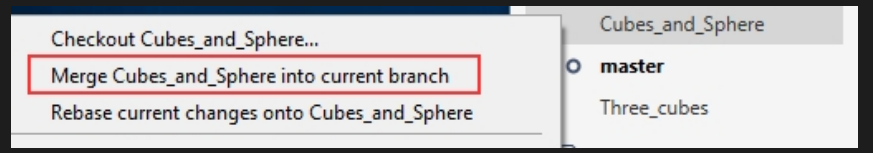


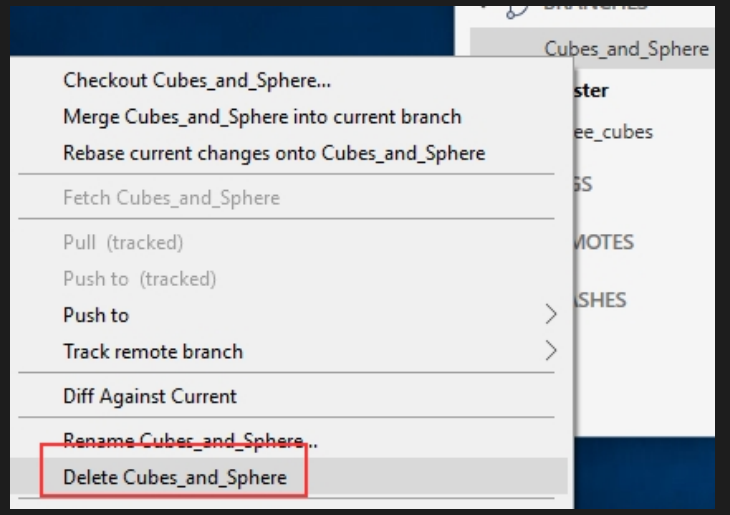


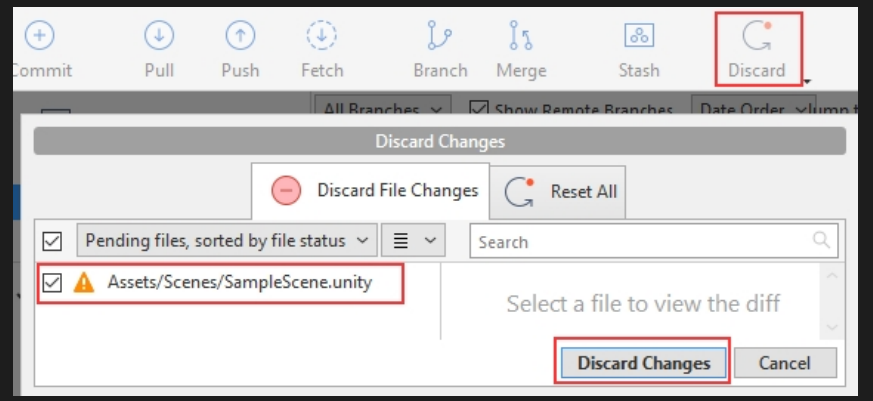
Merging

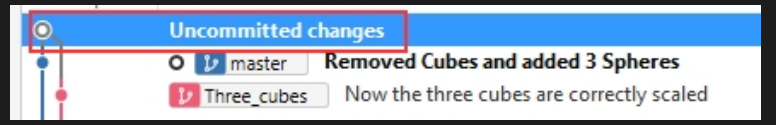
- After a Branch commit.. Are we going to use the stuff that you did on your branch or not?
  - Yes: Merge
  - No: Leave it hanging, or Delete it!
- Let's say that we want to keep Cubes_and_Sphere branch, and we want it to be part of our main project, our master branch
- We need to merge that branch in the master branch
  - Switch to master branch
  - Select Cubes_and_Sphere branch in the History
  - Click on "Merge Cubes_and_Sphere into the current branch"
- This merge was trivial, because Cubes_and_Sphere commit was a change that was created directly on top of master branch, so there were no conflicts during the merge
- Now we can Delete Cubes_and_Sphere branch: remember that Cubes_and_Sphere branch is a pointer, and now this commit already has a useful pointer on it, which is master

- What if we want to merge master and Three_cubes?
- Let's try that: select master branch and merge the Three_cubes into the master. Two things may happen:
  - Merge conflicts: both branches worked on the same scene, and when you asked to merge it, doesn't know what scene to take into account and how to merge them
    - For now let's discard this merge trial
      - Select the UncommittedChanges row in the history and click on Discard
  - No merge conflicts: this means that master scene and Three_cubes scene have the same game objects
  - if we don't have merge conflicts, this means that master scene and Three_cubes scene are easily merged by SourceTree: try another branch and use different primitives for the two brances (cubes for one, spheres for the other)

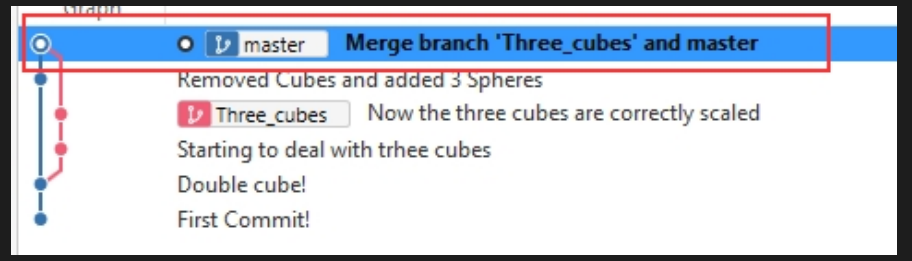






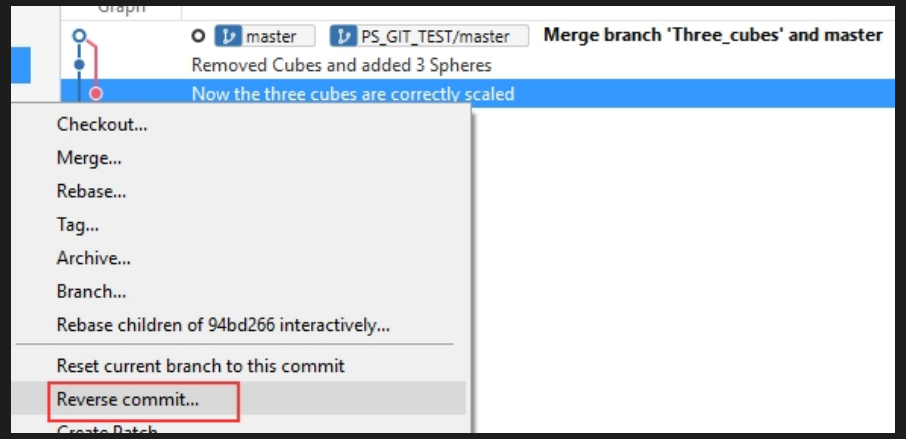


Merge Conflicts



- We saw that merge Three_cubes and master resulted in a merge conflict
- If we want to be able to continue to use Three_cubes prototype, we should avoid conflicts by duplicating that scene
    - Double click on Three_cubes branch to restore it
    - In Unity
        - duplicate the SampleScene: you'll create SampleScene1
        - Rename SampleScene1 to: Three_cubes. These will be a new scene with our changes that we want to merge with the master project, but we can't right now because these change were stored on the same SampleScene. These will store 1 changes inside SourceTree:
            - A scene creation (Three_cubes)
    - In SourceTree
        - commit these changes
        - switch to master branch and try to merge Three_cubes with it
        - There will be conflicts: select the uncommittedChanges row in the history
            - Look for the staged files that have an exclamation mark
            - Right click/Resolve conflicts
                - Mine: current branch – choose this
                - Theirs: the branch we're merging in
    - In Unity
        - SampleScene has the master version (with 3 spheres)
        - Three_cubes has the Three_cubes branch prototype version!
    - In SourceTree
        - We can delete also Three_cubes branch
        - Note that after you delete it, the History branch remains

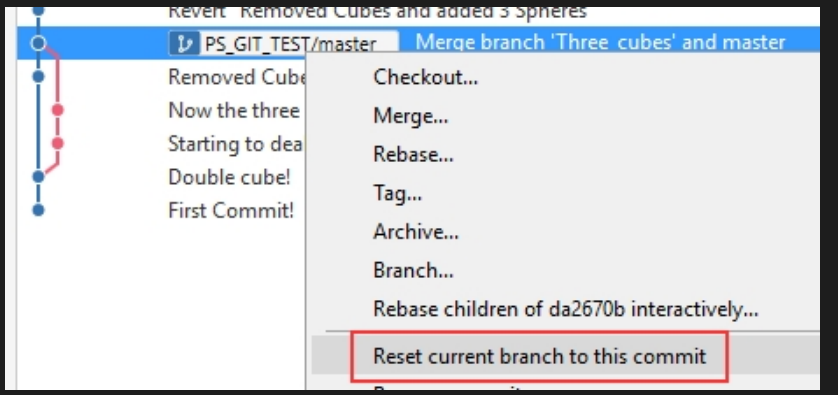
Revert / Reverse Commit



- What if we want to surgically remove one change that we did in the past
- You can even push the content to GitLab: these technique will play well even when you're already on GitLab
- We want to undo the commit: "Removed cubes and added three spheres"
- Select it in the History, and choose Reverse commit
- Now there is one new commit with the label "Revert"
- In Unity:
  - Open SampleScene: now there are 2 cubes instead of 3! We successfully removed ONLY that commit
- You can even Reverse the reverse: in ST, Click on the last History line (Revert…) and choose Reverse Commit
  - This is a non-destructive change
- In Unity:
  - Open SampleScene: now the 3 spheres are back! We successfully removed the previous reverse

