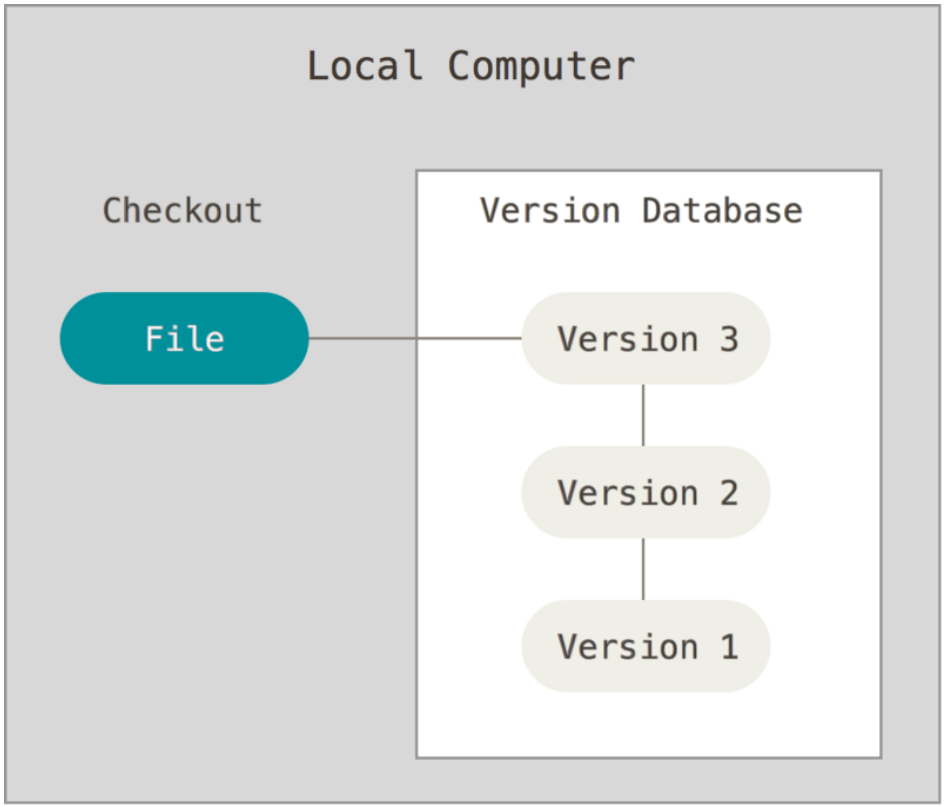
# Git

Git Introduction

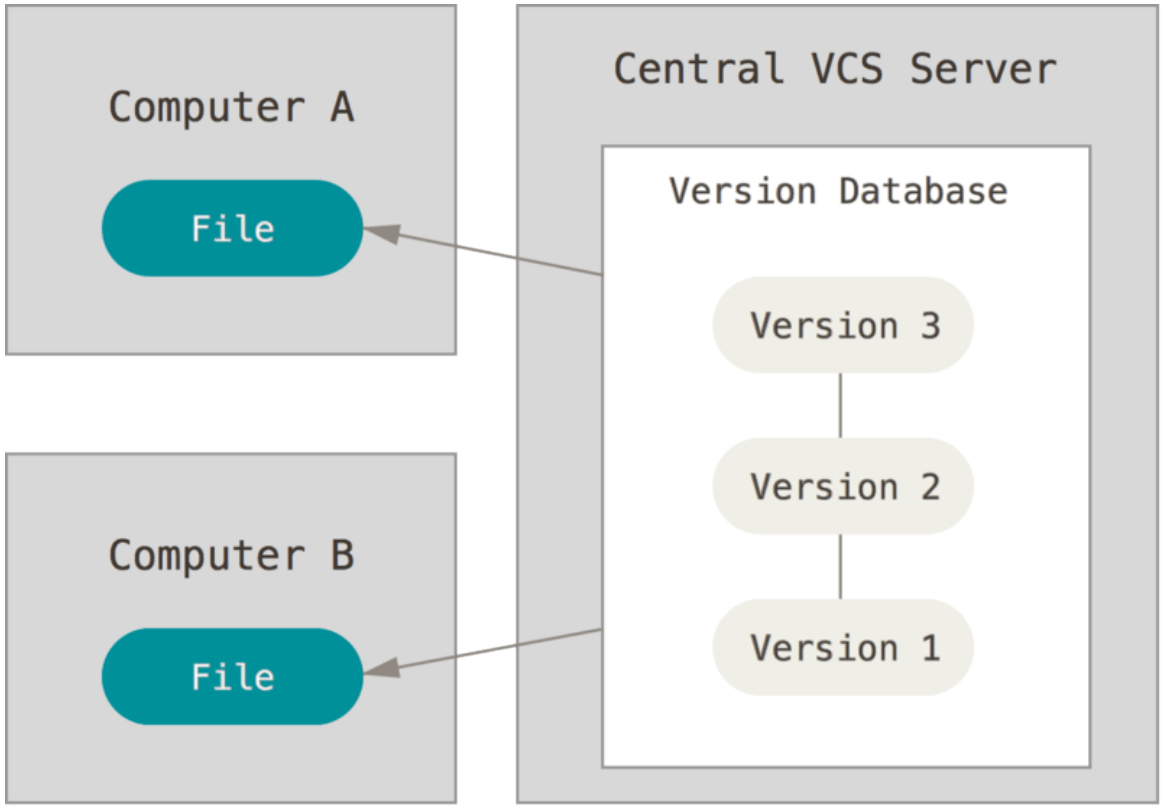
# What is git?

Git is a free and open-source **distributed version control system** designed to handle everything from small to substantial projects with speed and efficiency.

