

# Git And GitHub

---

SEPT 2018



## **Course Overview**

Git hub is the worlds favorite social coding platform and used by millions of developers around the world

A lot of developers use GitHub in combination with Git as their source management system, which is the main goal of GitHub

---

## **Objective:**

In this course we are going to see how to use Git and GitHub efficiently

We will how to use GitHub repositories, branching forking, project management and much more.

Understanding how to use GitHub for your projects


## **Prerequisites**

No prior knowledge required


## **Main topics to be covered :**

- Learn Git and its most commonly used commands using command line interface
  - Understanding GitHub repositories
  - How to create branches and forks
  - GitHub flow using pull requests
  - GitHub for the organization
-

# Brief History of Version Control

- The very first version control systems were developed in the early 70s and operated on a single file and had no networking support.
  - They operated on a single file, with multiple versions of that file, but there was no correspondents between different files within a repository.
  - This led to the obvious innovation of having a multi-file system or the second generation and this is simplified by centralized version control systems such as CVS, Visual SourceSafe, Subversion, Team Foundation Server, and Perforce.
  - All of these are multi-file centralized systems so we could check out into a working copy on the local system all the files necessary for particular version of a repository.
- 

# Brief History of Version Control

- Along came the third generation which are the distributed version control systems such as Git, Mercurial, Bazaar, and BitKeeper.
  - These work on changesets and these changes sets can be shift around and both clients and servers can have the entire repository present which allows to do some interesting things.
  - So we can see this gradual evolution of going from single file to multi-file to changesets.
  - Going from no networking such as centralized to a distributed and all the additional capabilities that get added with these new generations, a version control systems.
- 

# Brief History of Version Control

- **First Generation**
    - Single-file
    - No networking
    - e.g. SCCS, RCS
  - **Second Generation**
    - Multi-file
    - Centralized
    - e.g. CVS, VSS, SVN, TFS, Perforce
  - **Third Generation**
    - Changesets
    - Distributed
    - e.g. Git, Hg, Bazaar, BitKeeper
-

# Why Source Version Control

Backup/Archive : Source control is a type of backup. It is not just a normal backup that's difficult to access or restore

---

Versioning/History : It's an ongoing backup of every version of the source that is been created. Version control creates history or a trail of changes.

Undo changes and Comparing : Source control allows to undo changes or restore from the previous state, this helps in comparing various revisions of the code.

Collaboration/Teamwork : In the process of moving the source from the local system to a shared server, the source control can allow collaboration and teamwork on the project

Isolation of changes: Source control supports the concept of branching to make changes in isolation.

Code review : Source control provides excellent tool for reviewing code for collaborative learning



# Who needs source control?

- Software developers/engineers/programmers
  - Source code (Java, C++, Objective-C, Ruby, etc)
  - Models (UML, ERB)
  - SQL, configuration, text files
- Freelancers
  - (see above)
- Web Designers
  - Mockups, Web site assets (pages, images)
- Graphic Artists
  - Original art, vector graphics, Photoshop files
- Share code / Open source



# Source Control Options

- Two Main Types
  - Centralized
  - Decentralized / Distributed
- Centralized
  - Free: Subversion, CVS
  - Commercial: ClearCase, Perforce, Team Foundation Server (TFS)
  - Requires connection to central server for most operations
- Distributed
  - Mercurial (Hg)
  - Git
  - Most operations are local
  - Central server not required

# What is Git?

Git is a Version Control System (VCS) at a very basic level


Git will track the changes and allow to commit changes to a source control repository.

---

These systems are designed to facilitate collaboration among a group of developers working in the same cloud base

Git is a modern distributed version control system and this makes Git a bit different from most traditional version control systems.

As a distributed system, Git focuses a lot on branching and merging and it also has two additional concepts called pushing and pulling, which are distinct from the committing and updating.



---

## What Is Git?



Widely adopted source control system



Distributed



Free and open-source, created by Linus Torvalds

## Advantages of Git

- **Git is fast** : Git is really fast and scalable. Since it is distributed Git can also work with local history which makes it much faster than working with remote server
- **Disconnected** : Since Git is a distributed source management system one can work with it disconnected. Since Git is distributed, every developer has a full local copy of the complete repository. All the work is done in the local repository. Only after all the work is complete we can synchronize with the remote server.
- **Powerful But Easy** : Git is very powerful yet very easy to use. Users coming from other source control systems usually have to make a paradigm shift. Once the basic commands are learnt, one can master git within no time.
- **Branching**: Using branching is another benefit of Git. The typical workflow in Git is based on Branching. Its easy to create, merge and delete branches
- **Pull Requests** : Its not really a Git feature. Using the Pull requests the developer can ask for every view and a merge backend to understand the changes he or she has performed.

## Advantages of Using Git



A diagram illustrating the advantages of using Git. It features a central light gray rectangular area containing five colored boxes. The boxes are arranged in two rows: the top row has three boxes ('Fast', 'Disconnected', 'Powerful but easy') and the bottom row has two boxes ('Branching', 'Pull requests'). A small purple square is positioned to the left of the top row of boxes. A solid blue horizontal bar spans the bottom of the slide.

Fast

Disconnected

Powerful but easy

Branching

Pull requests

# What is Git?

Coming from the open source community, it was very important that Git be very fast by using local commits.

---

It needed to scale from very small to very large software projects, and with the number of developers involved in some of these large projects, it needed to be distributed and it was important that the system be very parallelized, and overall, it's really a system that's designed around creating collaboration in situations where collaboration would otherwise be a great challenge.

So that is why we use Git.



# What is Git?

- Distributed source control system
  - Not required to be decentralized
- Massively scales
- Open Source
- Developed for Linux project requirements
- Most operations are local
- Very fast
- Active community
- Most popular DVCS, VCS

## What is GitHub?

- GitHub is a Git hosting site.
  - GitHub provides a hosting service for Git repositories.
  - GitHub is used by many developers for personal projects and as well as by companies for small or large projects.
  - GitHub runs on top of Git.
  - GitHub is free, however the free plan requires that the code is open source.
  - One has to pay for private repositories
-



## What is Git?

Distributed Version  
Control System  
("DVCS")

Branch

Merge

Commit

Push

Pull

Very Fast

Scalable

Distributed

Parallel

## History of Git

Git is a relatively modern version control system.

---

It was created in 2005 after a series of events made its creation necessary.

For a few years prior to the start of the Git project, the Linux kernel was using a proprietary distributed version control system called BitKeeper.

When the relationship between BitKeeper and the Linux Kernel Development Team fell apart, the company who owned BitKeeper no longer offered a free-to-use license.

It was at this moment that the Linux kernel required a new distributed version control system to replace the old one.

This requirement is why Git was created.


# History of Git

Since 2005, Git has been growing, changing, and increasing in popularity.

An entire ecosystem has formed to support the needs of Git's users through hosted solutions, productivity tools, show integrations, and much more.

## CVC versus DVC

---

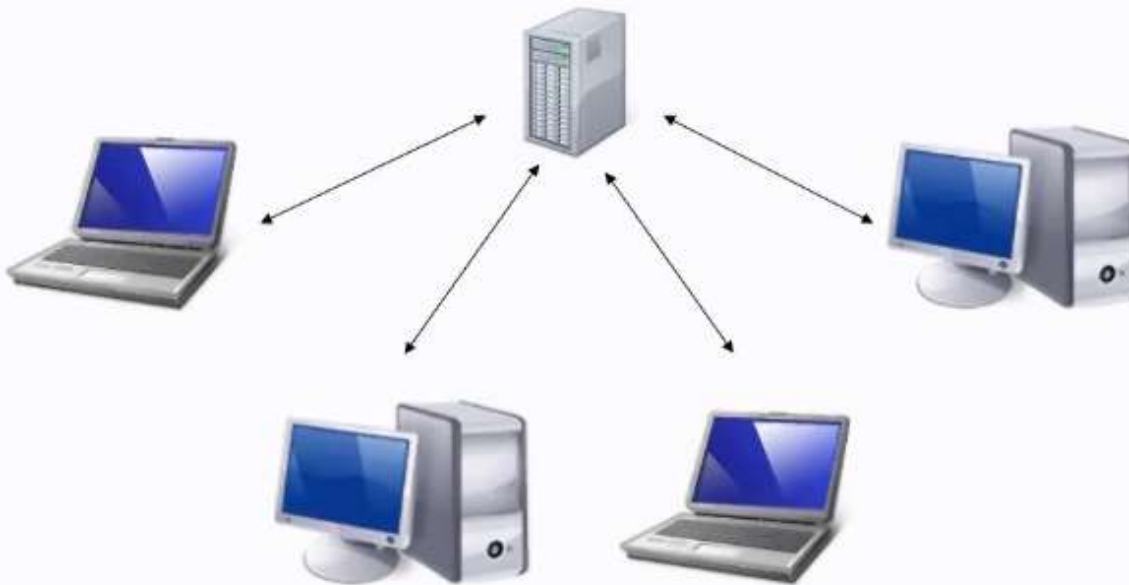
- CVC : Centralized Version Control
  - DVC : Distributed Version Control
- 
- A solid blue horizontal bar spanning the width of the slide, located at the bottom.