Reset to this commit

- Let’s say that I don’t want the last 2 revert commits, and I want that my history stops just after the Merge
  - Select the “Merge branch” row in the history and choose “Reset current branch to this commit”
  - We have 3 options: soft/mixed/hard, choose hard
  - Now the other reverse commit are disappeared
    - This is a destructive change

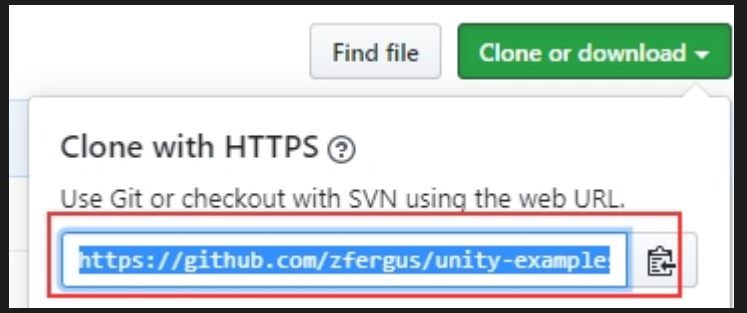


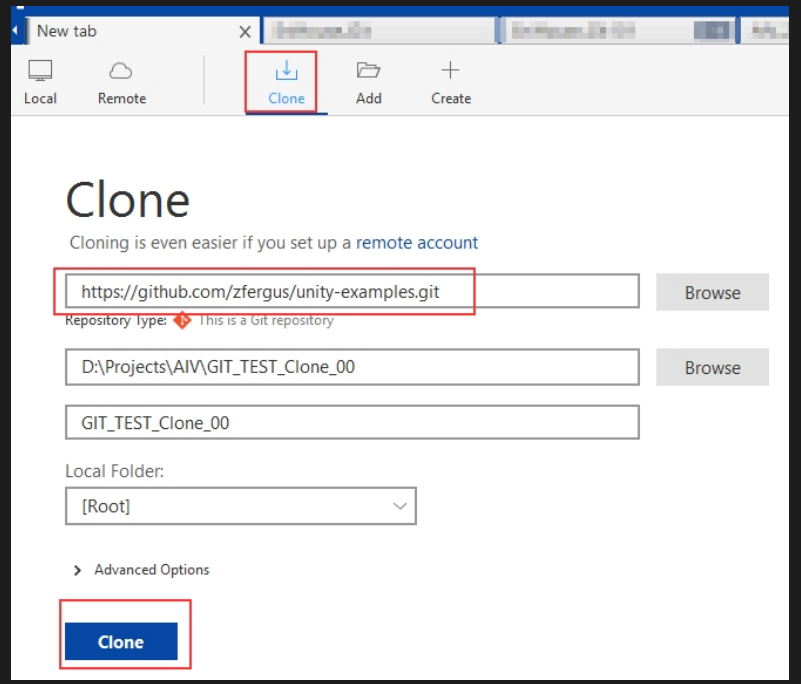
GitHub structure

- One repository has one or more branches, and one branch is associated to only one repository
- Repository can have multiple contributors, and the same contributor can contribute to multiple different repositories
- Each repository has a single person OR an organization that owns it

Cloning Remote Repos

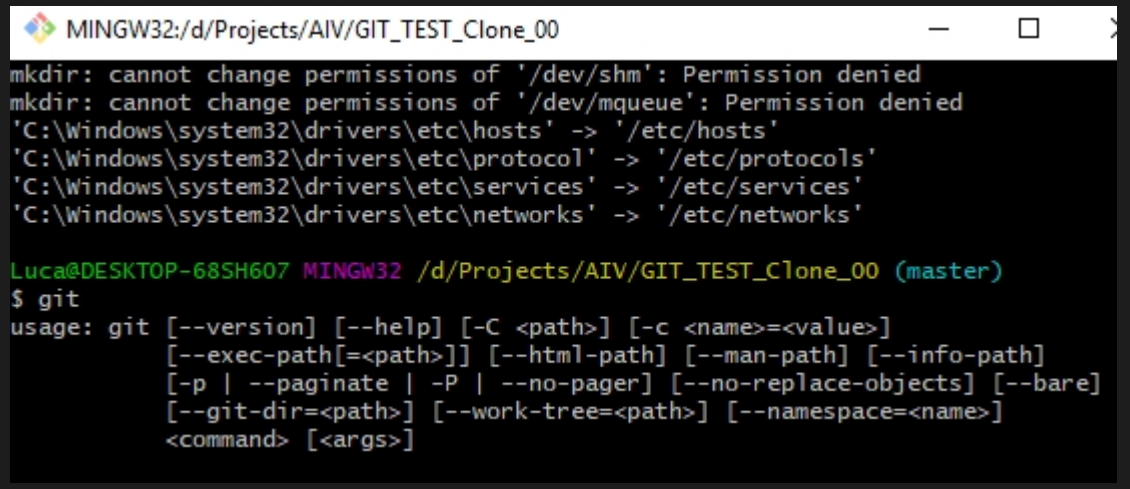
- Go to https://github.com/zfergus/unity-examples
- Clone/download > copy HTTPS link
- In ST
  - Add a Tab and choose Clone
  - Past the HTTPS link, and select Clone

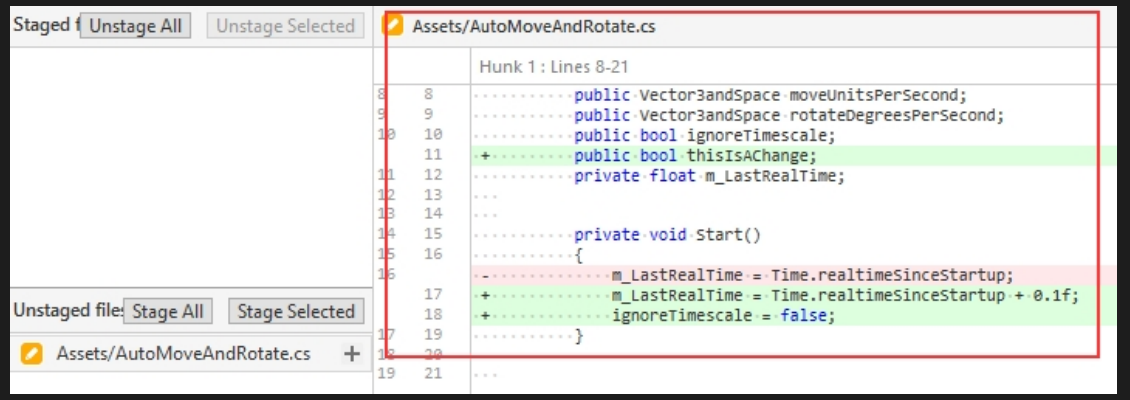


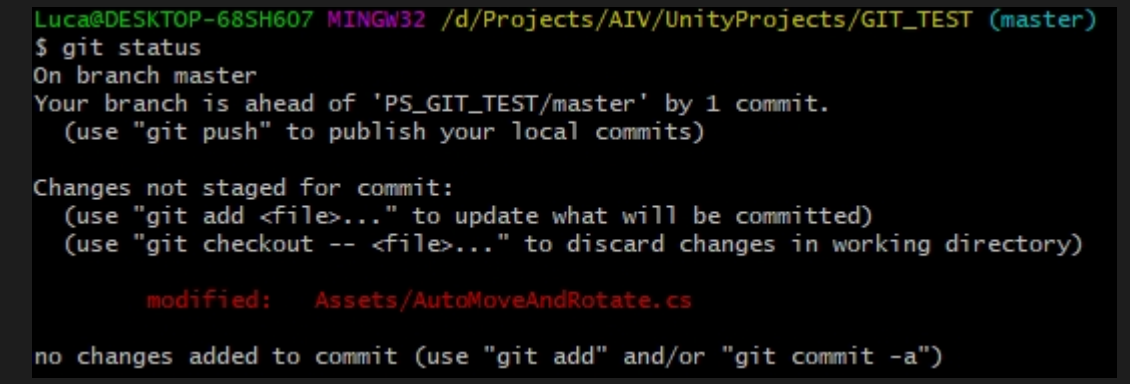


Terminal

- In ST, open the terminal. Type git and see the git commands
- In Unity
  - Add a cube in SampleScene
  - Add the AutoMoveAndRotate.cs script on the cube
  - Save
- In ST, commit
- In Unity, change AutoMoveAndRotate.cs script in some way
- In ST, take a look at the script changes
  - There is a Hunk, that is a “chunk” of code, from lines x to y
- In terminal
  - type git status
  - It tells you that
    - one script file has been modified, but not how
    - Use git add to move files from unstaged to staged
  - Type git add .
    - This will stage all files
- Back in ST, you’ll see that your script is in staged file group