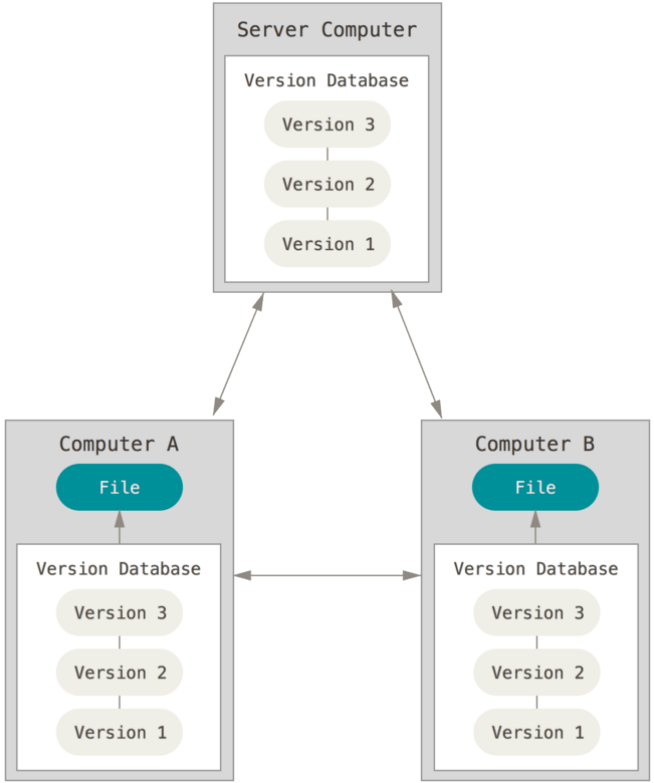
# Types of version controls


Local Computer
Checkout
Version Database
File
Version 3
Version 2
Version 1

Centralized Version Control Systems

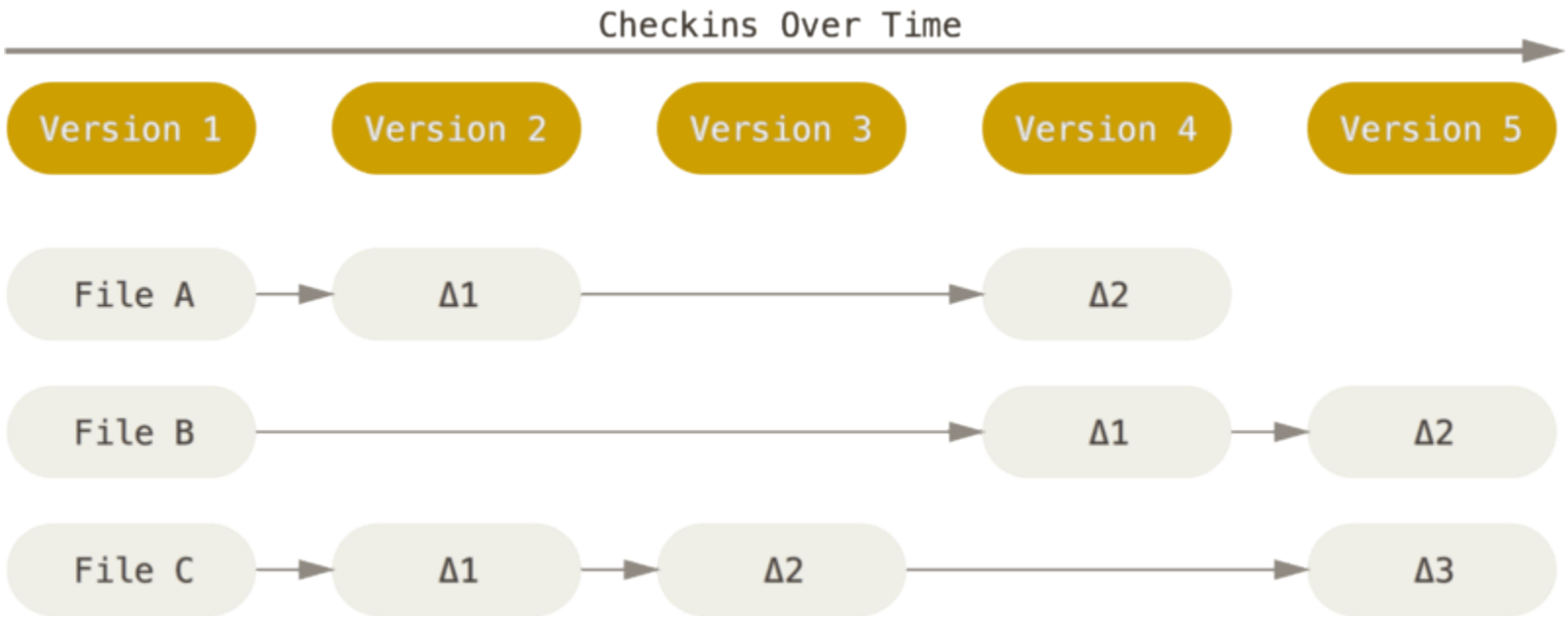


# Distributed Version Control Systems



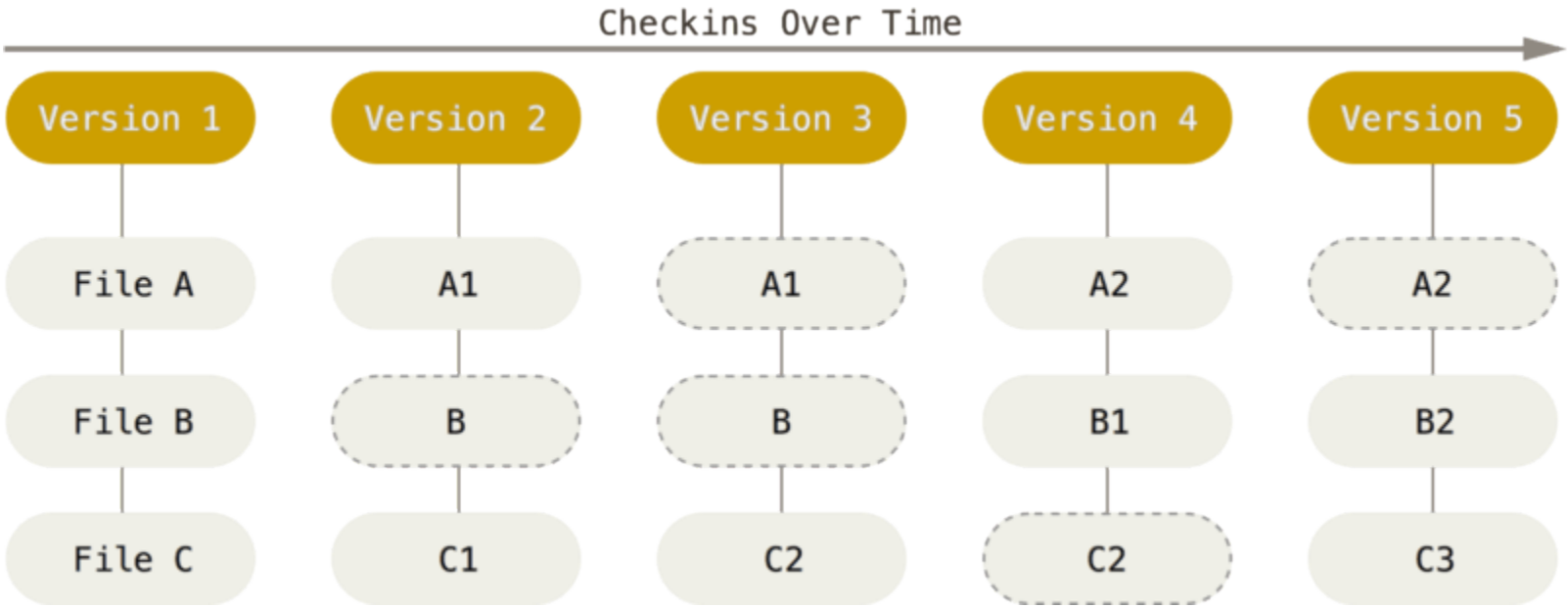
# Git Fundamentals

# Snapshots, Not Differences




Storing data as changes to a base version of each file

# Snapshots, Not Differences




Storing data as snapshots of the project over time

## Nearly Every Operation Is Local

Most operations in Git need only local files and resources to operate — generally no information is needed from another computer on your network.

# Git Has Integrity

Everything in Git is checksummed before it is stored and is then referred to by that checksum. This means it’s impossible to change the contents of any file or directory without Git knowing about it.

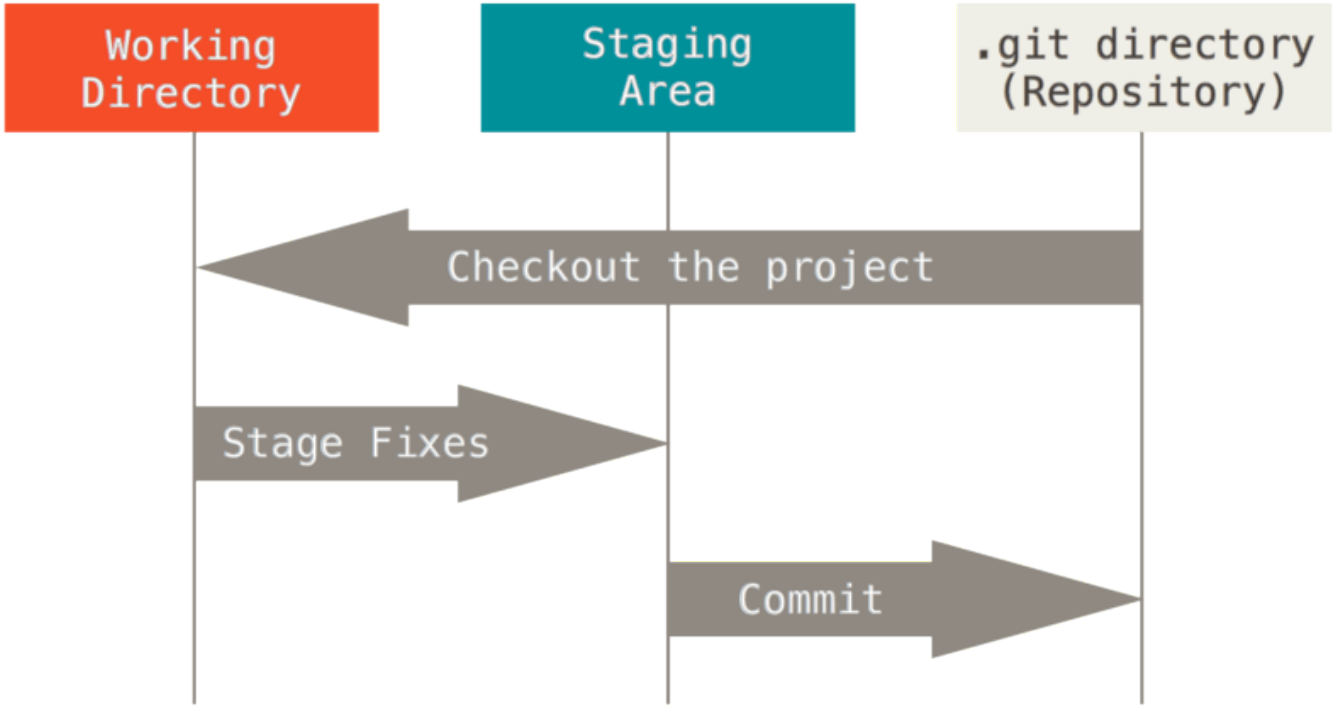
functionality is built into Git at the lowest levels and is integral to its philosophy. You can’t lose information in transit or get file corruption without Git being able to detect it.