# CVC versus DVC

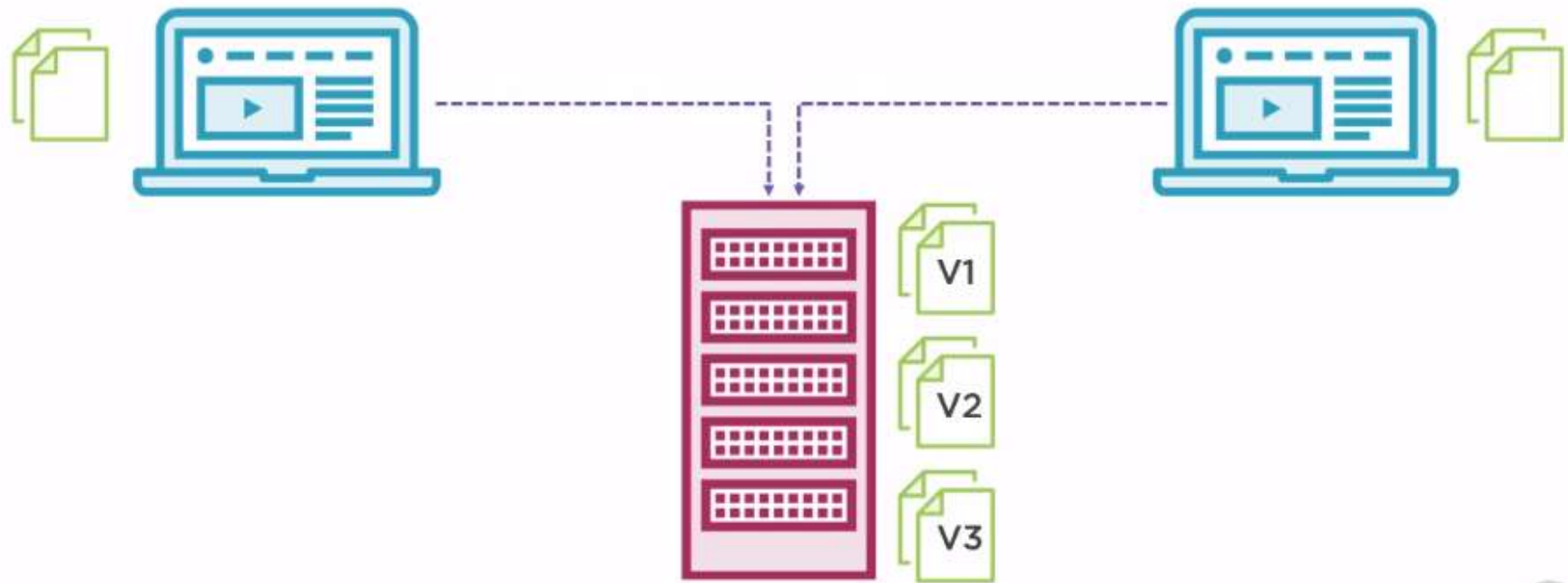
---

- So before we discuss the distributed nature of Git, we will take a look at the centralized structure
- Examples of CVC are TFS CFS etc
- In CVC each developer works on his or her own computer and makes commits to a central server that is managing all of the code.
- Notice in the illustration (next slide) that each developer receives updates directly from the server and sends updates directly to the server.
- This is a simplification of the process, but it illustrates how most developers work with source control today.


## Centralized Version Control



# Centralized Source Code Management



## How Centralized Source management system works ?

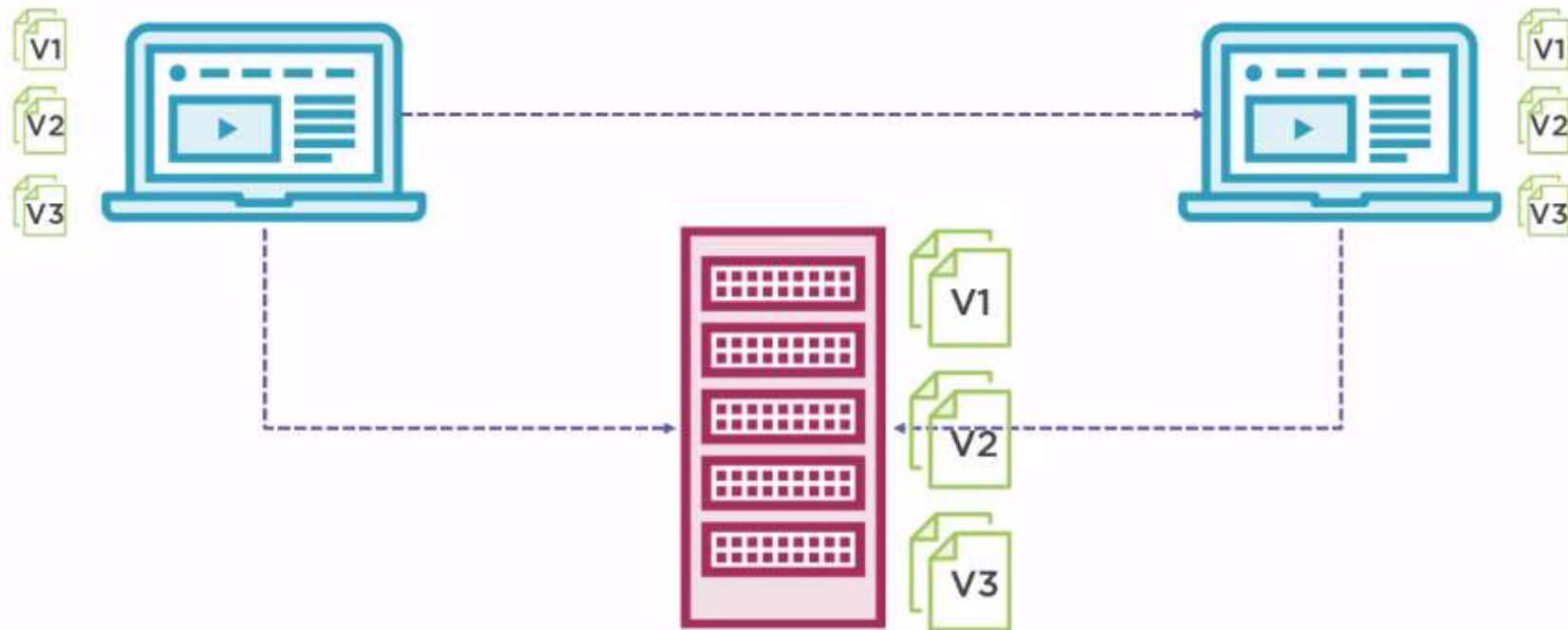
- In centralized source code management system we have a central copy of the source on the server.
- 
- Developers working on their machines will make change to code and that code will be committed to this central store.
  - Other developers can pull down this chain and their files will be updated as well.
  - In centralized system developers work with **changed Sets**.
  - Change Set is the number of changes that we treated as a whole.
- 
- A solid blue horizontal bar spanning the width of the slide, located at the bottom.




## Distributed Version Control

- In DVC the developer will clone the entire repository and therefore get the entire repository on their own machine.
- Thus in DVC a central store is not required or typically there could be one.
- In our case the central store will be GitHub
- Since every developer has a full copy of the entire repository including history we can work offline in DVC system.
- We can create local branches and only when we are ready to share our work with others we can send it to the central repository from where it can be distributed again.
- The process of sending information back to the repository is known as pushing.
- Once the new code is available, all the other developers can pull that change back to their own machine/s.
- Of course pushing and pulling requires to be connected, but otherwise most of the operations can be done entirely in a disconnected way.

# Distributed Version Control




## How Centralized Source management system works ?

- In centralized source code management system we have a central copy of the source on the server.
- 
- Developers working on their machines will make change to code and that code will be committed to this central store.
  - Other developers can pull down this chain and their files will be updated as well.
  - In centralized system developers work with **changed Sets**.
  - Change Set is the number of changes that we treated as a whole.
- 
- A solid blue horizontal bar spanning the width of the slide, located at the bottom.

# CVC versus DVC

- The first thing you need to know about a distributed version control system is how the distribution works.
- In this scenario, everyone has their own personal source control repository.
- In this illustration, that repository is represented by the folder that's displayed next to the laptop.
- Personal source control repository means that the commits are local.
- Each time we commit, we're storing a new version of the code in the local repository.
- This makes it easy to commit frequently since we're worrying less about the impact on other developers.
- We can instead focus on making sure that you commit as often as possible.
- Once we've have the code to the point where it's ready for others to consume, we can push those changes out for other people to access.
- This is sort of like the committing that you did in the centralized version control system in that this is the point where we take the code that's on our machine and send it up for others to be able to consume.


## CVC versus DVC

- Now that we have the basic concept of how distributed version control works, let us show the interactions of that distribution.
- 
- The first thing we'll notice is that there are a lot more options than we had before.
  - In fact, any of the repositories shown in this diagram can push or pull from any other repository.
  - Most teams and most workflows still focus primarily on the traditional structure.
  - Let's use an example.
- 

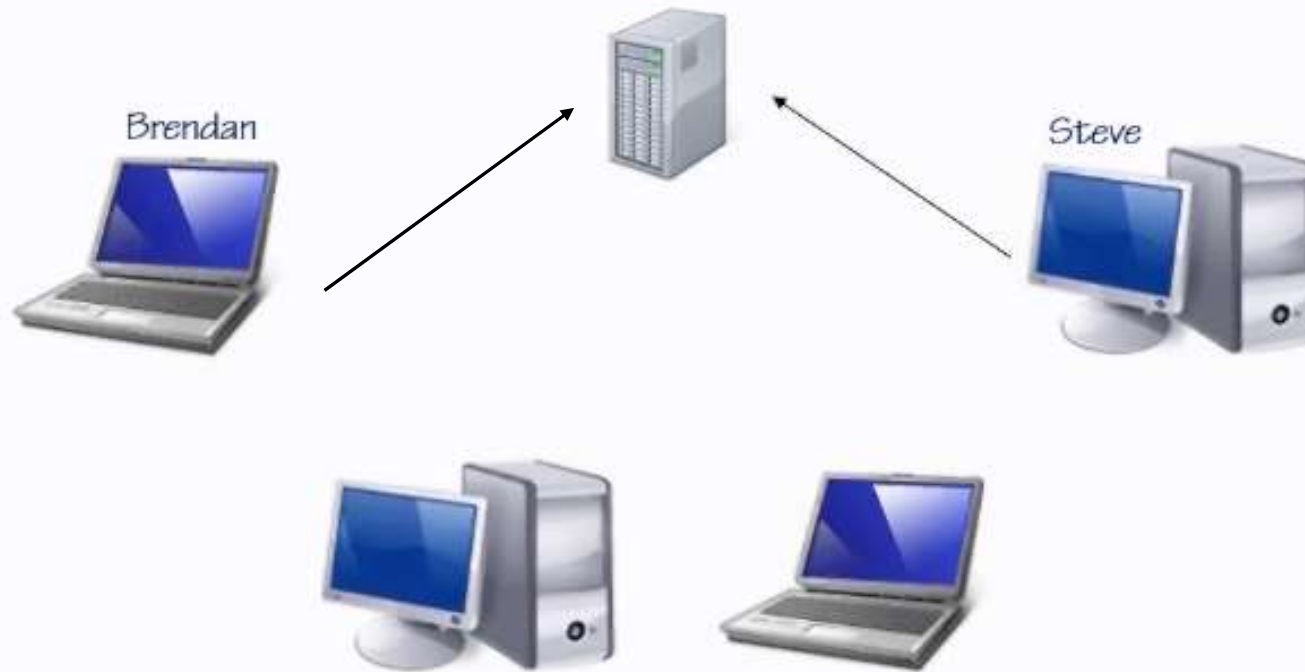
## CVC vs DVC

- We'll look at two developers, Brendan and Steve.
- These two are working together on a feature, and Steve's code is depending on some of the code that Brendan is working on.
- Since Brendan is not done with his feature, he has only been committing locally.
- Steve, however, would like to put some effort into his code and see how well it's going to integrate with Brendan's code.
- So Steve pulls the code from Brendan's repository instead of the central one. When Brendan finishes his code, he pushes it to the central repository.
- Then, Steve is able to pull the final version of Brendan's code from the central repository.
- Since he was already using the code, he is able to finish his task, commit locally, and then he can push his changes back to the central repository.