Terminal commit

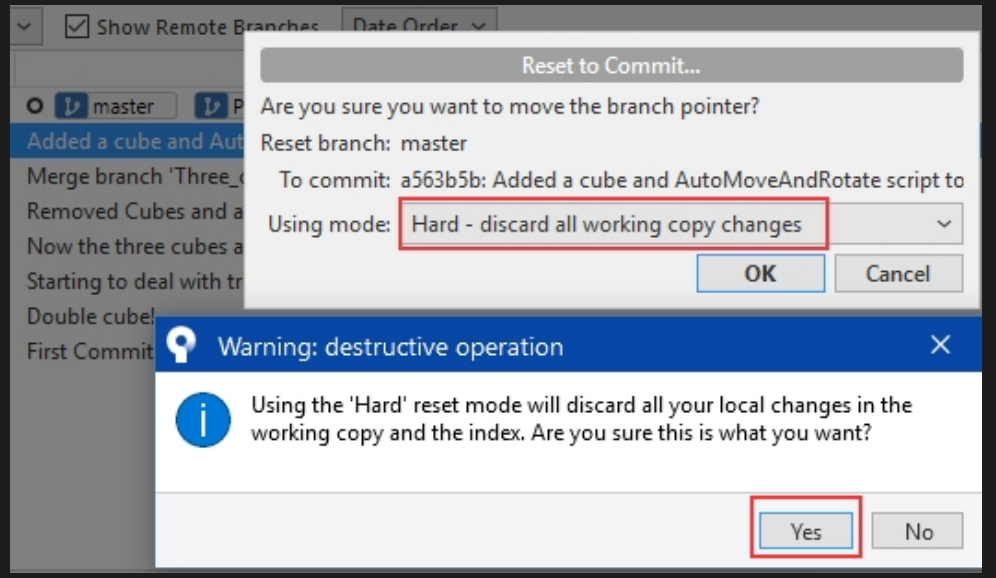
```
Luca@DESKTOP-68SH607 MINGW32 /d/Projects/AIV/UnityProjects/GIT_TEST (master)
$ git commit -m "Refactor"
[master e04d286] Refactor
 1 file changed, 3 insertions(+), 1 deletion(-)
```

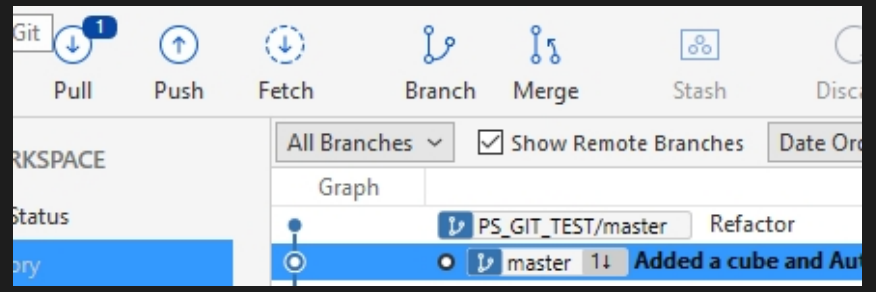
- In terminal
  - Type commit – help
  - Now we know that we have to write git commit <options>
    - Use –m to add a message
  - Type git commit –m “Refactor”
- In ST, there is a new commit done

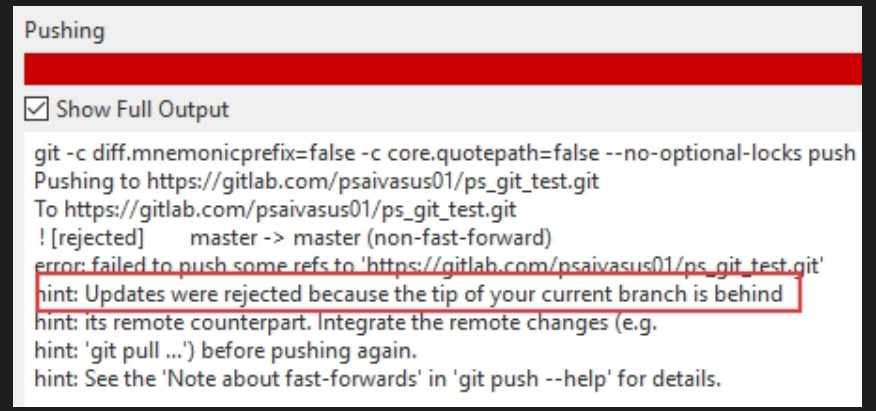
Forced push

- Imagine that you pushed your changes and you want to undo that push

In ST
- Push Refactor changes
- To view Repository online, Go to Repository/View repository online
- You can see the new commit also in the online repo
- Now we want to remove that push
  - Locally, if we want to do that, we simply select the previous commit and choose “Reset current branch to this commit”
  - Do it, and choose an Hard discard
- Now this is the situation
  - we don’t have our refactor commit locally
  - We are one line behind what is online: the “1” next to Pull button is encouraging us to pull, rather to push
  - If I try to push, this will result in an error because “the current branch is behind”
- If we want to update the online repository to be what we have locally in a forceful way, we need to do something called forced push







Forced push

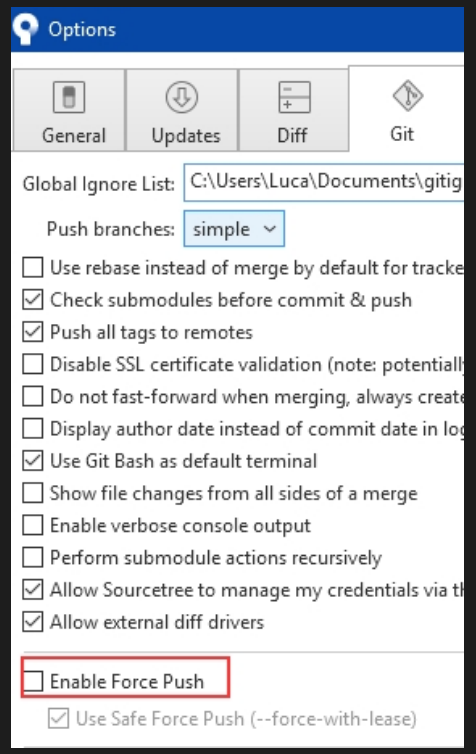
- If we try to push, force push checkbox is disabled
  - Go to Tools/Options/Git/EnableForcePush and enable it
  - Now try to push again: we still got an error, because we are in a “protected branch”
    - In fact, there is a lot of protection around Force pushing, because it is a destructive action

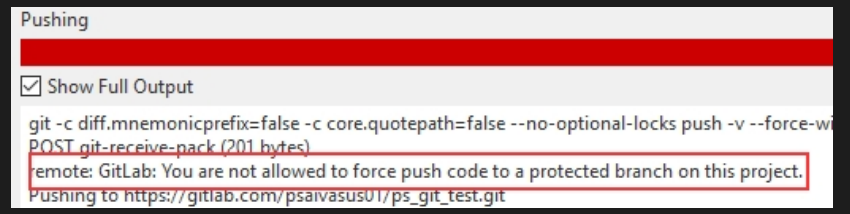
How to Unprotect that branch?