A SHA-1 hash looks something like this:
24b9da6552252987aa493b52f8696cd6d3b00373

# Git Generally Only Adds Data

When you do actions in Git, nearly all of them only add data to the Git database. It is hard to get the system to do anything that is not undoable or to make it erase data in any way. As with any VCS, you can lose or mess up changes you haven’t committed yet, but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository.

# The Three States




Working tree, staging area, and Git directory

# Installing and setup git

# Installing Git

- Installing on Windows

  The most official build is available for download on the Git website. Just go to https://git-scm.com/download/win and the download will start automatically.

- Installing on Linux

  If you're on a Debian-based distribution, such as Ubuntu, try apt:

  $ sudo apt install git-all

- Installing on macOS

  The most official build is available for download on the Git website. Just go to https://git-scm.com/download/mac and the download will start automatically.

# First-Time Git Setup

Now that you have Git on your system, you'll want to do a few things to customize your Git environment. You should have to do these things only once on any given computer; they'll stick around between upgrades

Git comes with a tool called **git config** that lets you get and set configuration variables that control all aspects of how Git looks and operates. These variables can be stored in three different places:

1. **[path]/etc/gitconfig file**: Contains values applied to every user on the system and all their repositories. If you pass the option --system to git config, it reads and writes from this file specifically. Because this is a system configuration file, you would need administrative or superuser privilege to make changes to it.
2. **~/.gitconfig or ~/.config/git/config file**: Values specific personally to you, the user. You can make Git read and write to this file specifically by passing the --global option, and this affects all of the repositories you work with on your system.
3. **config file in the Git directory (that is, .git/config) of whatever repository you're currently using**: Specific to that single repository. You can force Git to read from and write to this file with the --local option, but that is in fact the default. Unsurprisingly, you need to be located somewhere in a Git repository for this option to work properly.

You can view all of your settings and where they are coming from using: git config --list --show-origin

# Your Identity

The first thing you should do when you install Git is to set your user name and email address. This is important because every Git commit uses this information, and it’s immutably baked into the commits you start creating:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

# Your default branch name

By default Git will create a branch named master when you create a new repository with git init. From Git version 2.28 onwards, you can set a different name for the initial branch.

To set main as the default branch name do:

```
$ git config --global init.defaultBranch main
```

# Checking Your Settings

If you want to check your configuration settings, you can use the git config --list command to list all the settings Git can find at that point:

```
$ git config --list
user.name=John Doe
user.email=johndoe@example.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

You may see keys more than once, because Git reads the same key from different files ([path]/etc/gitconfig and ~/.gitconfig, for example). In this case, Git uses the last value for each unique key it sees.
You can also check what Git thinks a specific key value is by typing git config <key>:

```
$ git config user.name
John Doe
```

# Performing a local Git workflow in Visual Studio

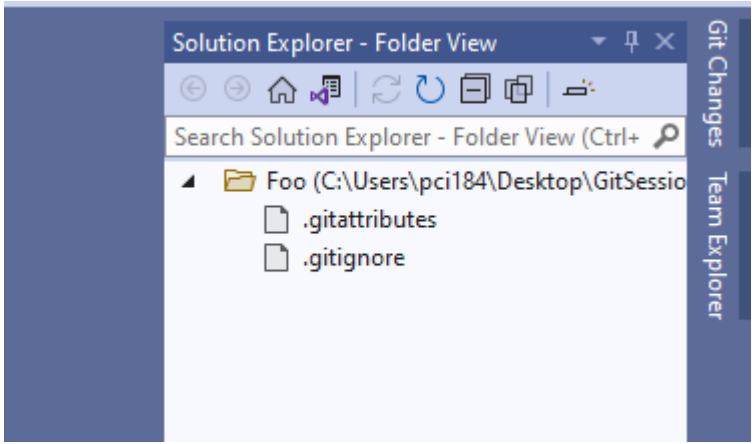
# Create Directory and add Files,Create Repository

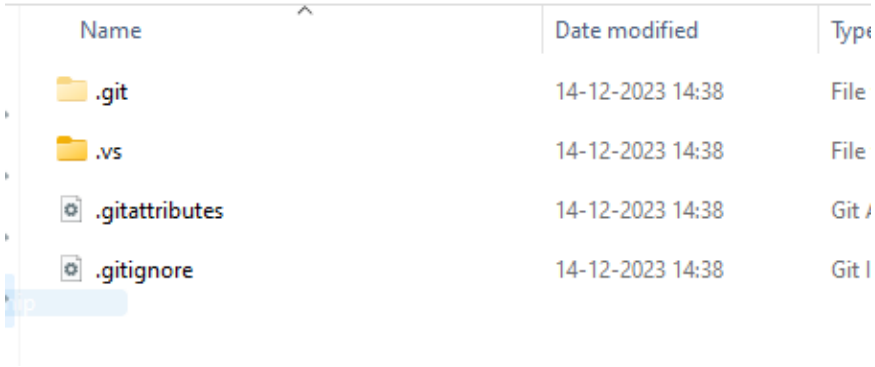
**=> Make a new directory**

**=> Open it in Visual Studio**

**=> From Menubar go to: Git -> Create Git Repository -> Apply Local Only -> Create**

**.git folder** gets created
Git does not care whether you start with an empty directory or if it already contains files.
All files inside the repository folder, excluding the **.git folder**, are the working tree.



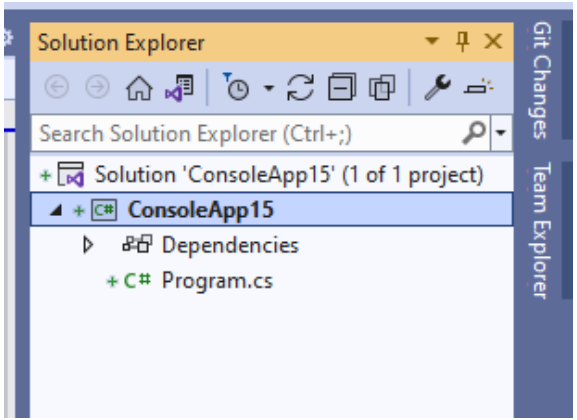



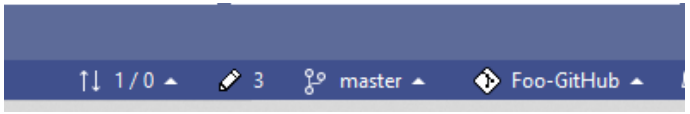
# Create A new Project, Check current status

**=> Go to: File -> New -> Project -> Console App(c#) ->Set location(created directory) -> Next -> Give name ->Next -> Create**

Here “+” sign shows
“Pending Add”

These are Untracked Files



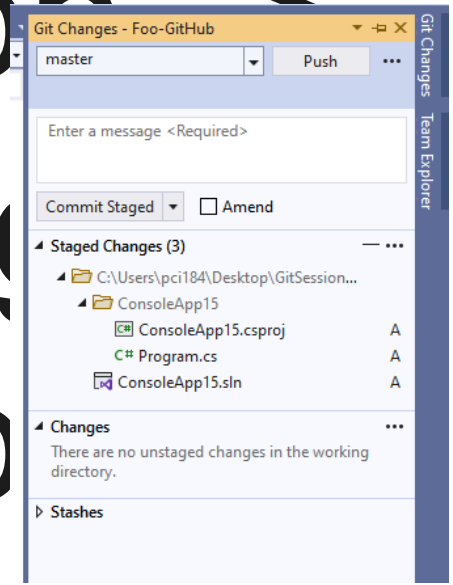



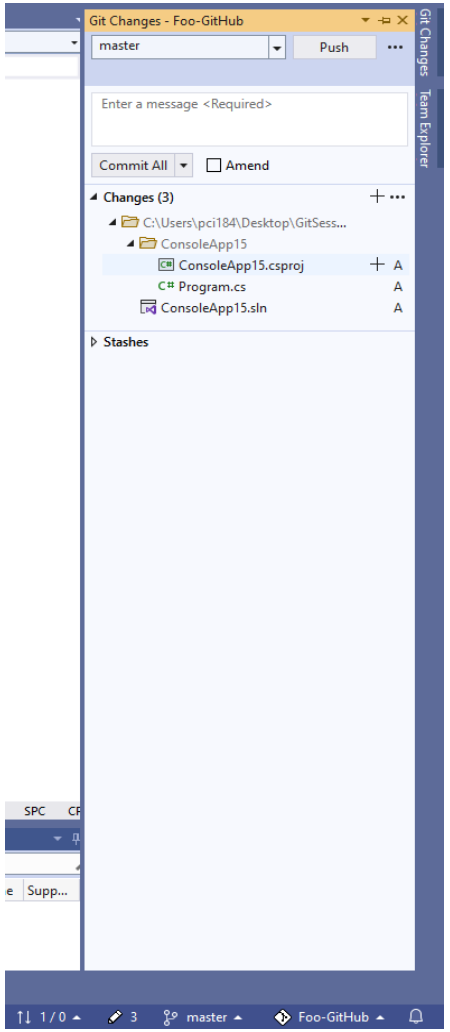