## CVC vs DVC

- When Steve does the merge, he will not have complex merging to deal with like he would have done if he'd done this with a centralized versioning system that wasn't keeping track very closely of change sets.
  - The reason for this is that Steve had Brendan's changes already when he was writing his code, so the server knows that the code that Steve was working with is the same code that Brendan pushed, and not just code affecting the same files.
  - The unique ID of Brendan's change set allowed all this to happen.
- 

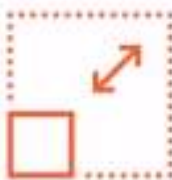
## Distributed Version Control







**Committed**



**Modified**



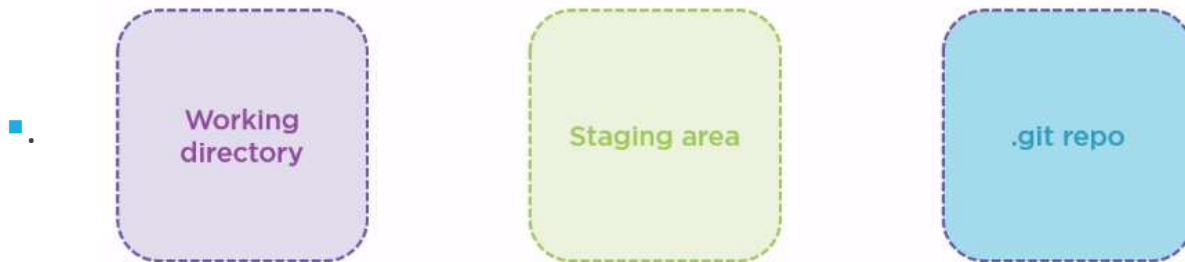
**Staged**

### Three states of Git

- The 3 states in which the files can be in are
  - Committed
  - Modified
  - Staged
- In Committed state the data is stored in a local database.
- In Modified state the file has been changed but it hasn't been committed to the local database yet.
- Finally, Staged means, the modified file is marked to be part of the next commit snapshot.
- All the above states are still local.

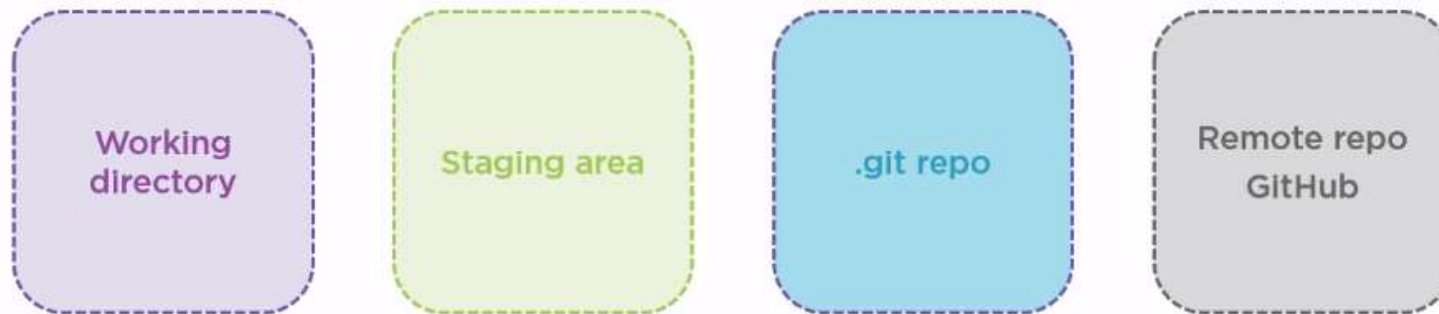
## The 3 Areas of Git

- Since files can be in 3 states, there are also 3 areas where the files can live.
- **Working Directory**: In this area the content will be created, edited and deleted typically for the existing files.
- These files will be extracted from the local Git database.
- **Staging Area**: This area can be seen as the area in which the changes from the local directory will be staged before they will be committed.
- Basically when the files are in the staging area they are waiting to be committed.
- In staging area the files will probably be part of the next commit.
- **.git Repo** : When the commit is performed the changes will be stored in the local git repository. This is a directory entirely managed by git itself.
- This area also gets created when we clone the source from another computer or from GitHub



## Adding a Remote

- Although in a distributed management system its not required to have a remote, many of the files will have a central place i.e. basically the master.
- When we have made a local commit to the local repository, we can send these to a central location, and that in our case will be GitHub.
- So GitHub can be the fourth area where file can be stored.
- Although we are adding a remote, we still have the benefit of the distributed source management system.



# About Git

## About Git

---

- Created by Linus Torvalds, who also created Linux
- Prompted by Linux-BitKeeper separation
- Started in 2005
- Written in Perl and C
- Runs on Linux, OS X, Windows, and many other operating systems
- Design goals
  - Speed
  - Simplicity
  - Strong branch/merge support
  - Distributed
  - Scales well for large projects

# Installing Git on various OS

## Installing Git

- **Windows**
  - msysGit (<http://msysgit.github.com>)
- **Mac OSX**
  - brew install git
  - DMG (<http://git-scm.com/download/mac>)
- **Linux**
  - apt-get install git-core (*Debian/Ubuntu distros*)
  - yum install git-core (*Fedora distros*)
  - For other distros, check your package manager

# Installing Git (on Windows)

Firefox - Git for Windows

msysgit.github.com

## Welcome to the home page of Git for Windows


Git is a powerful version control system aiming to be the fastest decentralized source code management tool on this planet.

Having its root in the Linux development community, Git used to be quite dependent on POSIX features usually only provided by Unix-style Operating Systems. Thanks to the efforts of [a few contributors](#), this project succeeded in providing an almost feature-complete fork of Git on Windows. Being solely driven by volunteers in their spare time, it is nevertheless quite stable.


In order to develop Git for Windows, these volunteers rely on a build environment that is based on the [MSys/MinGW](#) project. To sort out the confusion revolving around the naming scheme, let's have a look at this table:

### Git for Windows

Logo:




### msysGit



Audience:	Pure users of Git	Testers, developers, custom installer maintainers
Support:	Unfortunately none	Questions regarding building Git for Windows should be directed to the <a href="#">msysGit mailing list</a> .

## Installing Git



msysgit

Git for Windows

[Project Home](#)
[Downloads](#)
[Wiki](#)
[Source](#)

Filename ▼	Summary + Labels ▼	Uploaded ▼	ReleaseDate ▼	Size ▼	DownloadCount ▼	...
<a href="#">Git-1.7.10-preview20120409.exe</a>	Full installer for official Git for Windows 1.7.10 <span>Beta</span>	Apr 9		14.6 MB	151744	
<a href="#">Git-1.7.9-preview20120201.exe</a>	Full installer for official Git for Windows 1.7.9 <span>Beta</span>	Feb 1	Feb 1	14.4 MB	284814	
<a href="#">Git-1.7.8-preview20111206.exe</a>	Full installer for official Git for Windows 1.7.8 <span>Beta</span>	Dec 6	Dec 6	13.6 MB	182574	
<a href="#">Git-1.7.7.1-preview20111027.exe</a>	Full installer for official Git for Windows 1.7.7.1 <span>Beta</span>	Oct 2011	Oct 2011	13.4 MB	135362	
<a href="#">Git-1.7.7-preview20111014.exe</a>	Full installer for official Git for Windows 1.7.7 <span>Beta</span>	Oct 2011	Oct 2011	13.4 MB	47737	
<a href="#">Git-1.7.6-preview20110708.exe</a>	Full installer for official Git for Windows 1.7.6 <span>Beta</span>	Jul 2011	Jul 2011	13.2 MB	269868	

## Working With Git

- We can work with Git and GitHub in several ways
- 1. Using the command line : This is the most commonly used approach to work with Git and GitHub
- 2. Using the UI : The following are the various GUI tools that can be used

### Using a UI for Git

GitHub Desktop

Sourcetree

GitKraken

Git Extensions



## Getting the machine ready for GitHub

- Git is widely supported on all OS – Linux, Windows, Mac
  - **What is required**
    - Download Git from <https://git-scm.com/downloads>
    - An editor (Notepad++, or any editor of your choice)
    - GitHub account
  - Setting up a Windows environment for Git
    - Go to <https://git-scm.com/downloads>
    - Git installs the GitBash – It's a command line interface that is easy to work with
-

# The Basics – Overview

## The Basics

- Getting Started / New Repo
- Information / Support
- Basic Workflow
- File Operations
- Exclude Unwanted / Transient Files
- Undo Mistakes

---

```
$ git
```

◀ It all starts with “git”...

```
$ git config
```

◀ Configure the tooling

```
$ git init
```

◀ Initialize a git local repo

```
$ git clone
```

◀ Download a project from remote

```
$ git add
```

◀ Prepare a file (to staging)

```
$ git commit
```


◀ Commit a file to the repo



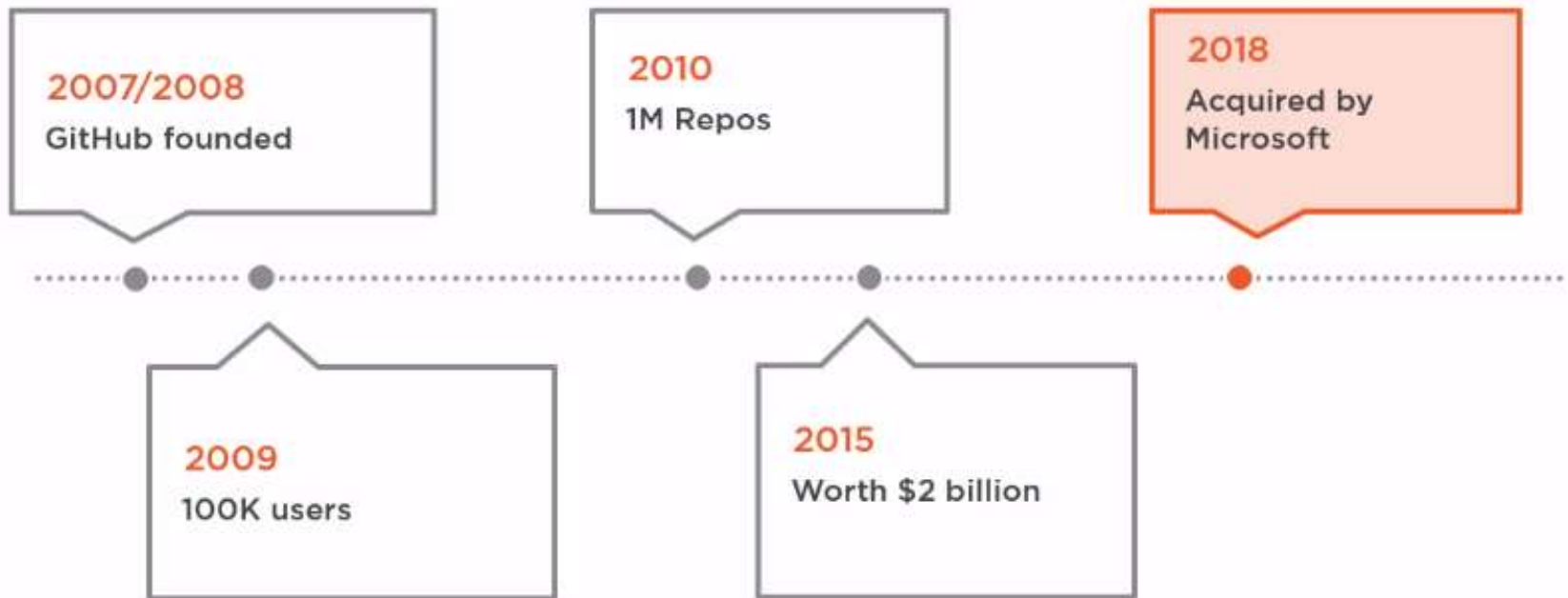
## **Module 2: Getting started with GitHub:**