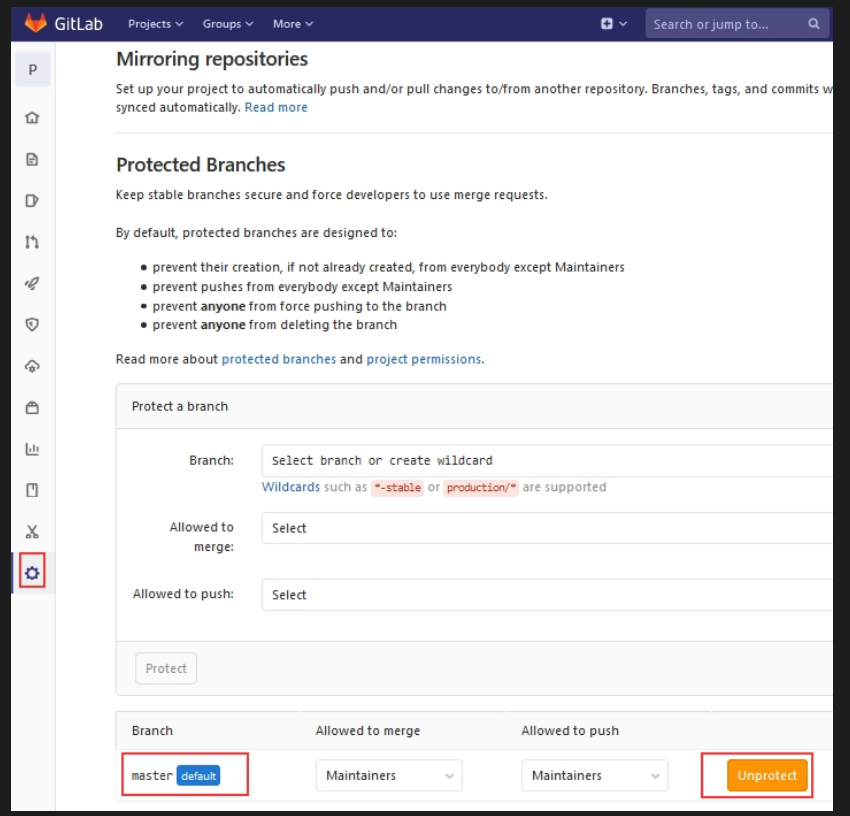
On GitLab
- Go to Settings/Repository/Branches
- Unprotect master branch

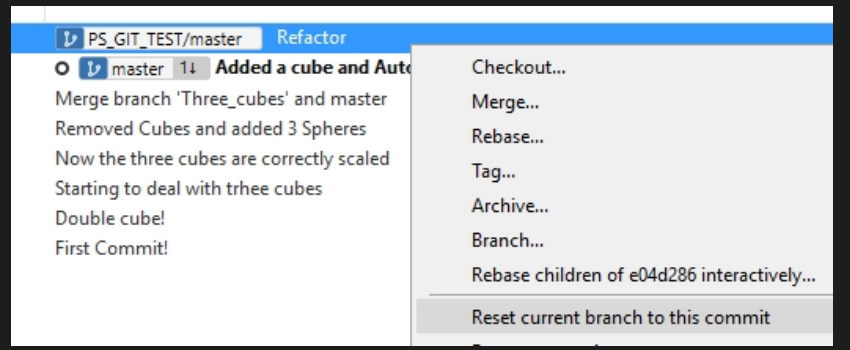
- Now you can Push via ST Client, or, on Terminal:
  - git push –f

- Or, you can restore the state using the online version, and Reset the current branch to the online commit









Diff stats for Refactors

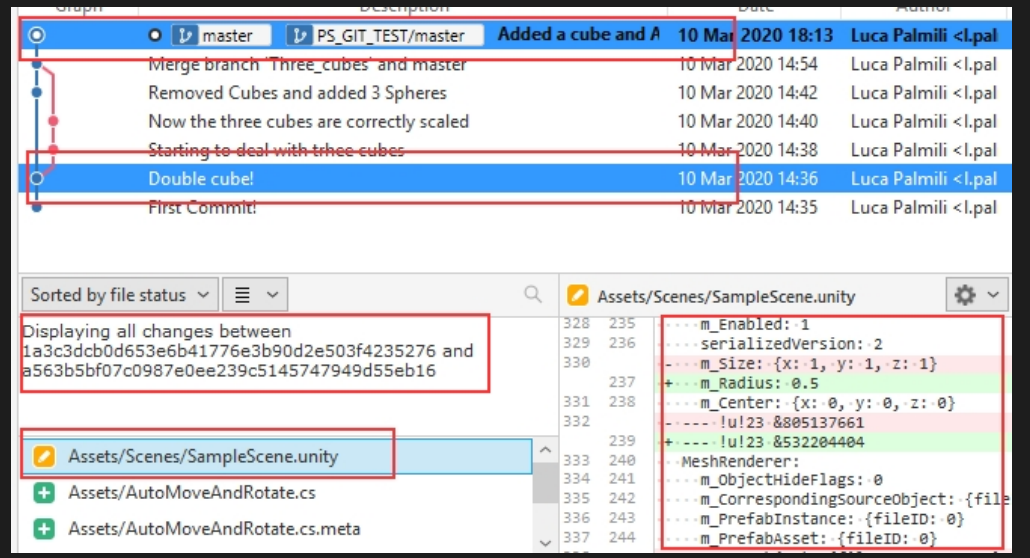
- This will help you to keep control of your refactors: when you change code hoping that it will become easier to read, easier to maintain, but you don't change its functionality

ST

- Click on one commit and ctrl-click another previous one: now SourceTree is displaying all changes between the two commits!

ST Terminal

- git diff
- If you've made no changes since the commit you've checked out, then this git diff will display nothing, no changes [1]
- Select an older commit and right-click/Copy SHA to Clipboard. Now use it in your terminal, e.g.
  - Git diff 1a3c3dcb0d653e6b41776e3b90d2e503f4235276
- You can scroll up down with arrows, q to quit. It is not better than our UI interface [2]. Then, let's try something that the Sourcetree UI can't show: statistics [3]
  - Git diff [SHA] --stat
- Now we can see the differences in terms of lines added/removed. If we want to focus on .cs files[4]:
  - Git diff [SHA] --stat .cs




```
Luca@DESKTOP-68SH607 MINGW32 /d/Projects/AIV/UnityProjects/GIT_TEST (master)
$ git diff

Luca@DESKTOP-68SH607 MINGW32 /d/Projects/AIV/UnityProjects/GIT_TEST (master)
$
```

1

```
Luca@DESKTOP-68SH607 MINGW32 /d/Projects/AIV/UnityProjects/GIT_TEST (master)
$ git diff 1a3c3dcb0d653e6b41776e3b90d2e503f4235276
diff --git a/Assets/AutoMoveAndRotate.cs b/Assets/AutoMoveAndRotate.cs
new file mode 100644
index 0000000..edbce0e
--- /dev/null
+++ b/Assets/AutoMoveAndRotate.cs
@@ -0,0 +1,41 @@
+using System;
+using UnityEngine;
```

2

```
Luca@DESKTOP-68SH607 MINGW32 /d/Projects/AIV/UnityProjects/GIT_TEST (master)
$ git diff 1a3c3dcb0d653e6b41776e3b90d2e503f4235276 --stat
 Assets/AutoMoveAndRotate.cs          |  41 +++
 Assets/AutoMoveAndRotate.cs.meta     |  11 +
 Assets/Scenes/SampleScene.unity      | 428 ++++++++++++++++++-------
 Assets/Scenes/Three_cubes.unity      | 577 ++++++++++++++++++++++++++++++++++
 Assets/Scenes/Three_cubes.unity.meta |   7 +
 5 files changed, 953 insertions(+), 111 deletions(-)
```

3

```
Luca@DESKTOP-68SH607 MINGW32 /d/Projects/AIV/UnityProjects/GIT_TEST (master)
$ git diff 1a3c3dcb0d653e6b41776e3b90d2e503f4235276 --stat *.cs
 Assets/AutoMoveAndRotate.cs | 41 +++++++++++++++++++++++++++++++++++++++++
 1 file changed, 41 insertions(+)
```

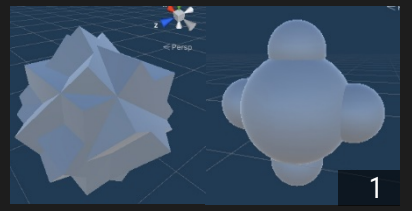
4

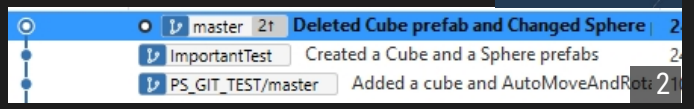
Cherry pick

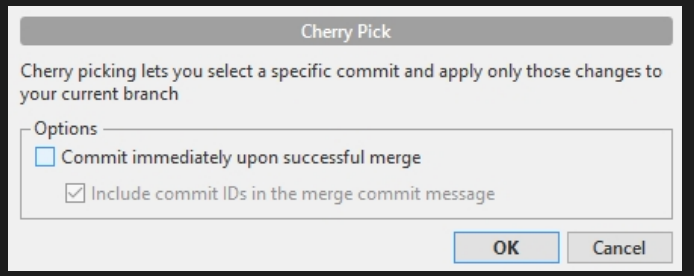
- Let’s say you want to pick some file from a previous commit