At the bottom,

Pencil icon shows the “Changes made”

to repo.

# Click on that pencil icon ➔ Git Chang window o



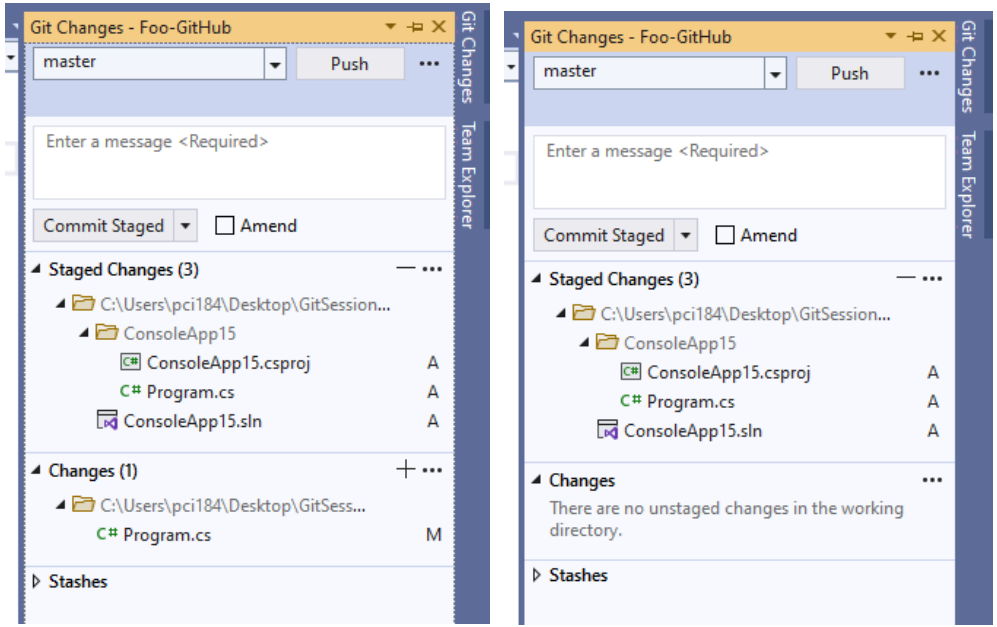



# Changes to Program.cs

Add one line to Program.cs,
It Comes under Changes tab.

”M” indicates Modified file.

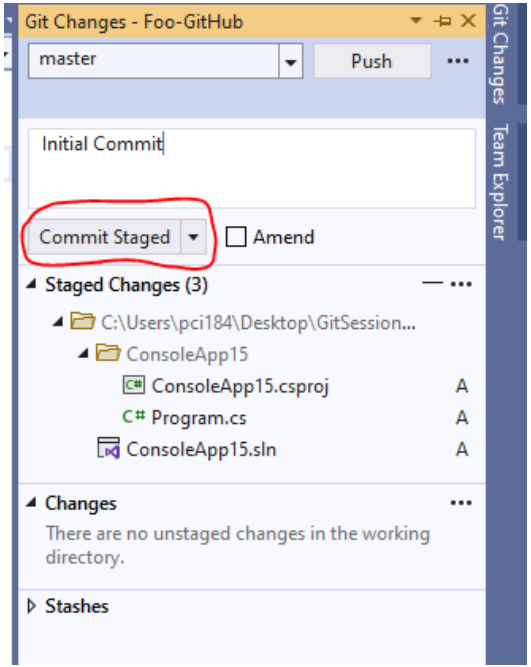
Hover on Program.cs in Changes and click ”+” to stage those changes

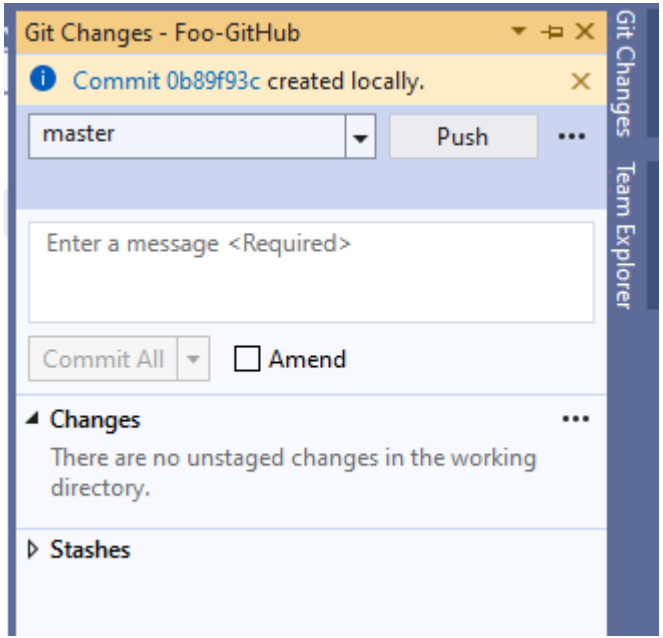


# Commit staged changes to the repository

The commit operation, created a new version of your files in the local repository inside the .git folder.

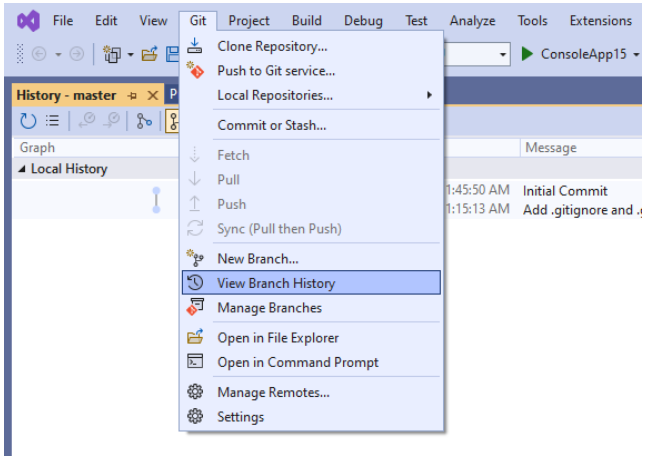
Do not forget to add the Commit message.

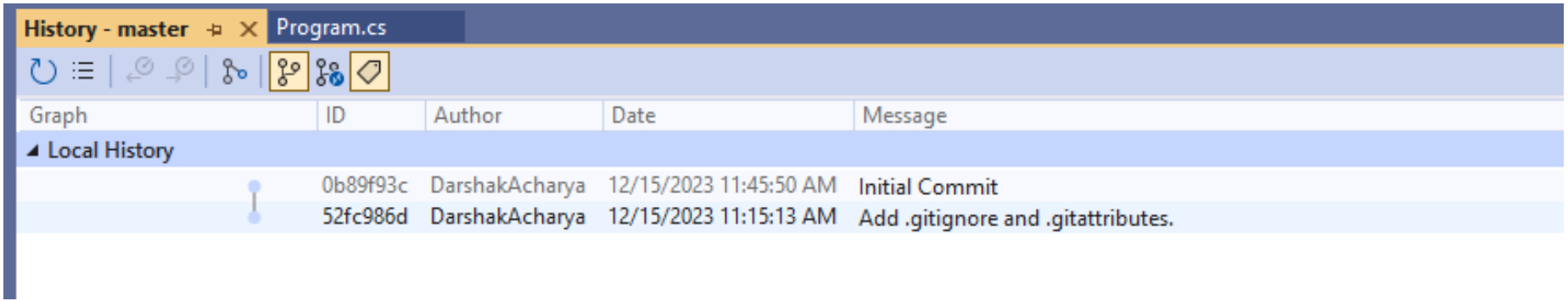


# Viewing the commit history

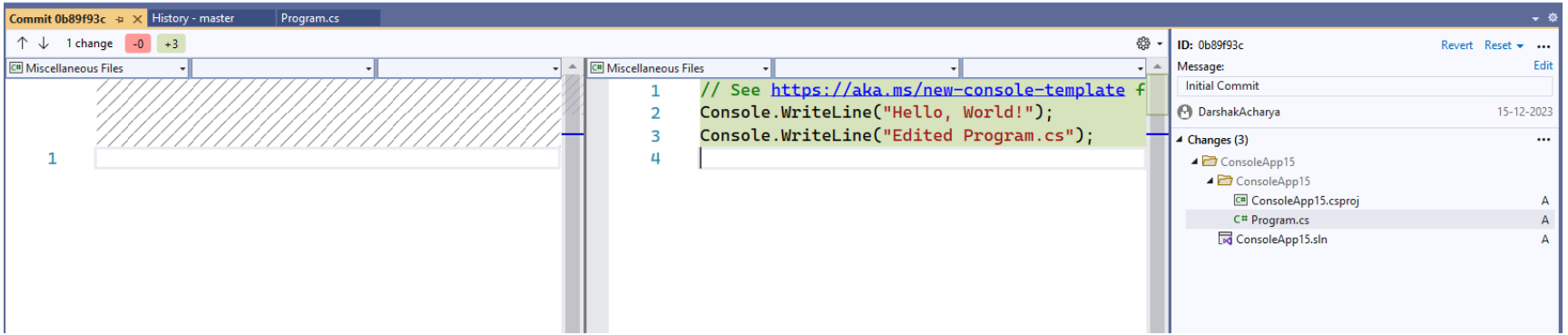
Go to the Git -> View Branch History



# Viewing the changes of a commit

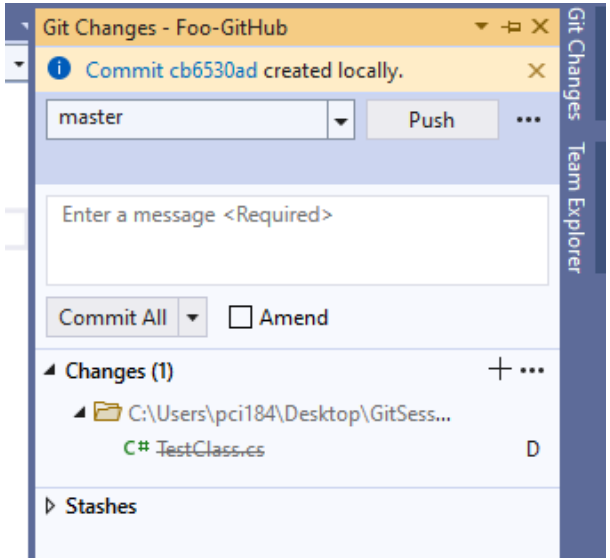
Double click on any commit to see the changes of that commit



# Remove files