- Introduction to GitHub
  - Creating a GitHub Account
  - Brief look at how GitHub works with repositories
  - Connecting the local machine to GitHub
  - Using GitHub's search
-

## **What is GitHub?**

- GitHub is a web based hosting service for Git
  - It is used by software developers to put their code in
  - GitHub extends what we can do with Git
  - GitHub offers the features offered by Git such as DVC and Source Code Management
  - By the mid of this year GitHub has close to 20 million users and about 60 million repositories
  - GitHub not only offers features for storing code but also to manage entire project
  - GitHub is praised for its focus on social aspect of coding
  - GitHub is also used for collaborating the projects.
  - GitHub has been acquired by Microsoft in 2018
- 

# GitHub's History



## GitHub's Main Features

Code. Code. Code.

Pull requests

Issues and Project  
management

Documentation


Notifications

Teams

Gists

Integrations

- **GitHub and Its Main Features**

- GitHub is a web-based hosting service for Git.
  - It's mostly used by software developers to put their code in.
  - GitHub therefore basically extends what we can do with Git.
  - GitHub offers the features offered by Git, such as distributed version control and source code management.
  - GitHub has close to 30 million users and close to 60 million repositories.
  - GitHub is praised because of its features around the social aspects of coding.
  - The cornerstone of GitHub is code, and code, and more code.
  - All that code lives in repositories and repositories are a very important concept.
- 



- **GitHub Features**

- **Pull Requests**: The core concept of GitHub is the concept of pull requests.
- Using pull requests, part of the GitHub flow, developers can request that their changes are pulled into another branch.
- Typically, this is then combined with a code review, which can be conducted directly from the GitHub interface.
- This is very relevant for all the social aspects of GitHub, as well as where just anyone can propose changes to existing projects.
- GitHub is a full-fledged **project management tool**.
- We can work with issues to register work that needs to be done, such as a bug that needs to be fixed.
- **Documentation** : Another thing that GitHub enables is documentation.
- There are several ways that you can use GitHub to document your project, including GitHub pages and a wiki.
- The social aspect, as well as the project management features, are emphasized with the fact that GitHub contains a very nice system for notifications. We can register to be notified about issues being assigned to us.

- GitHub Features

- **Teams**: GitHub also supports the concept of teams.

- Using teams, we can organize or and get a fine-grained way to manage who in our team can do what. \_\_\_\_\_

- Indeed, GitHub comes with a role-based system that enables us to assign permissions in our projects to certain team members.

- **Gists**: Git also comes with the concept of gists.

- A gist is a simple way to share, typically, a single file or even just a part of a file. I like to see these in the context of sharing snippets.

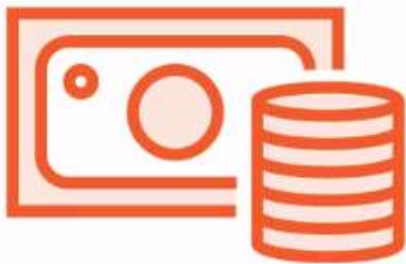
- **Integrations**: GitHub also offers integrations with many other tools.

- For this reason, there's a GitHub marketplace where we can browse through a long list of products that work with Git.

- The people behind Git also have built quite a number of interesting tools themselves, including Atom, which is an editor, the GitHub Desktop client, and quite a few more.

- GitHub offers several plans for us to choose from.

## Different Plans to access GitHub



**Free**

**Paid**

- Developer
- Team

**Business Cloud**

**Enterprise**

Free : For public repositories

Paid: For private repositories both for individual developer and team

Business Cloud : Will give the ability to still run everything on GitHub and host it but get option of single sign on with our own account

Enterprise : Provides a plan to run GitHub on our own service

Some of the common alternatives to GitHub are **GitLab, BitBucket** etc

- Just like Git , GitHub is based on the concept of repositories
  - So what are repositories?
  - Basically repositories can be seen as a folder, where our project resides
  - In the repository all the files that are part of the project are stored.
  - Not only the project files but also the history of the files are stored in the repository
  - GitHub's main way of working is to reuse the repository.
  - When you create a new project, we typically start by creating a new repository.
  - .
-

## ■ The Basic building block of GitHub :Repositories

- It is possible to create a repository through the GitHub interface or through the command line interface
- When a new repository is created by the user, it is initially owned by that user.
- Generally a repository is owned by the organization
- Through GitHub it is possible to manage the permission of the repository to manage different people
- Repositories can be public or private
- Repositories are building block of GitHub
- It is the folder for your project, all files related to the project will reside in that repository
- To work with GitHub you need to create at least one repository
- An individual or organization can own the repository

# Repository Landing Page

GillCleeren / BethanysPieShopMobile

Watch 1 Star 8 Fork 3

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

Xamarin reference architecture, used in Pluralsight course

6 commits

2 branches

1 release

1 contributor

Branch: master New pull request

Create new file Upload files Find file Clone or download

GillCleeren Merge pull request #2 from GillCleeren/dev

Latest commit 1+209e5 on Jun 6

BethanysPieShop.API	Move from VSTS	3 months ago
BethanysPieShop.Mobile	Move from VSTS	3 months ago
.gitignore	Initial commit	3 months ago
BethanysPieShopMobile.sln	Move from VSTS	3 months ago
README.md	Update README.md	3 months ago

README.md

## Bethanys Pie Shop Mobile (Xamarin.Forms)

Xamarin reference architecture, used in Pluralsight course "Building an Enterprise Mobile Application with Xamarin.Forms"

Owner

Repository name



github-2018-zensar ▾

/

Great repository names are short and memorable. Need inspiration? How about **stunning-memory**.

Description (optional)

A demo repository



**Public**

Anyone can see this repository. You choose who can commit.



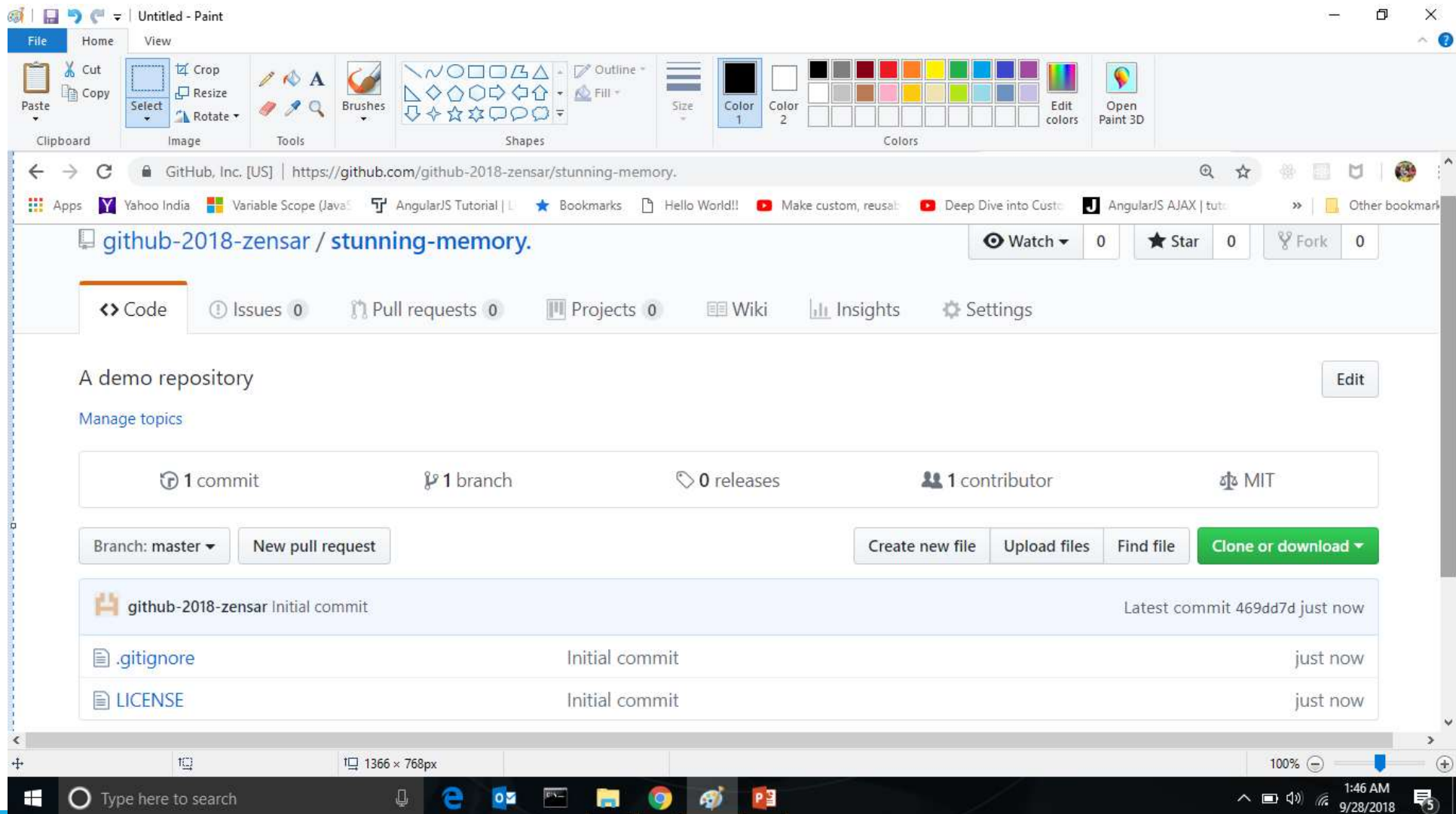
**Private**

You choose who can see and commit to this repository.