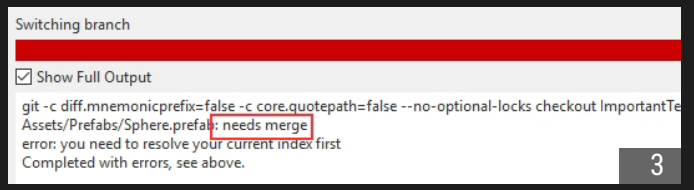
In Unity

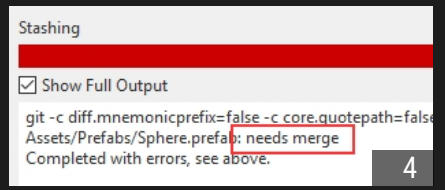
- Create a Cube prefab and a sphere prefab [1]
- Commit
- Delete the Cube prefab and change the sphere prefab
- Commit [2]
- Now let’s say you want back ONLY the Cube prefab from the previous commit
- Right-click on the commit you want to restore/Cherry Pick
  - Untick CommitImmediatlyUponSuccessfulMerge, it’s safer. Click OK
  - If it doesn’t show anything, try to View/Refresh the UI
- Now probably we have a combination of staged files and unstaged files, because there are conflicts
  - Click on Resolve conflicts/use Mine on the Sphere prefab (and on all the others conflicting files)
  - Click on “-” (minus) near the files you are not interested in
  - Commit: now you successfully picked up only the Cube prefab!





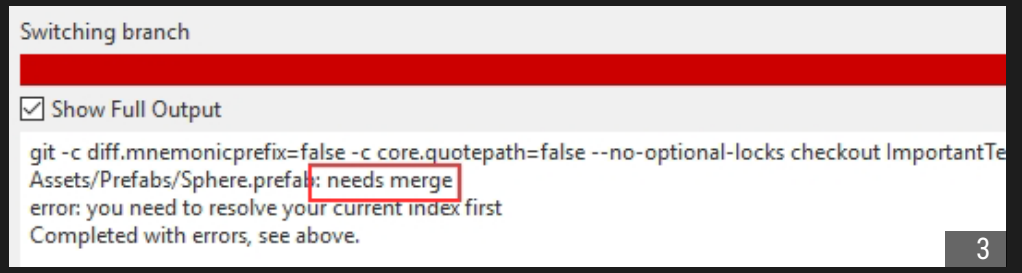


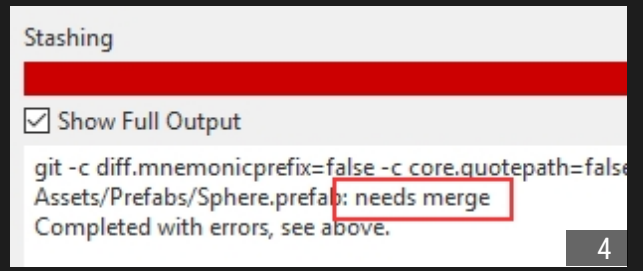




Common Problems

- Let’s say you have conflicts, so probably you have a combination of staged and unstaged files: this is a common situation. Imagine that: you’ve got uncommitted changes, how do you change branch now, without losing anything?
  - Try to switch branch: Error, you need to merge [3]
  - Try to Stash what you’ve got: Error, you need to merge [4]
    - One quick way to do that would be: StageAll, then Stash

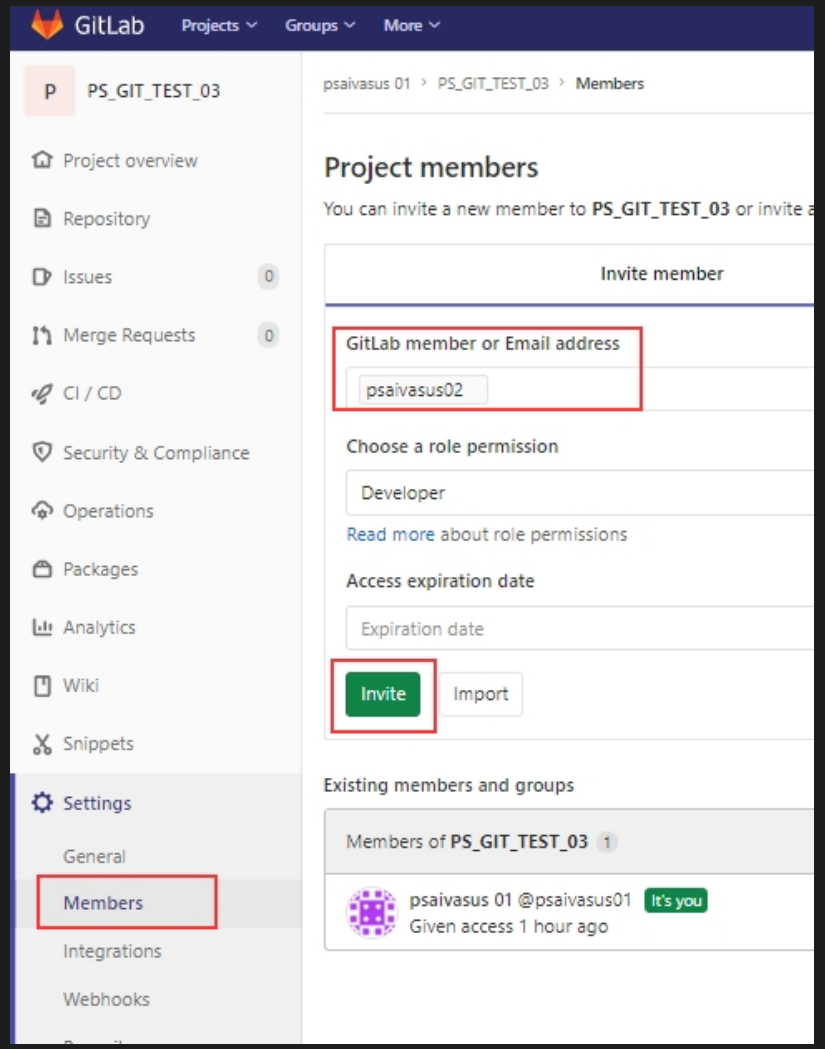




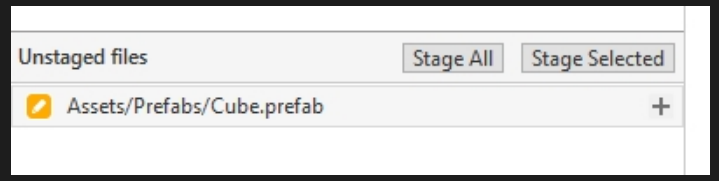
Team members

GitLab

- Login on the project administrator (Maintainer - M) account
  - ProjectSettings/Members/InviteMember



Unity serialization & Prefabs



These are the correct settings in Unity to serialize project files for Git

- ProjectSettings/Editor
  - VersionControl VisibleMetaFiles
  - AssetSerializationMode ForceText

- Add a prefab to SampleScene
- Commit
- Change the prefab inside that scene

ST

- The only unstage file is the prefab, not the scene! This is good: one person could be working on the same file the other could be working on the prefab!