```
Add a new Class “TestClass.cs”
Now Stage and Commit it.


Now remove the class.
```

```
Changes will be reflected when clicking Pencil Icon/Git Changes


Here we have deleted the file so “D” denotes that.

Also one line is drawn to show the deleted/removed file.

Again, Stage and Commit the changes with a message.
```



# Correct the commit with git amend

Add new class “ServicesClass.cs”

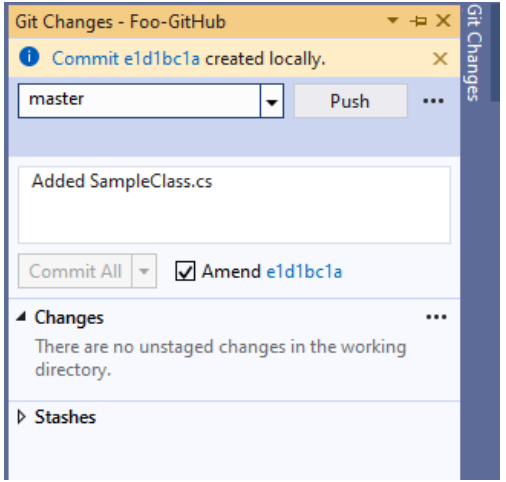
Stage and commit it with the message “Added SampleClass.cs”

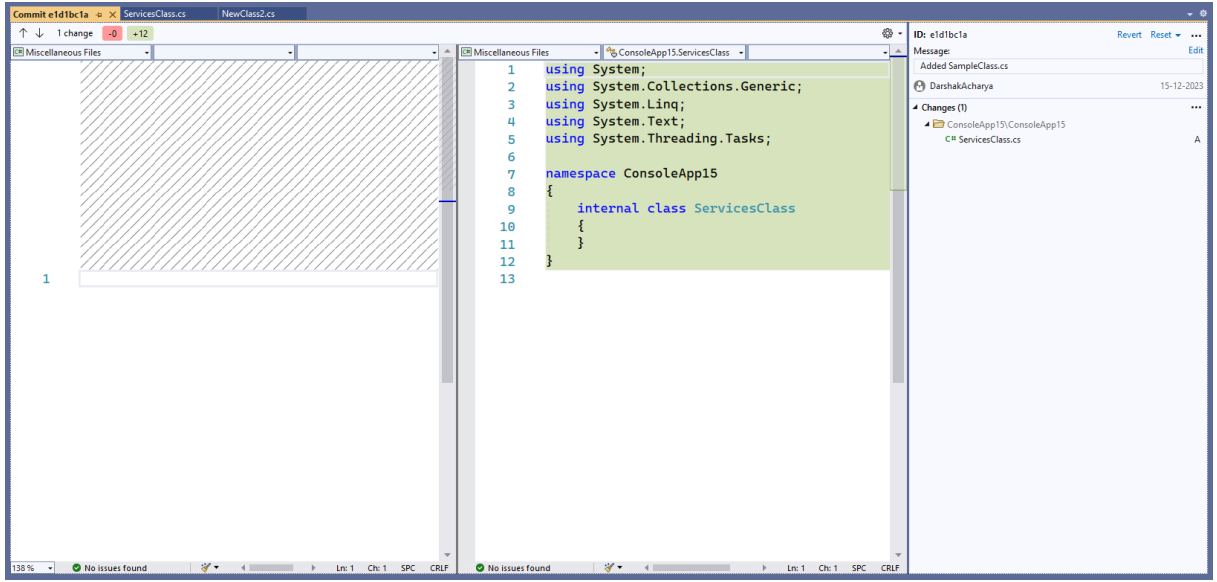
Commit it.

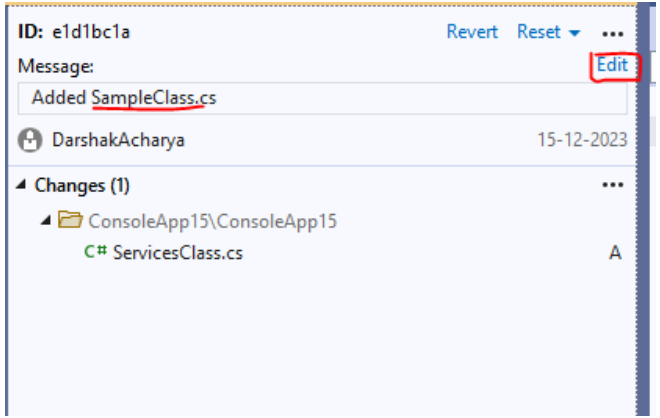
Here, we have written “SampleClass.cs” instead of “ServicesClass.cs” by mistake.

To correct this tick on Amend from Git Changes window.

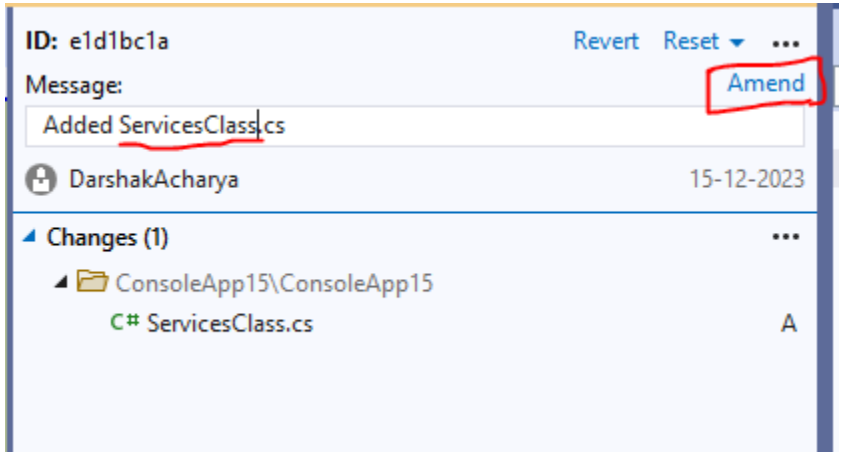
Now Click on CommitID(blue-number) near Amend checkbox.

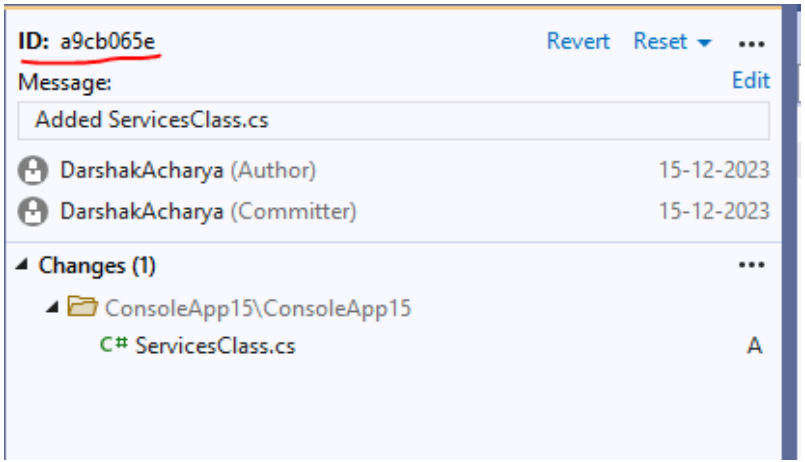



Commit e1d1bc1a
ServicesClass.cs
NewClass2.cs
1 change
-0
+12
Miscellaneous Files
ConsoleApp15.ServicesClass
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp15
{
internal class ServicesClass
{
}
}
No issues found
Ln: 1
Ch: 1
SPC
CRLF
ID: e1d1bc1a
Revert
Reset
Message:
Edit
Added SampleClass.cs
DarshakAcharya
15-12-2023
Changes (1)
ConsoleApp15\ConsoleApp15
ServicesClass.cs
A


ID: e1d1bc1a
Revert
Reset
Message:
Edit
Added SampleClass.cs
DarshakAcharya
15-12-2023
Changes (1)
ConsoleApp15\ConsoleApp15
ServicesClass.cs
A

It creates a new commit with the adjusted changes. The amended commit is still available until a clean-up job removes it. But it is not included in the git log output hence it does not distract the user.



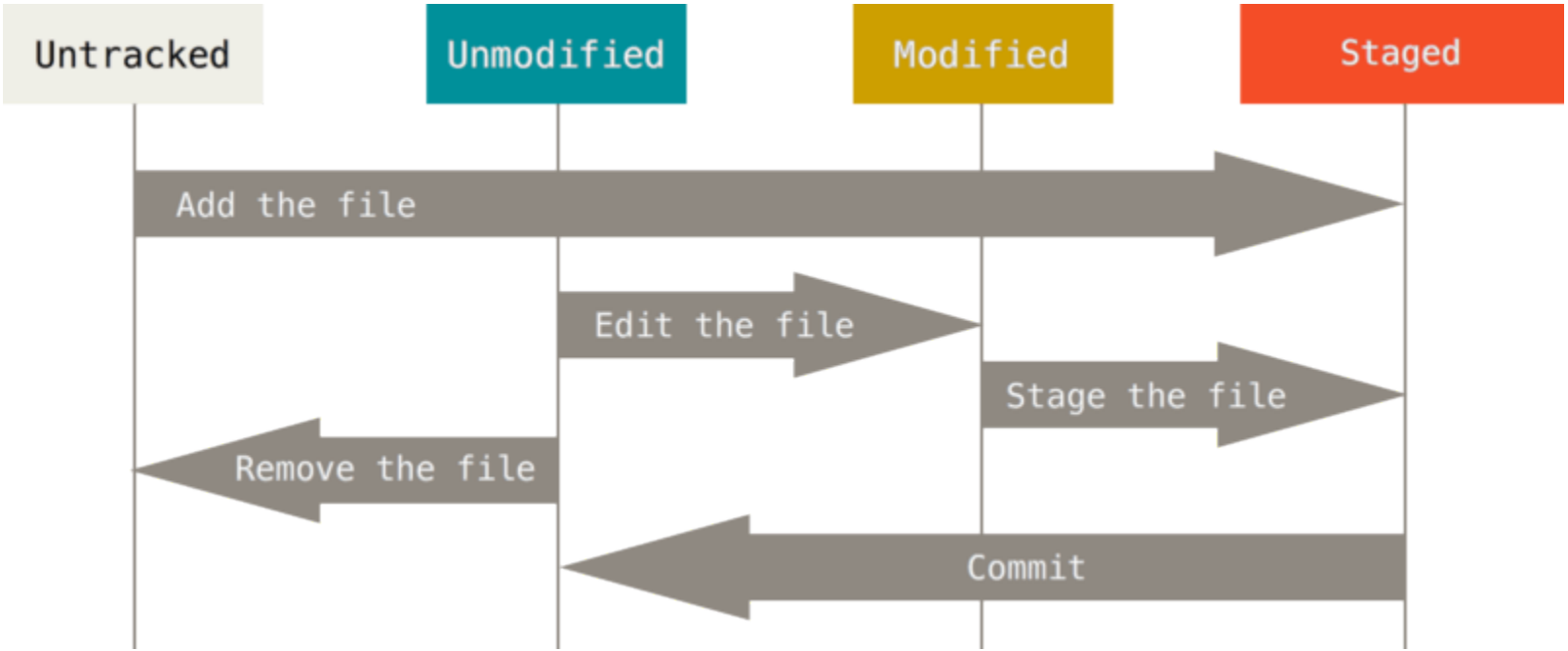



# Important note for git amend

**You should use the git --amend command only for commits which have not been pushed to a public branch of another Git repository.**

**The git --amend command creates a new commit ID and people may have already based their work on the existing commit. If that would be the case, they would need to migrate their work based on the new commit.**

# The lifecycle of the status of your files



# Working with Remotes

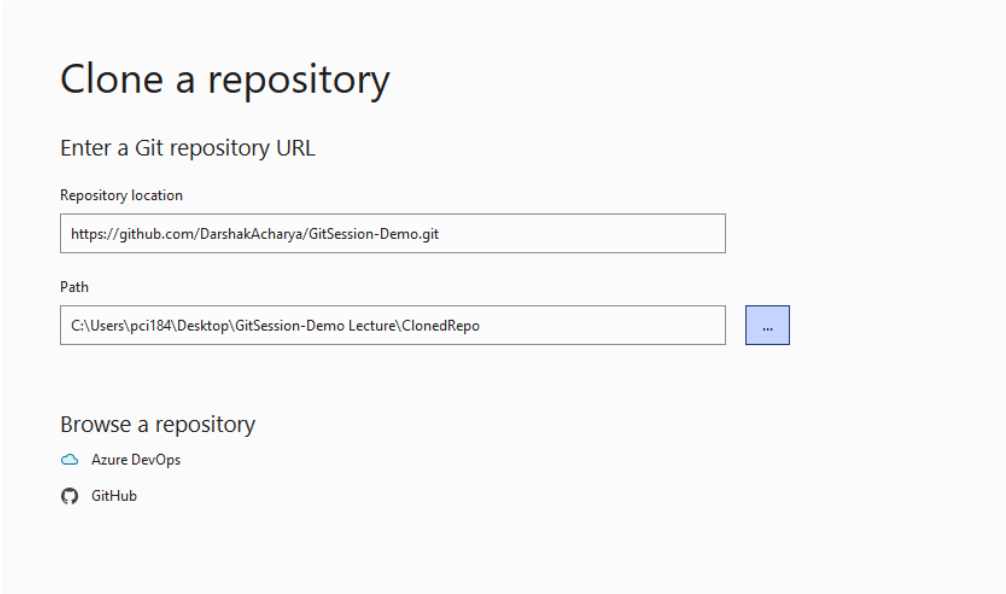
# What are remotes?

- Git allows you to synchronize your repository with more than one remote repository.
- Remote repositories are versions of your project that are hosted on the Internet or network somewhere.
- In the local repository you can address each remote repository by a shortcut. This shortcut is simply called remote.

# Cloning a repository

The git clone command copies an existing Git repository. This copy is a working Git repository with the complete history of the cloned repository. It can be used completely isolated from the original repository. Git supports several transport protocols to connect to other Git repositories; the native protocol for Git is also called git.

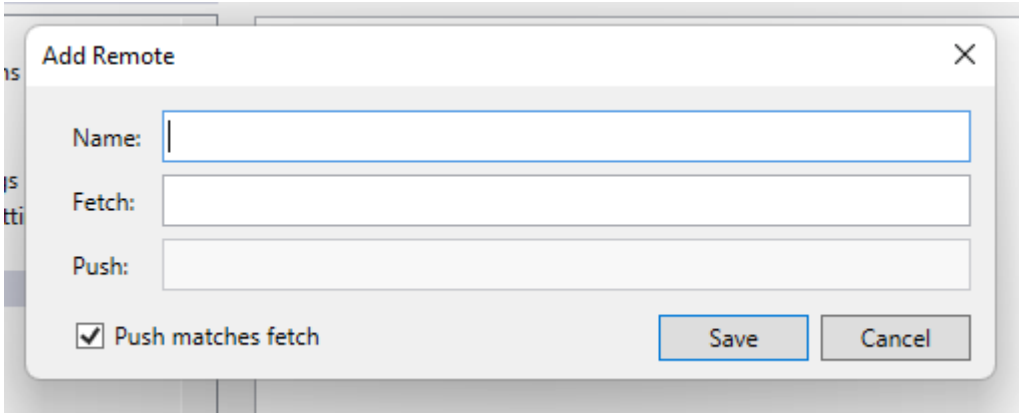
```
Go to Git -> Clone Repository
Set repository link, Add Path and Hit Clone.
```



# Adding remote repositories

If you clone a repository, Git implicitly creates a remote named **origin** by default. The origin remote links back to the cloned repository.
You can push changes to this repository via git push as Git uses origin as default. Of course, pushing to a remote repository requires write access to this repository.
You can add more remotes via the git remote add