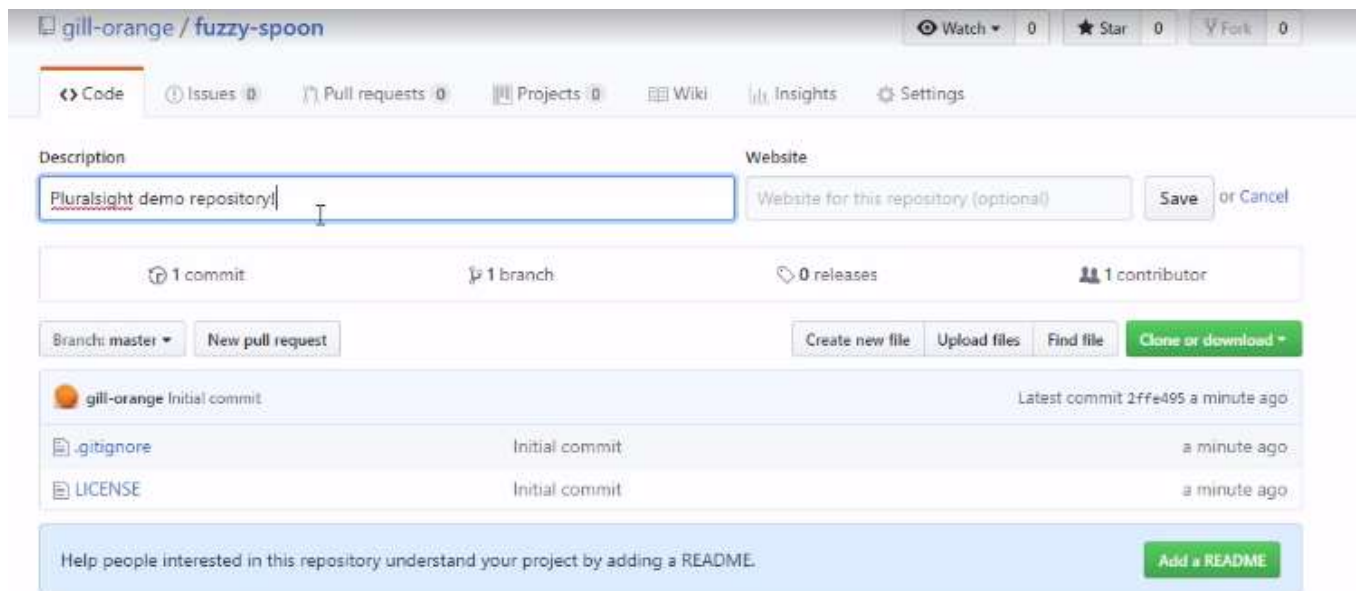
**Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.





## Making Changes




In the above repository make changes in the description and click on the save button.  
Once you save the changes, repository has one commit  
Click on the commit, it will show that it was the initial commit

gill-orange/fuzzy-spoon x

GitHub, Inc. [US] | https://github.com/gill-orange/fuzzy-spoon

Search or jump to... Pull requests Issues Marketplace Explore

Your repository details have been saved.

 gill-orange / fuzzy-spoon Watch 0 Star 0 Fork 0


Code Issues Pull requests Projects Wiki Insights Settings



Pluralsight demo repository! [Edit](#)

[Add topics](#)

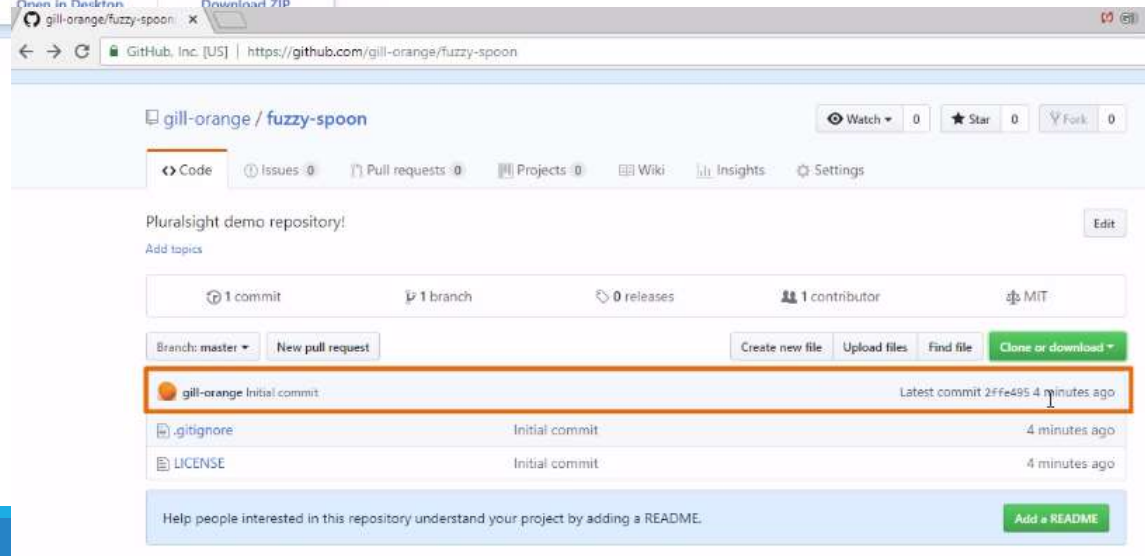
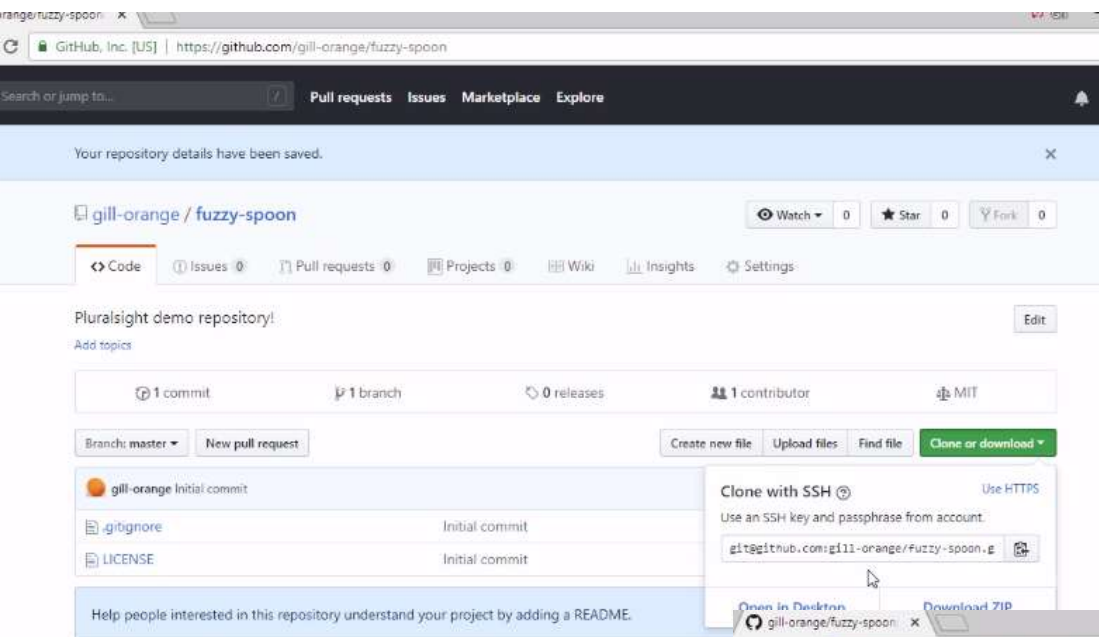
1 commit 1 branch 0 releases 1 contributor MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

 gill-orange Initial commit Latest commit 244e495 3 minutes ago

 .gitignore	Initial commit	3 minutes ago
 LICENSE	Initial commit	3 minutes ago

Help people interested in this repository understand your project by adding a README. [Add a README](#)



## Special Files On GitHub

README.md File

LICENCE

Contributing and Contributors

---

Changelog : All major changes between different versions of the project

Support : Tells us all possible ways to get help with the project

Code\_Of\_Conduct : This contains the guidelines for the rules they have to follow when interacting with the project