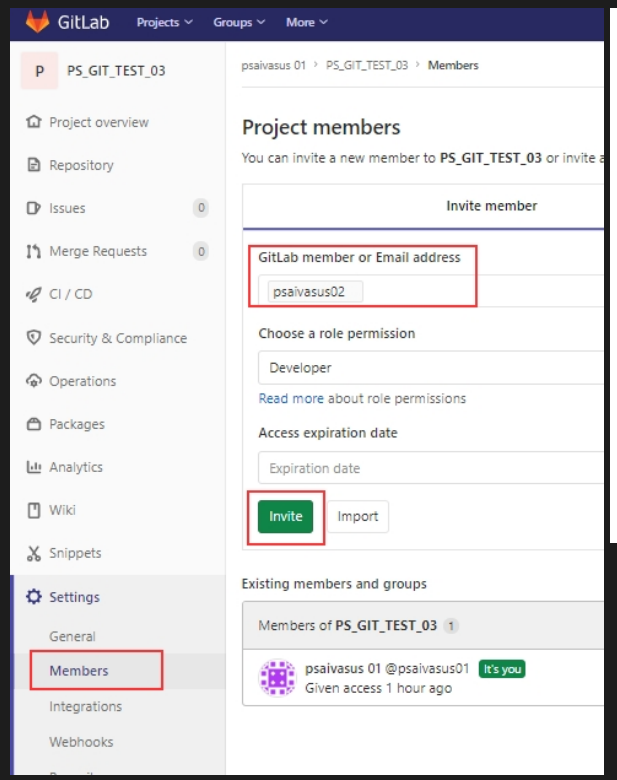
Issues / Milestones

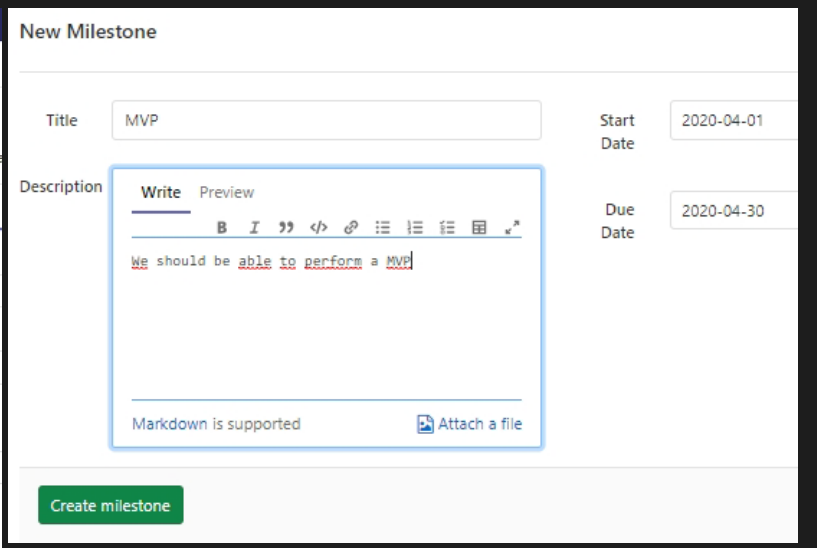
GitLab
- Login on the project administrator (Maintainer, M) account
- Login on the project Developer (D)
  - Now the common project should appear under Projects/YourProjects

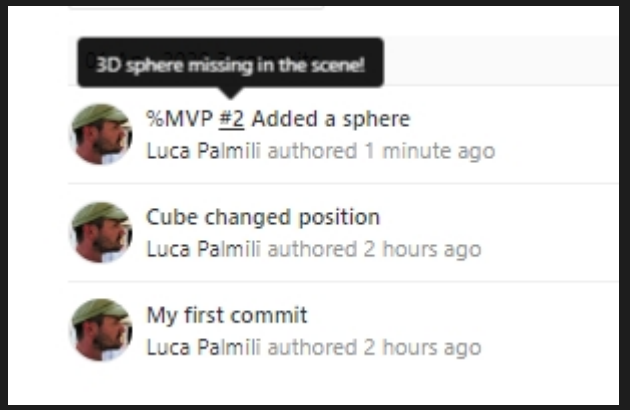
M
- Create a Milestone “MVP”
- Create an issue and assign to D

D
- Ask something about this issue to M
- Close the issue
- Open an Issue on MVP: ask to add a sphere

M
- Add a sphere and perform a commit/push
- Since you are solving the issue #2, if you write in your commit: #2 you link your commit comment to the issue, when it will be displayed in GitLab/Repository/Commits
- The issue number is listed under GitLab/Issues/List







Snippets

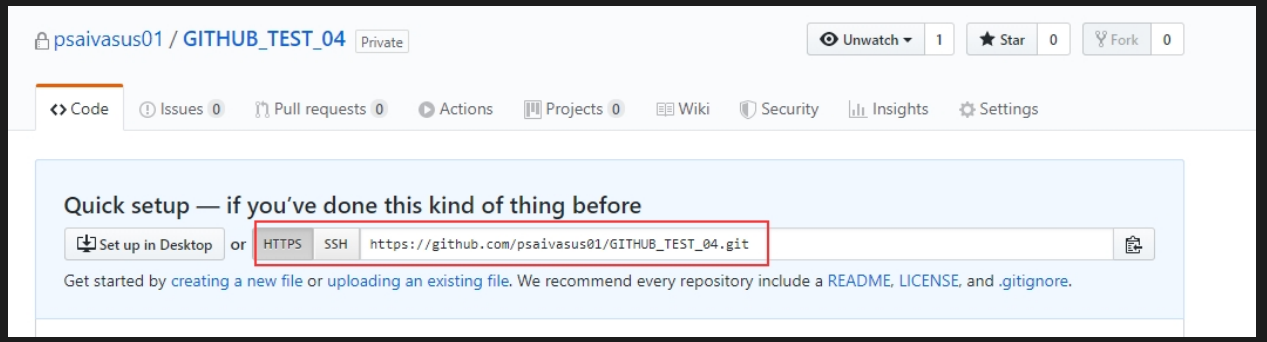
GitLab

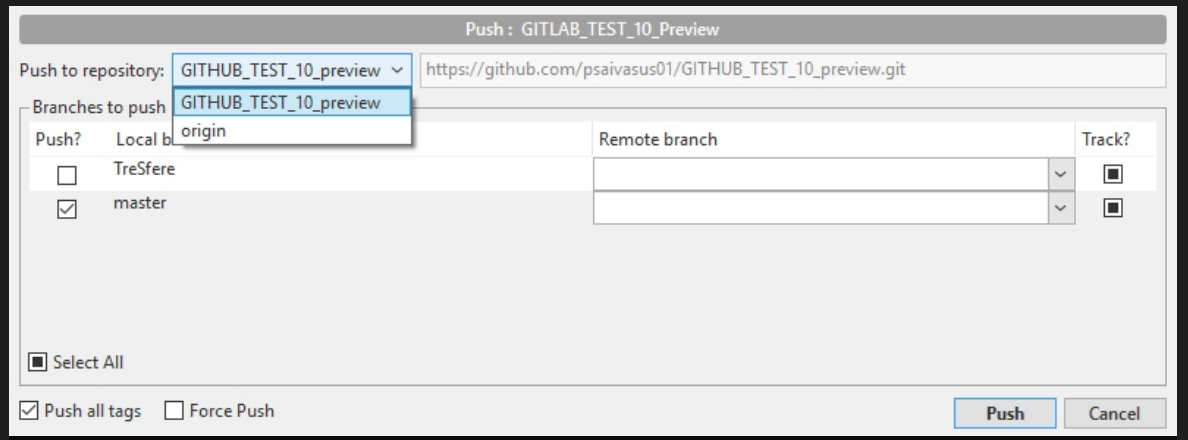
- Snippets (equivalent to GitHub Gists) are text/code files that you want to share inside your project
- GitLab/Snippets/NewSnippets
- You can refer snippet #2 using $2 in your commit message

Use GitHub

We are going to add GitHub account to your local Repo

- Create GitHub account
- Add your GitHub account to ST (login using OAuth)
- Add your remote GitHub Account to your existing GitLab Account
- Perform the first Push



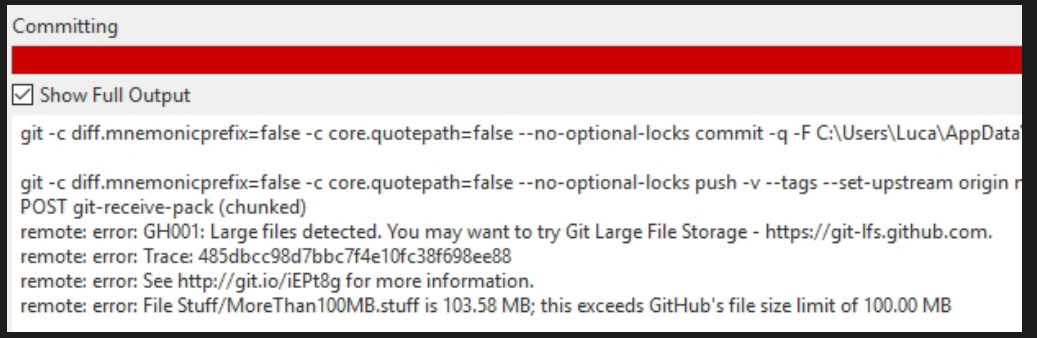


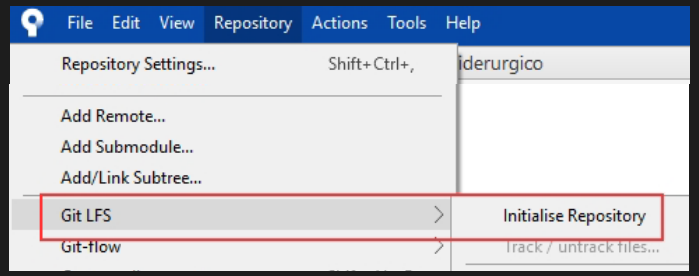
You can also use more than a Remote Repo linked at one single Local Repo