```
Go to Git -> Manage Remotes -> Add(Name, Fetch, Push) -> Save -> OK
```



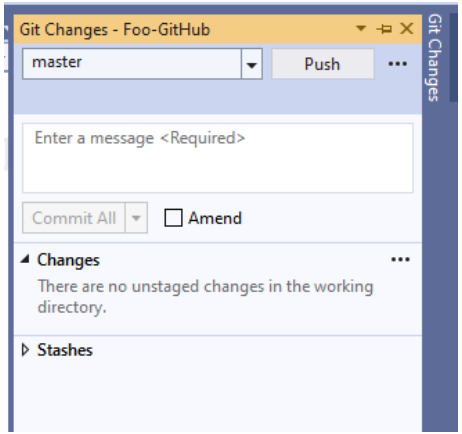
# Rename remote repositories

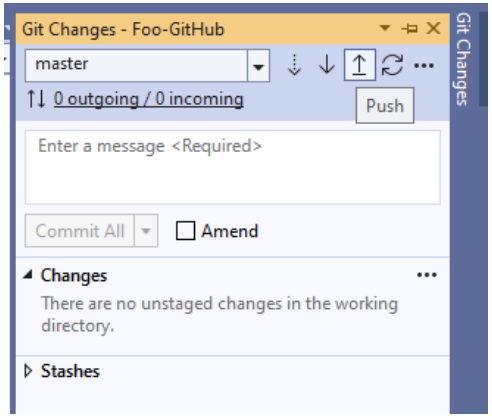
```
## Rename an existing remote
$ git remote rename pb paulboone
$ git remote -v
origin       git@github.com:kathak-dabhi/git-demo-excersice.git (fetch)
origin       git@github.com:kathak-dabhi/git-demo-excersice.git (push)
paulboone   https://github.com/paulboone/git-demo-excersice.git (fetch)
paulboone   https://github.com/paulboone/git-demo-excersice.git (push)
```

# Push changes to remote repository

The Git Push command allows you to send data to other repositories.
By default it sends data from your current branch to the same branch of the remote repository.

```
Make sure you have an account on GitHub
Git Changes -> Push (Fill account details, choose Private/Public preferences)
```

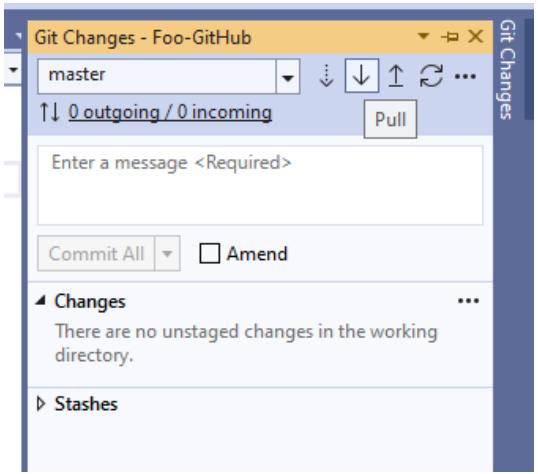


# Pull changes from remote repository

The git pull command allows you to get the latest changes from another repository for the current branch. The git pull command is actually a shortcut for git fetch followed by the git merge or the git rebase command depending on your configuration