**README file is special file known by GitHub**

- Root, .github or docs folder

**Rendered automatically on landing page**

**Typically written in markdown (.md)**

## Sample Markdown syntax

```
# This is an H1
```

# Means rendering H1 tag

---

```
## This is an H2
```

## Means rendering H2 tag

```
* Red
```

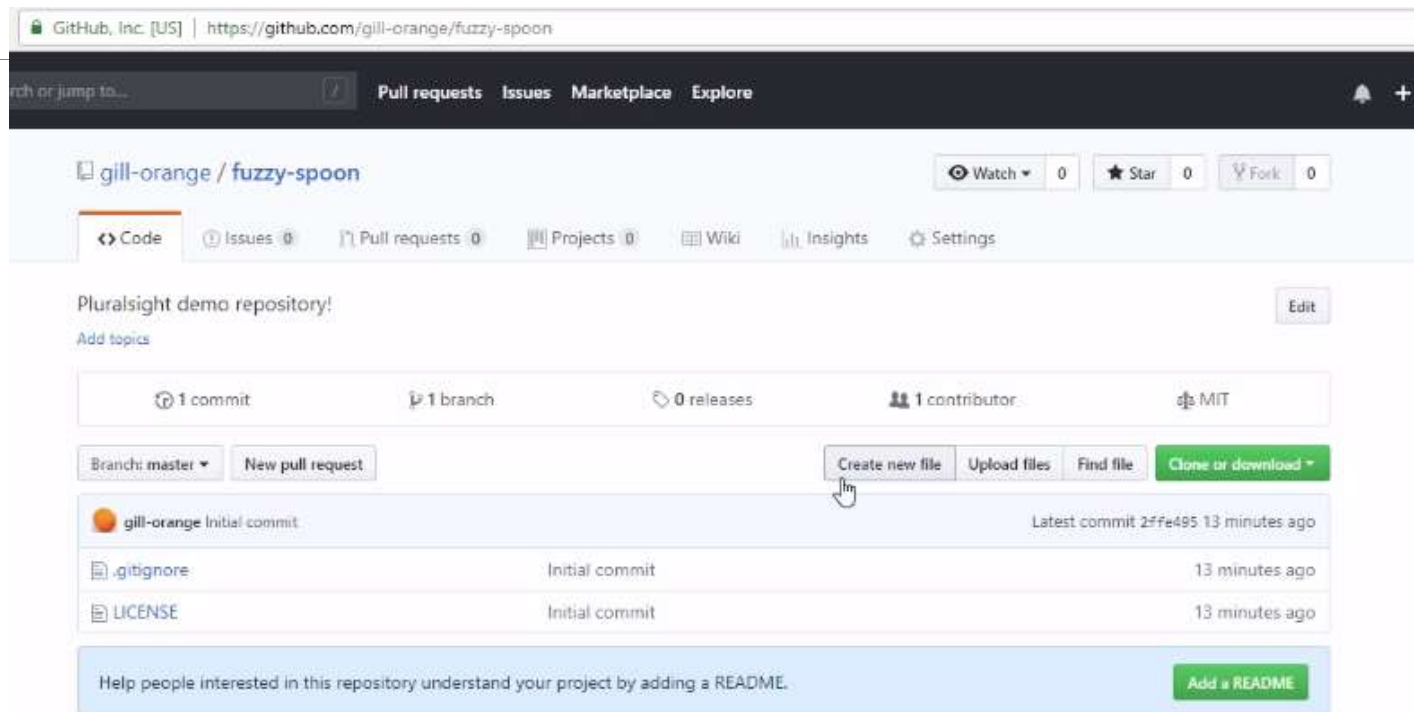
```
* Green
```

```
* Blue
```

\* Means rendering list

Full syntax available at  
<https://daringfireball.net/projects/markdown/syntax>

## Adding README Manually



## Working With GitHub Locally

- Remote repo is the GitHub repository
- First thing you do is perform a clone
- Git clone will create a local copy on your machine of the remote repository for you to work with
- Clone command will also initially create remote tracking branches for all branches in the remote repo
- Clone will also check the default active branch of the remote
- Basically clone will create a copy of the remote repo on the local repo
- Now we can add new files to the local repo.
- Using the add command we can make sure that git starts tracking the files
- Once we are satisfied with the new add, we can commit in the local repo
- Now we push the changes to the remote repo.
- If multiple people are working in the project then there could possibly be a push conflict.
- Before pushing we check if other changes have been made.
- If changes are there we need to fix those changes in the local machine and fix those first.

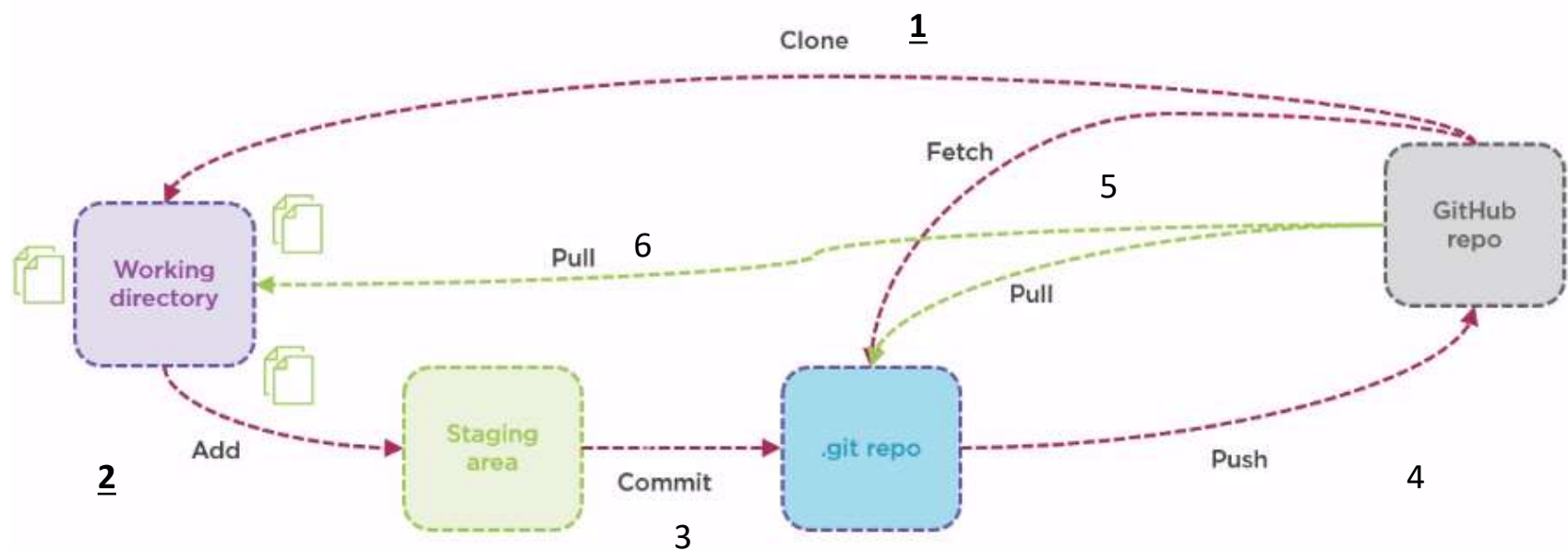
## Working With GitHub locally – continued

- There are two options to fix the changes
  - 1. First you can perform a Git fetch, this will bring down the changes made in the remote repo, which is not present locally as yet.
- 
- Only when merge is done the changes will be merged
  - 2. Alternatively we can perform a git pull
  - Pull is a short cut to both FETCH and MERGE
  - So with PULL, changes will be done to the local repository as well as merge at one go
  - After git merge, we can perform push back to remote repo



## Working With GitHub Locally

# Using Git Commands with GitHub



## Adding And Editing Files On GitHub

- Most of the time we do editing or adding first in the local repo and then push it to the remote
- But if only minor changes are to be done we can directly do it on the remote repo

---

## Adding and Editing Files on GitHub



The screenshot shows a web editor interface for editing a file named `index.html` on GitHub. The interface includes a breadcrumb path `improved-memory / index.html` and a button labeled `or cancel`. Below this, there are two tabs: `Edit file` (active) and `Preview changes`. The `Edit file` tab shows the following HTML code:

```
1 <!DOCTYPE HTML>
2
3 <html>
4   <head>
5     <title>Sample site</title>
6     <meta charset="utf-8" />
7     <meta name="viewport" content="width=device-width, initial-scale=1" />
8     <link rel="stylesheet" href="assets/css/main.css" />
9   </head>
10  <body>
11
12    <!-- Header -->
13    <header id="header">
14      <div class="logo"><a href="#">Welcome to our site</a></div>
15    </header>
16
```

The interface also includes a toolbar with options for `Tabs`, `8` (line numbers), and `No wrap` (line wrapping).

## **Archiving Non-Active Repositories**

- When the project is no longer maintained
- This can be done using the on archive function in GitHub
- We can also update the README file for the users to know about it.
- We can also close issues and pull requests
- Once archived the information in the repository is not lost, but all items , such as issues, become read only.

## **Repository Features**

- So far we have understood that repository is the foundational building block on GitHub
- Everything revolves around the repo
- GitHub provides some extra features to the repo

---
- We can assign TOPICS to the repo
  - Topics are used to classify the project
  - This can help your project to be easily visible when someone is performing a search
- Projects:
  - From the settings of the repo we can work with projects
  - Projects are used to track progress
- In the repo we can see or manage issues
  - This is used to track bugs, todos, feature requests and more.
- Pull Requests
  - With pull requests GitHub makes it easy to collaborate with the code
  - Pull request allow users to suggest changes and allow repo owners to make the changes

## **Repository Features**

- Insights
  - Gives a lot of information about your project including the traffic for your repo pages
- Settings

---

  - With settings we can add multiple settings including the collaborators, default branch and many more
- **Bringing In More Features**
- GitHub is a social coding platform
- In GitHub you can think of two groups for working with the project
- First : Collaborators : These are fixed group of people, these are the core development team working on the project.
- Collaborators have more permissions on the repository, the most important one being commit accessing the main repository.
- That means you as a project owner will trust one or more people to be able to change the code in the project.
- This is usually the QA group

## **Bringing In More Features**

The second group of people are the contributors

These can be anybody outside the core team of collaborators

A contributor is the one who uses the project in his/her daily work and he/she is proposing a change or an improvement.

Contributors are everyone with lower permissions

## Branching, Merging and Pull Requests in GitHub Module

---

**An overview of branches**

**Working with branches on GitHub**

**Pull requests and the GitHub flow**

# Overview Of Branches

## ■ Branching in traditional Source management



Diverging from the main development line

Often very expensive

- Copy of the source directory

## Branching With Git



Much more lightweight

Branching is very fast

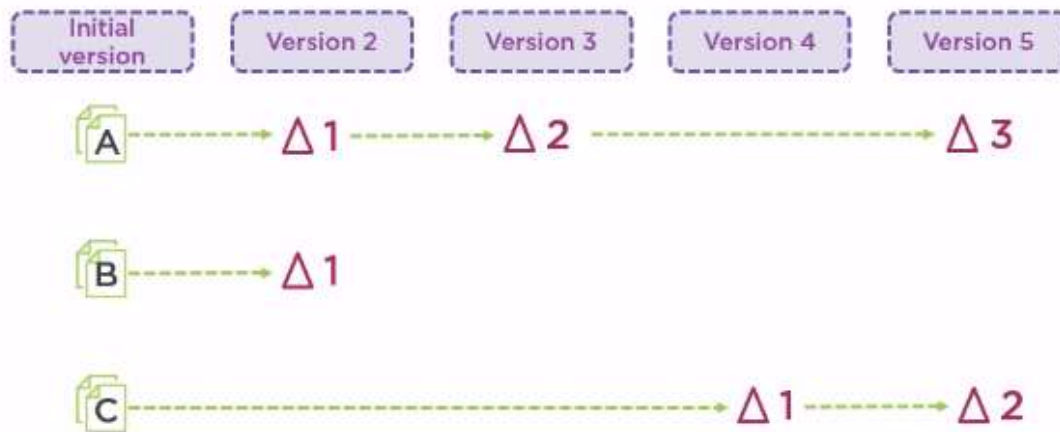
Encouraged to be used

Works because of the way Git works

- Snapshots

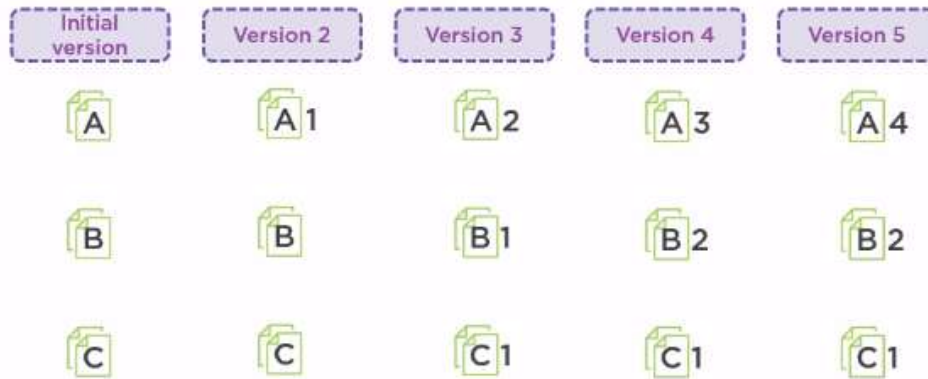


## Traditional Source Management




- In this system we are check in the information.
- These information's are stored as files
- The deltas made to these files are stored and this is what a changed set consists of .
- In the figure you see that there are a couple of files and over time that changes are made to these files.
- The changes are then stored as deltas.
-