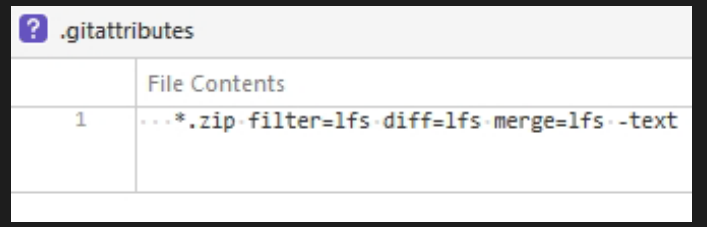
LargeFileSystem LFS

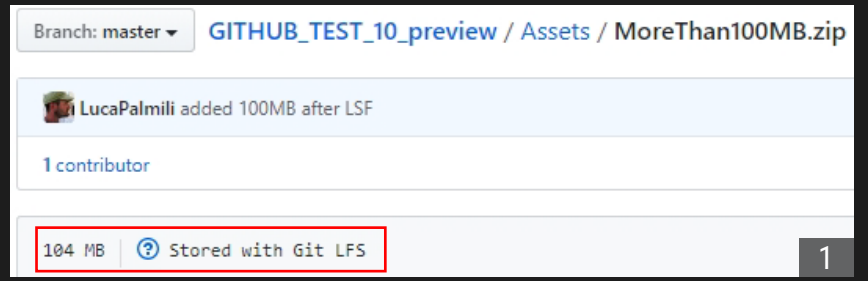
- We're going to go over the GitHub 100MB limit per single file
- We need to setup Git LFS: Repository/GitLFS/InitializeRepository
- Add *.zip file extension
- This will create a .gitattributes file, that keeps tracks of LFS files
- AFTER this setup, add a file bigger than 100MB into your Asset folder [MoreThan100MB.zip]
- Push on GitHub
- Go to your remote GitHub account and click on the zip file. You should see "Stored with Git LFS" note [1]

Without this setup , an upload bigger than 100MB would result in error [1]





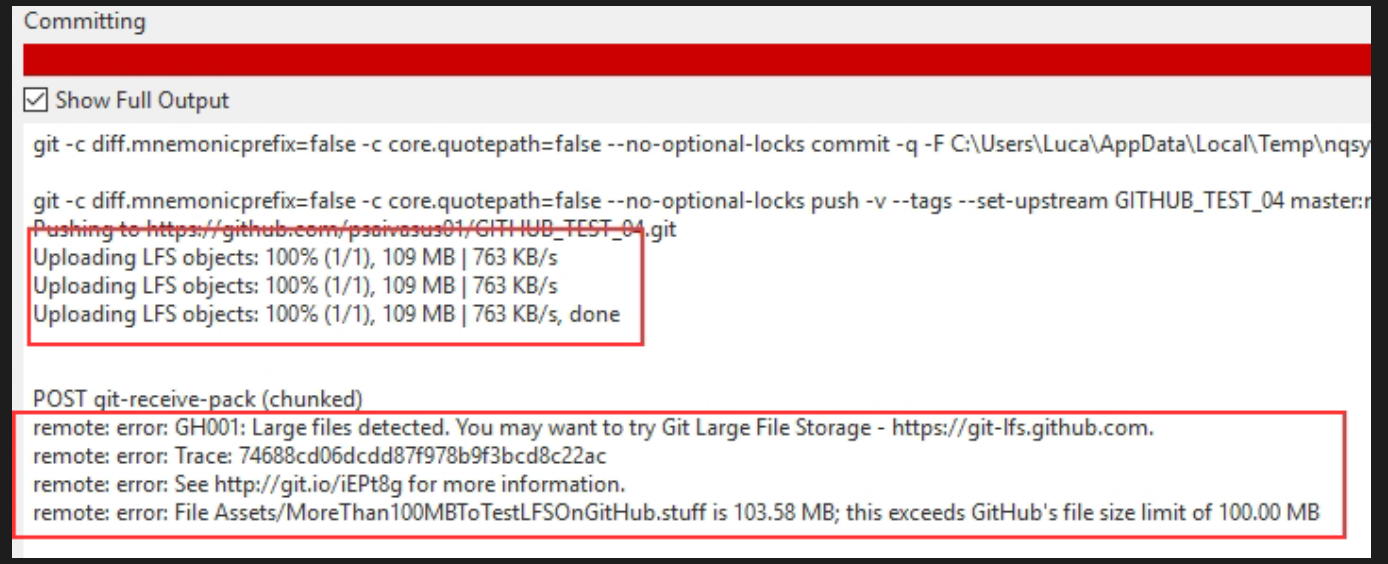


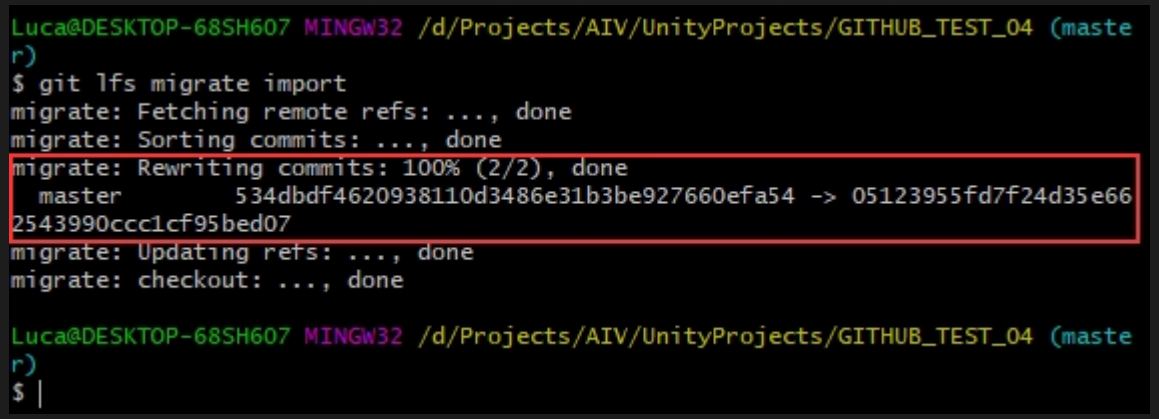


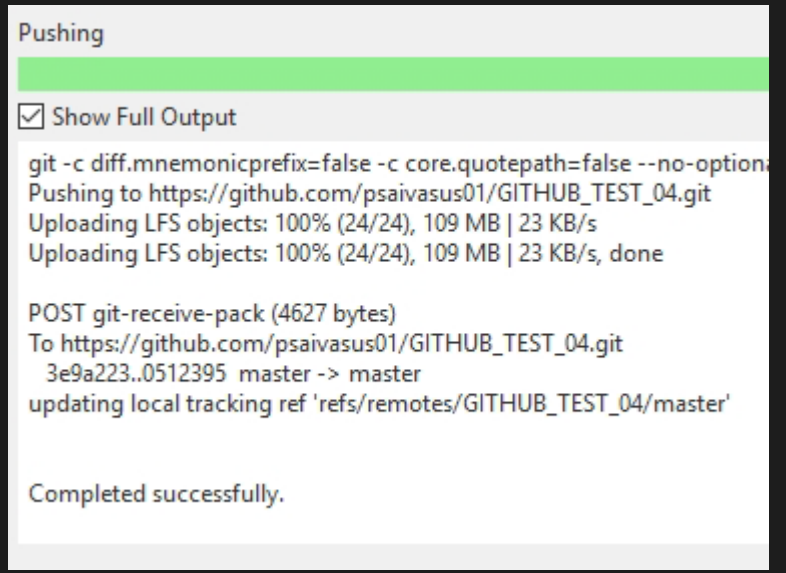
LFS

If you added some large file previously, when you hadn't Git LFS configured yet, you need to migrate over to get LFS

- In this case, some file in our history > 100MB wasn't correctly added to LFS
- Type git lfs migrate import in the terminal to rewrite the history (some commit SHA numbers could change)
- Check if .gitignore had some changes
- Push again
    - Now you should have no errors