```
Git Changes -> Pull Icon
```

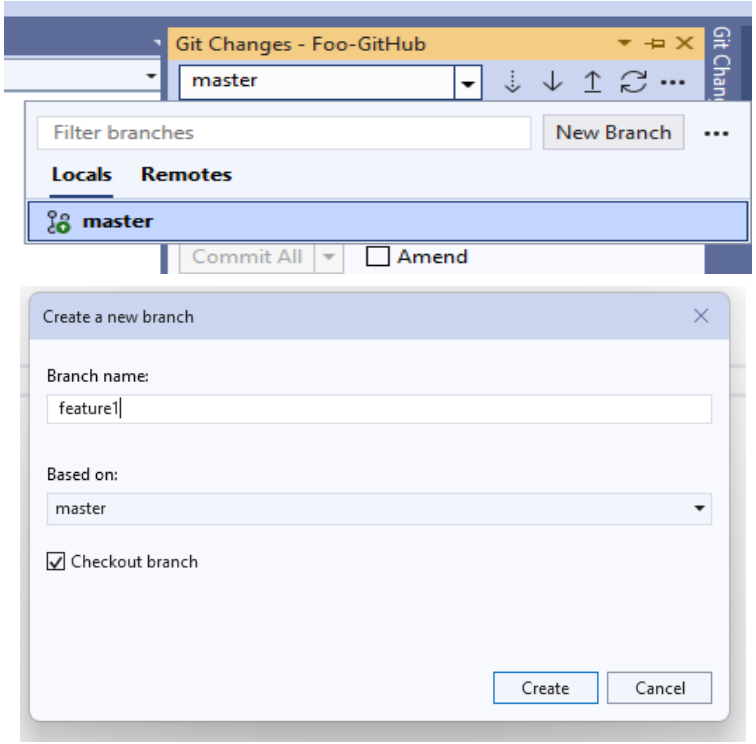


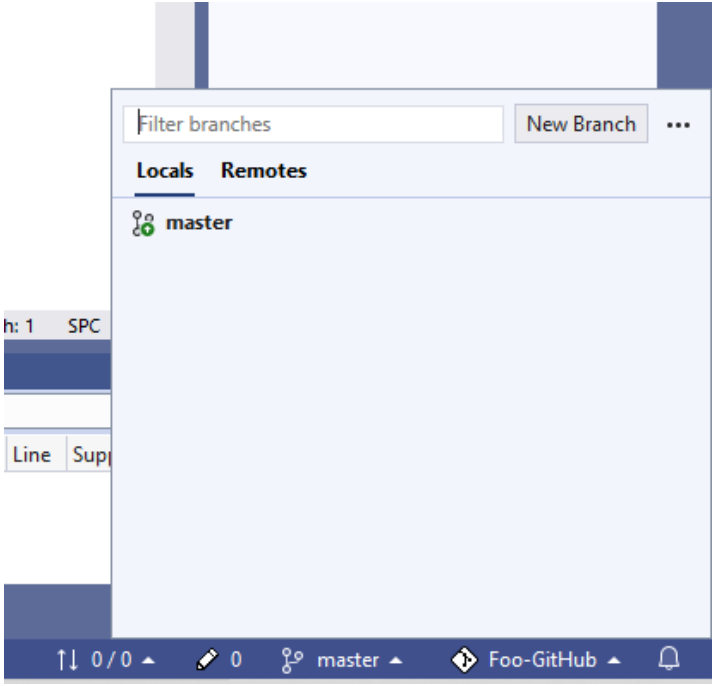
# Using Branches

# What is Branch

Git allows you to create branches. Branches are named pointers to commits. You can work on different branches independently from each other. The default branch is most often called master.

To list all branch
**Git Changes -> Click on Dropdown**(master)  Or  **Click on master branch logo at bottom right**
**From here we can see branches of: Local & Remote. To change branch click on it.**
**Click 'New Branch' to create a new branch.  Right click -> Rename to rename the branch.**



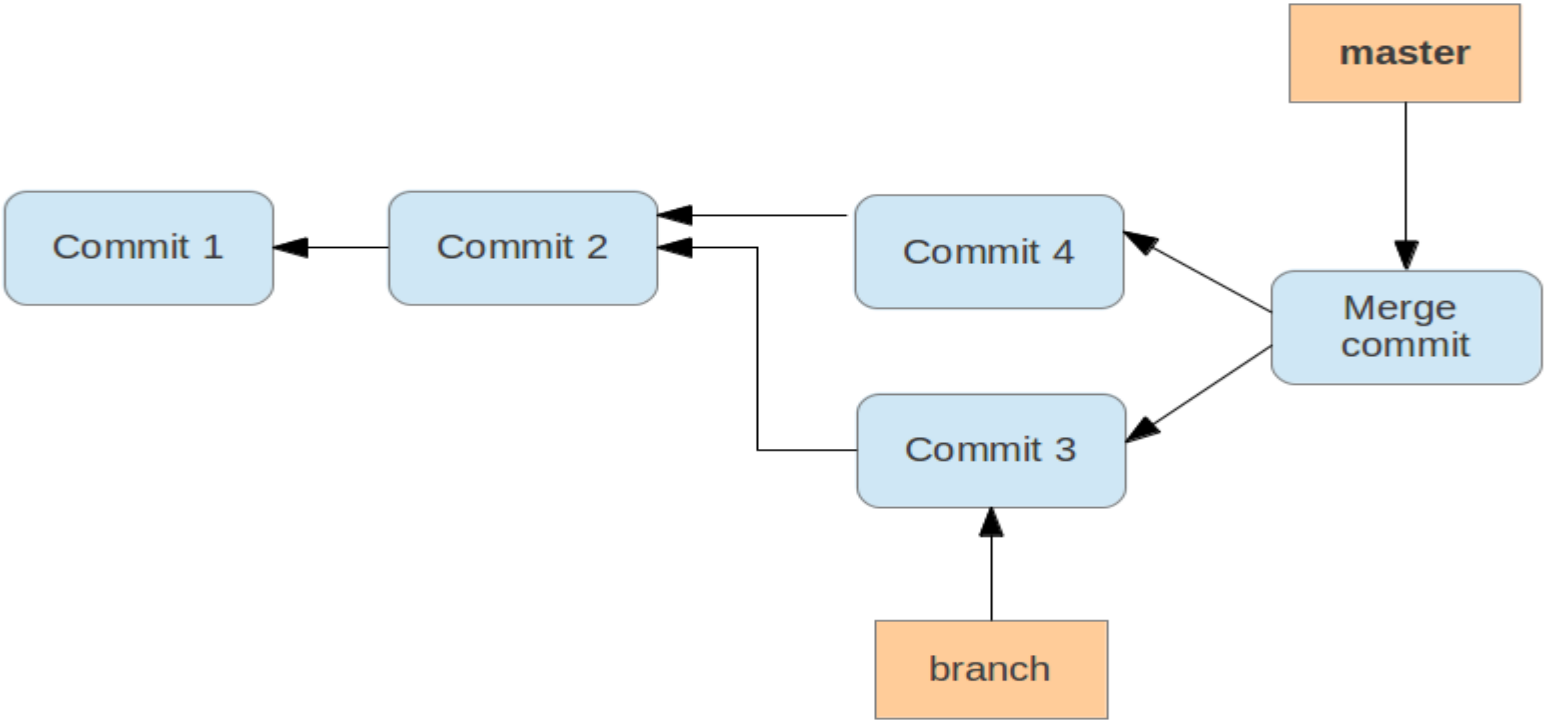
# Push changes of a branch to a remote repository

You can push the changes in a branch to a remote repository by specifying the target branch. This creates the target branch in the remote repository if it does not yet exist.
If you do not specify the remote repository, **the origin is used as default**

**Git Changes -> Push**

**It will create the same Branch in the remote and then update the code there**

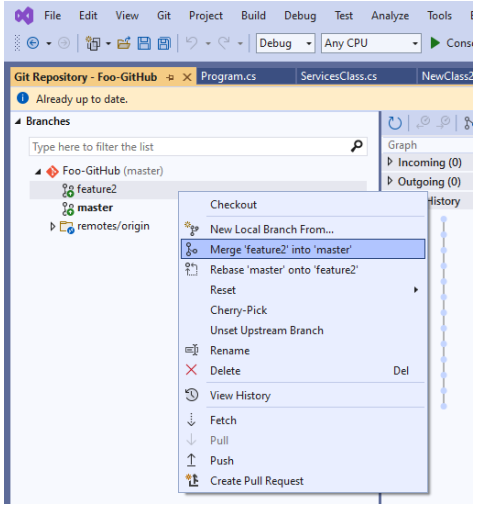
# Merging


master
Commit 1
Commit 2
Commit 4
Merge commit
Commit 3
branch

# Merge two branch

The git merge command performs a merge. You can merge changes from one branch to the current active one via the following command

```
Be in the Branch master (default)

Go to Git changes -> Manage Branches -> Right Click on master(the branch which you want to merge) -> Merge
```



# Conflict  during a merge operation

A conflict during a merge operation occurs if two commits from different branches have modified the same content and Git cannot automatically determine how both changes should be combined when merging these branches.

```
For this, there is the --theirs and the --ours options on the git checkout command. The first option keeps
the version of the file that you merged in, and the second option keeps the version before the merge
operation was started.

If your last file changes are final
$ git checkout --ours about.html
$ git add about.html

If others changes are final
$ git checkout --theirs about.html
$ git add about.html
```

Review the conflict in the file and Solve conflict

```
Git marks the conflicts in the affected files. one file has a conflict and the file looks like the following
listing.

<<<<<<< HEAD
Change in the first repository
=======
Change in the second repository
>>>>>>> b29196692f5ebfd10d8a9ca1911c8b08127c85f8


Select and solve conflict by selecting final version on changes
$ git checkout --theirs about.html
$ git add about.html

# add the modified file
$ git add about.html

# creates the merge commit
$ git commit -m "Merge changes"

# push the changes to the remote repository
$ git push
```

Thank You