## The Concept of Snapshots in Git (and GitHub)




- In Git/GitHub we work with a series of snapshots
- When a commit is performed the snapshot of all files is made and a reference is created.
- But it does not create a copy of each and every file for every commit.
- Instead if the file wasn't changed Git will store just the link to the previous file.
- This is the main difference between Git and other source controls

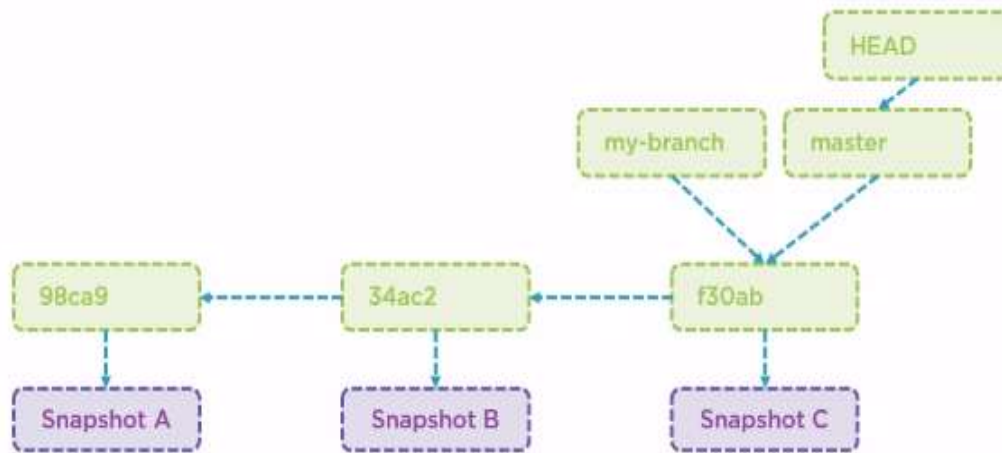
## **Commit Git**

- Every time a commit is created, Git will store a pointer to the snapshots
  - The commit object of which the pointer is a part will also contain info like email commit message etc.
  - In Git a branch is nothing but a pointer to one of these commits.
  - Default branch is the master, and it will always be created.
  - The pointer will point to the last commit made and it moves forward for every commit we create.
  - Hence creating branch is a cheaper operation in git
  - Branches are created for almost all development.
- 

## Create a branch

- When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not.
  - Branching exists to help you manage this workflow.
- 
- When you create a branch in your project, you're creating an environment where you can try out new ideas.
  - Changes you make on a branch don't affect the master branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.
- 
- A BRANCH POINTS TO A SNAPSHOT IN THE HISTORY OF THE PROJECT
- 

## Branching in Git Continued



## Branching Commands for local repo

```
$ git branch [branch-name]
```

---

```
$ git checkout [branch-name]
```

```
$ git push -u [origin] [branch]
```



## **Pull Requests And the GitHub Flow**


- Now that we have seen how to create branches we will see how to use pull requests.
- This is typically called as the “GITHUB FLOW”
- When a new branch is created it will not have any impact on the main branch
- After creating the branch, we need to integrate the changes we have made into another branch perhaps the master branch.
- This can be done by creating the pull request.
- By using the pull request (pr) we are telling the others that we have made changes in a separate branch.
- When we open the pull requests we are announcing the changes, and we are opening for discussions.
- After commit we can merge into another branch or the branch we have created.
- To support this GitHub contains a number of links to create a pull request.
- GitHub also contains an interface for the collaborators to review the pull requests.

## Pull Requests GitHub Flow

# GitHub Support for Pull Requests

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



base: master

compare: title-change

✓ Able to merge. These branches can be automatically merged.

change instructions

Write

Preview

AA B i " < > ↺ ☰ ☷ ☶ @ 📎 ↶

Leave a comment

Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.

📄 Styling with Markdown is supported

Create pull request

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

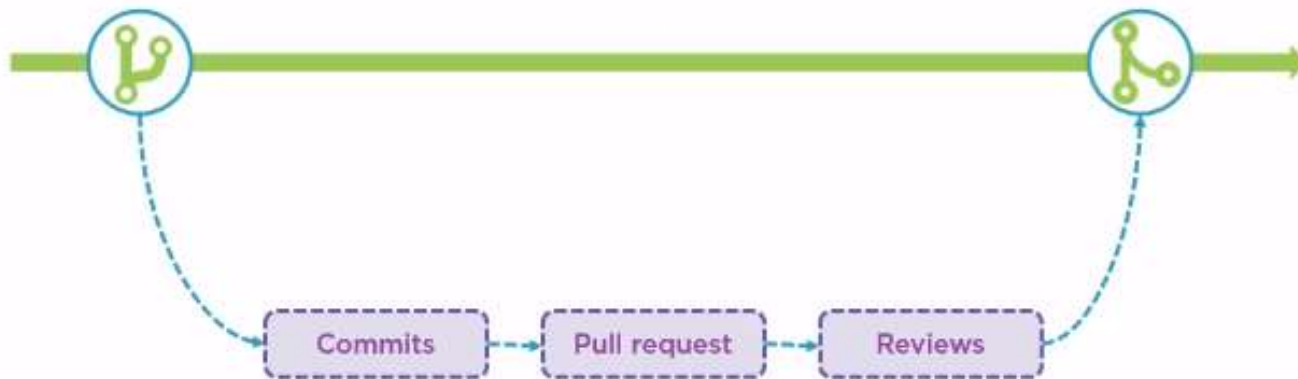
None yet

Milestone

No milestone



# GitHub Flow



## Tags And Releases

We already know that a BRANCH POINTS TO A SNAPSHOT IN THE HISTORY OF THE PROJECT.

---

**A tag points to an important point in history**

- V1.0

**2 types of tags:**

- Lightweight
- Annotated
  - Message, checksum...

**Tags can be added later on an existing commit**

**Command: \$ git tag**

**Command: \$ git tag -a //will create an annotate tag**



## Working with Tags on GitHub



Download the code (zip)



Commits to current



Information about the commit


## Tags And Releases

- We've now seen tags, let's now turn our attention to releases.
- Releases are based on tags, and as mentioned, both are very closely related.
- Probably the most notable difference between tags and releases is that with releases, we can add release notes.
- Through the GitHub interface, we can create releases in Markdown format again, and on that same page we can also add binary files related to this release.
- Releases are GitHub's way of packaging and providing software to your users.
- It basically replaces downloads that we use to provide the software to your users.
- Releases can also be used with attached binaries files.
- Although they are very closely related, we have seen how both of them have a specific goal. But in the end, tags and releases are pointers with extra information to a specific point in the history.

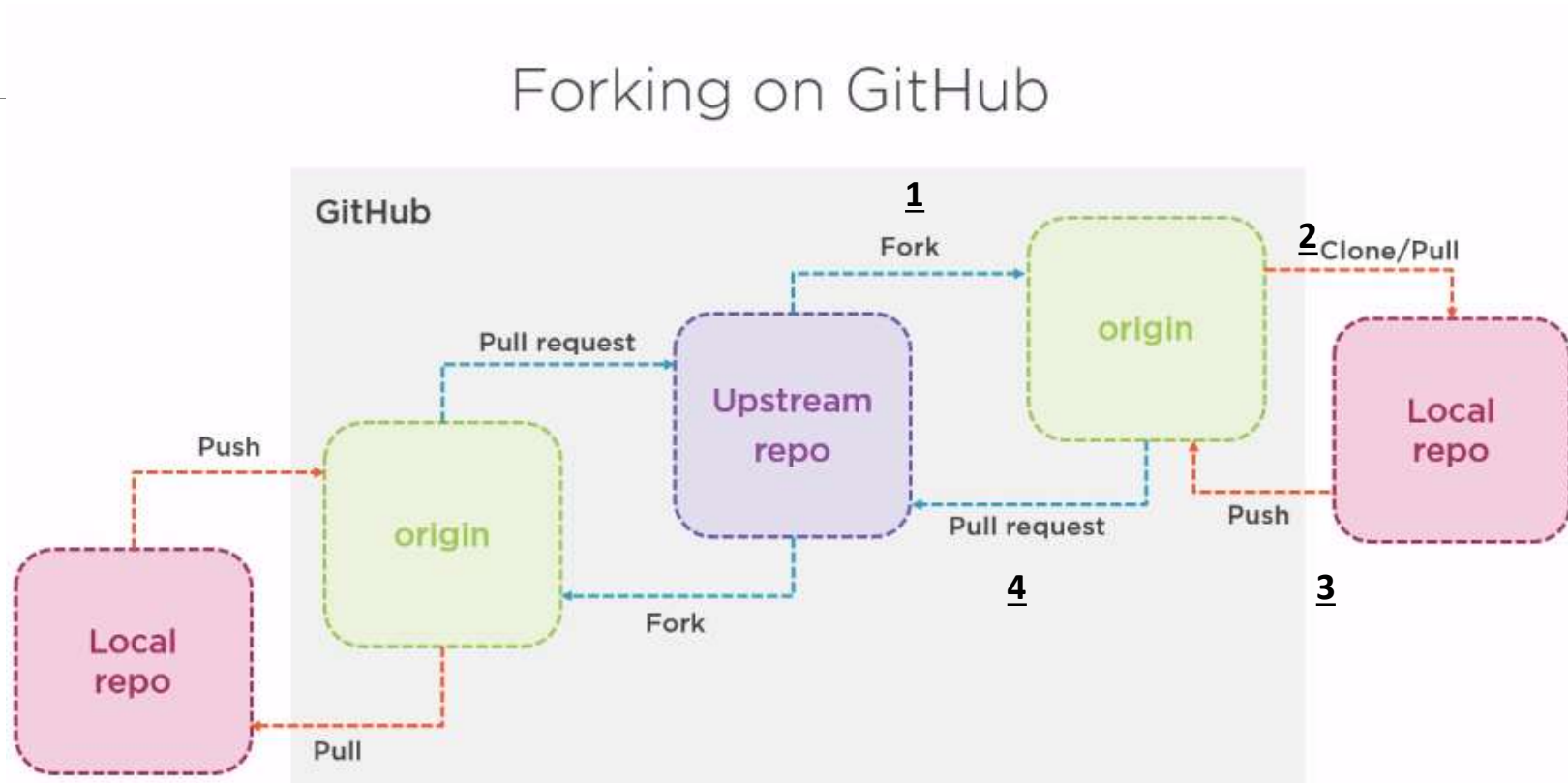
## Social Aspect of GitHub (Working with Forks)

- A fork is a copy of a repository.
- Yes, it's in fact nothing more than a copy of an existing repository.
- We can make changes to that copy without impacting the original repo.
- Now of course being GitHub, it's more than just a plain copy.
- When we create a fork, the copy basically remains linked to the original repo.
- So, if there are any updates in the original repository, we can get these changes into the fork as well.
- Also the opposite is true, If we have created a fork, it's possible from this fork to create a pull request so that our changes can then optionally be merged back into the original repo.
- Forking is possible for anyone.
- Once a fork is created that fork is now your own repository and you can manage it entirely.