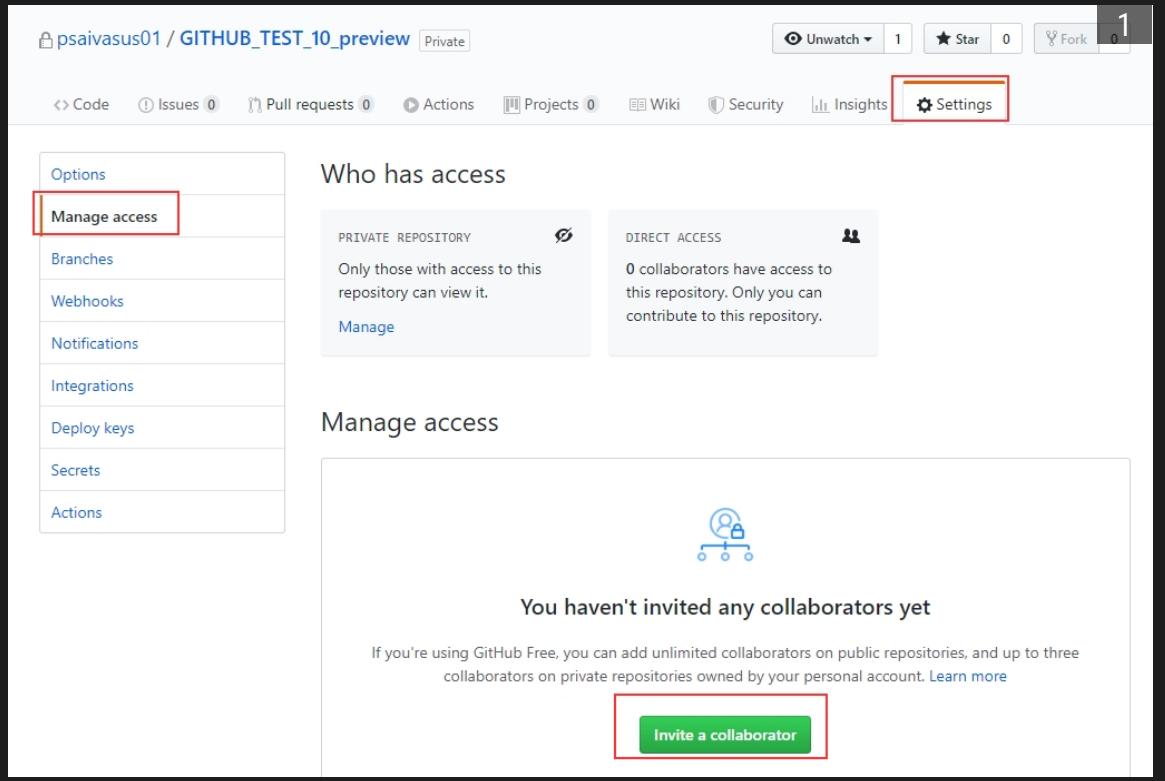
GitHub Team members

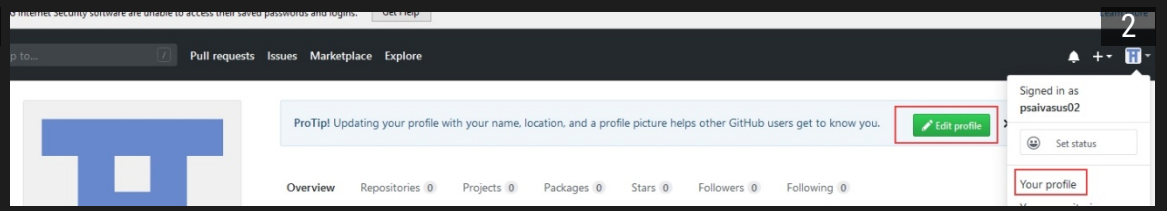
GHub (Manager)

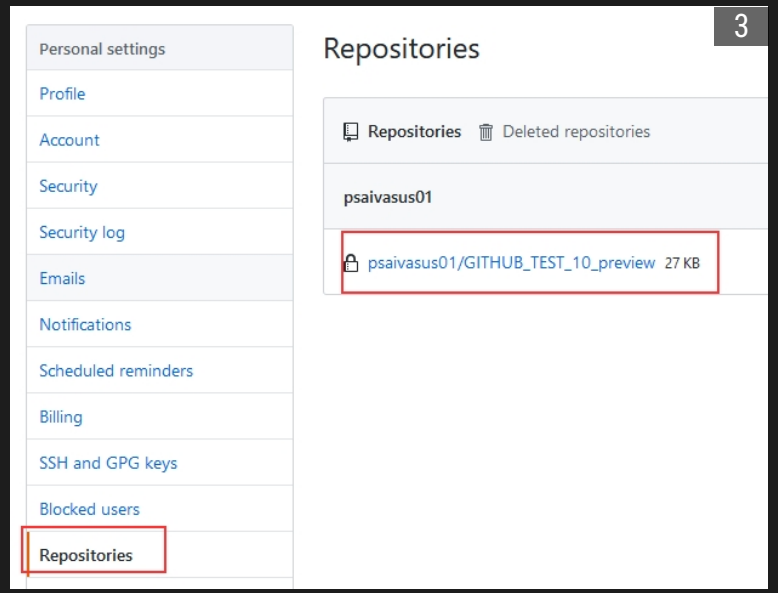
- Choose one Manager and add other Collaborators emails
- GitHub/YourProfile/Repositories
- Go to Settings repository tab /ManageAccess/InviteCollaborator [1]

GHub (Collaborator)

- Accept Manager invitation delivered to your email
- GitHub/YourProfile/EditProfile [2]
- From the left column, choose /Repositories
- You should see the Manager repository [3]







GitHub Unity

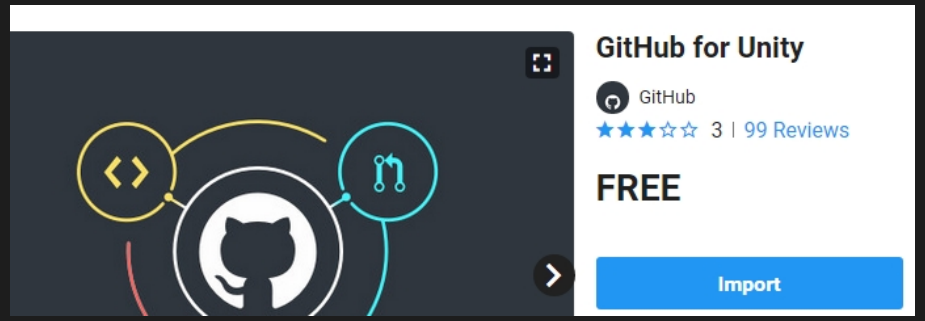
GHub (Collaborator)
- Clone Manager repository in your local Repo: it is important to have new local Repo ONLY with the GitHub account

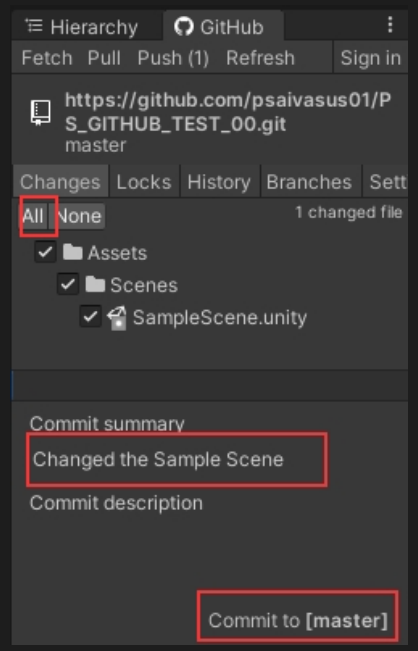
GHub (Manager)
Unity
- Add GitHubForUnity Asset store plugin

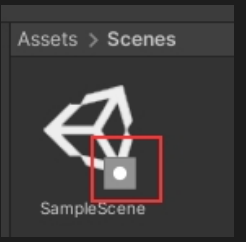
ST (Manager)
- Commit changes, Push on GitHub Repo

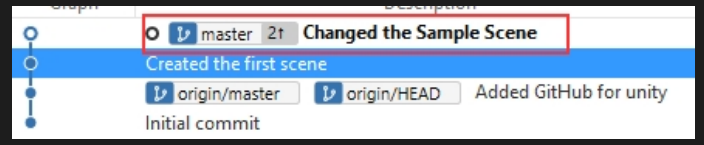
ST (Collaborator)
- Pull

Unity (Manager & Collaborator)
- You can access the GitHub plugin opening Window/GitHub
- Login into GitHub from Unity
- From now on
  - you can use Sourcetree or the githubPluginTool inside unity to commit changes: they are both talking to Git, GitPlugin for Unity is just another git UI, like SourceTree
    - Use always ST, is safer
  - Unity ProjectSettings are configured in the right way
  - LFS is correctly configured: now SourceTree Repository/GitLFS has InitializeRepository option disabled, even if you
  - You have Locks









Lock

- Push/Pull buttons are at the top side of the GitHub plugin
- If Pull is not refreshed, try to Fetch first
- Use refresh to refresh locks status

- It is a simple messaging system to show you that a given file is being intended to be edited by somebody else

- If you want to change a scene and you want to be sure nobody makes changes to that scene
- Right click on the asset you want to lock
  - A yellow icon appears near that asset, and the list of locks is refreshed
  - Now if someone else wants to edit that file:
    - He should open a yellow-locked assets (as a warning)
    - He should save changes and see a purple Lock icon on it as a double-warning (hey, you are changing something that was locked!)
  - So if I have some purple-locked file inside my Changes list in Unity, I can
    - Talk to the other person that is locking the file: maybe my change is more important
    - Discard my change (right click inside GtiHub/Changes plugin tab/discard): the asset-icon should be back to yellow again
- If I want to release the lock, right click on the asset and then release
  - If others don't see the yellow-lock icon disappear, they can try to click the Refresh button inside GitHub window