## ■ Social Aspect of GitHub (Working with Forks)

- So that means that on the new project, we can bring collaborators, change the repo settings, and so on.
  - Forks are typically created to test ideas.
  - Forks are definitely related to the way of working big branches.
  - Forks are also a GitHub thing, they don't come with Git itself.
  - One major difference between forks and branches is that branches all work on the same repository, whereas forks work on different repositories.
  - In terms of being able to create pull requests to merge your changes back into the upstream repository, things are very similar here.
- 

## Social Aspect of GitHub (Working with Forks)




## ■ Social Aspect of GitHub ( Pull Requests Revisited)

- In the area of forks, we can also work with pull requests.
- As explained, forks are not really different from branches, and therefore we can also use pull requests in the context of forking.


---
- Using a pull request, as a developer, one can indicate that one wants to update other stakeholders about the changes made.
- Since we're now talking about forking, it means we want to let the owners of the upstream repository know that changes have been made and it has to be added to the parent repo.
- we already saw with pull requests, it might be that the changes one makes in the fork will be accepted, and therefore merged in the upstream repository.
- Since pull requests are really a GitHub thing and not a Git thing, it's quite normal that GitHub offers a very nice interface for working with pull requests.
- Although we are now working with forks of a repository, the way that we create the pull request is similar to the way that we create one back when were working with branches.



## ■ Social Aspect of GitHub ( Pull Requests Revisited)

- When the owner of the upstream repository decides that the changes of a fork are to be merged, we'll need to actually perform the merge.
  - Merging can be a tricky.
  - Now when doing a merge, it really depends what changes have been made.
  - If the changes are on different lines in a file or files, chances are that GitHub will be able to figure out everything itself and perform the merge automatically .
  - Sometimes though if there is a conflicting change, GitHub will let us know that it can't complete the merge automatically.
  - GitHub won't perform the merge until all conflicts have been resolved.
  - Once you have resolved everything, a new commit, a merge commit, will be created that contains the final result.
  - As mentioned, GitHub can figure out if the merge can be done automatically.
- 

## ■ Social Aspect of GitHub ( Pull Requests Revisited)

- Alternatively, we can also select not to use the GitHub interface and instead perform the merge locally.
  - We'll need to use a diff tool in this case, which we can configure from the GitHub config.
  - Several tools exist, and of course we can select which one we'll use.
  - Often used options include KDiff3, P4Merge, vimdiff3 and Beyond Compare.
- 

# Social Aspect of GitHub ( Pull Requests Revisited)

## Pull Requests



Update others about changes you have made in a branch or fork

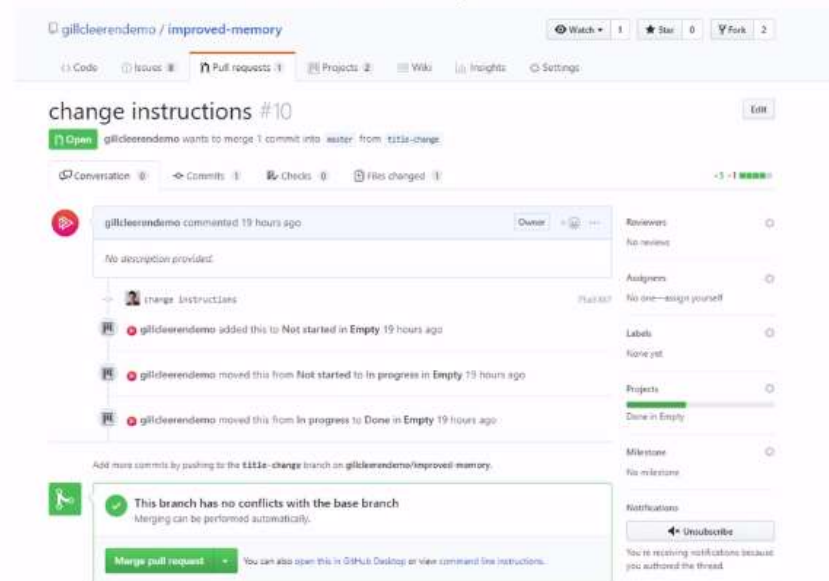


Discuss and review




Optionally merged into main branch

## GitHub's Pull Request Interface




## ■ Social Aspect of GitHub ( Using Gists)

- Now that we have seen probably the most social way of working on code in the previous part, let us change gears and see what other features GitHub brings to support its social coding platform, and we'll start with gists.

---
- Gists are a side project from GitHub meant to allow users to share snippets and notes.
  - Gists are available via a separate website under the GitHub domain. Namely, **[gist.github.com](https://gist.github.com)**. The interface very simple.
  - It allows to create a description for the gist and then the actual code.
  - Using gists we can share files or snippets, so part of a file.
  - Gists are used to remember something for ourself or share with your colleagues or even to your blog post.
  - While it's aimed at sharing files, we can combine multiple files into one single gist.
- 

## ■ Social Aspect of GitHub ( Using Gists)

- Gists, however, don't support directories.
  - Behind the scenes, gists use the same infrastructure as GitHub itself, meaning that everything here is still a repository as well.
- 
- When working with gists, there are basically two types, public and secret ones.
  - As the name implies, public gists are public.
  - They're searchable so everyone can find them pretty easily.
  - The secret ones, on the other hand, are private.
  - It's basically more something that we don't want to be discovered or aren't ready to share with everyone just yet, however, the link itself isn't private.
- 

## ■ Social Aspect of GitHub ( Using Gists)

- While private or secret gists don't show up in a search, for example, if you have the URL you can still access it.
- Finally, gists are also downloadable in the form of a zip file.
- One of the places where we can see gists being useful is as mentioned, embedded inside of a blog post.
- Via the gist page, one can get strict code that allows us to embed this gist into a page.

Gist is a simple way to share snippets and notes with others.

## Gists on GitHub

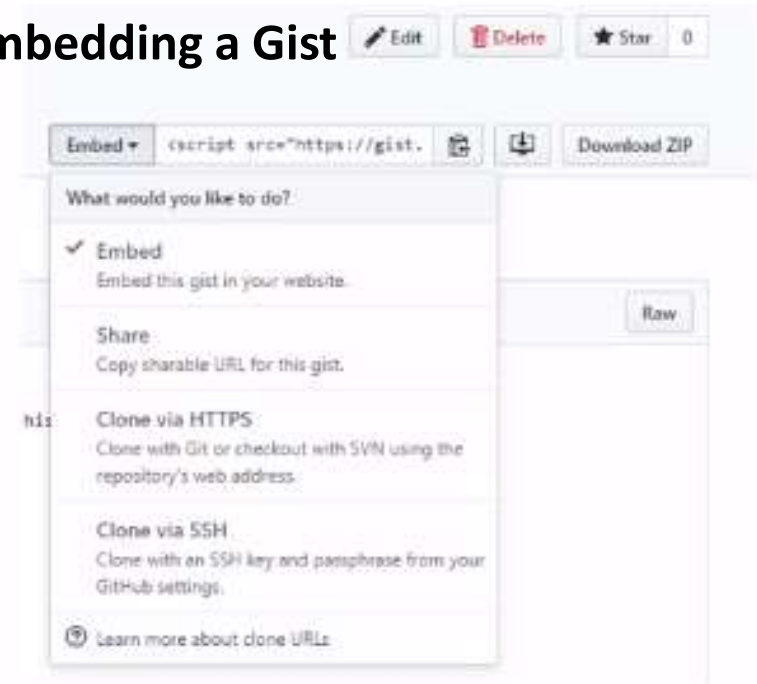
Allow sharing single file,  
snippet or even a full project

Repository-based

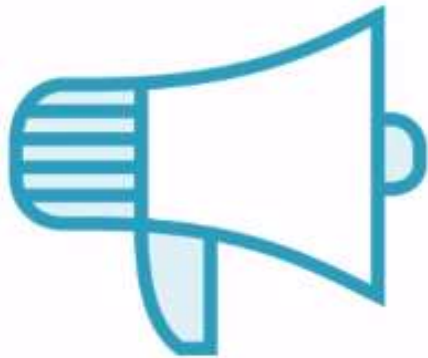
Public or secret

Downloadable

## Embedding a Gist



## Social Aspect Of GitHub(Managing Notifications)



### **Types of notifications**

- Participating notifications
- Watching notifications

### **Updates about activities or conversations**

- On GitHub
- Via email



## Social Aspect Of GitHub(Managing Notifications)

### Viewing Notifications Within GitHub

All activity



sample-organization233 forked sample-organization233/improved-memory from gilcleeren demo/improved-memory 12 hours ago

[gilcleerendemo/improved-memory](#)



Some description goes here

🌐 CSS Updated Aug 27



JamesNK pushed to [aspnet/Mvc](#) 4 days ago

2 commits to [JamesNK/escapointdatasource+routepattern](#)  
4f50a18 WTP  
194f835 Update MvcEndpointDataSource to use RoutePattern  
15 more commits »



JamesNK pushed to [aspnet/Mvc](#) 4 days ago



2 commits to [master](#)  
95a3a3a+ Replace remaining references to global routing (#8112)  
e3d45d7 Merge branch 'merge/release/2.2-to-master'



pranavlm pushed to [aspnet/Mvc](#) 4 days ago

7 commits to [pranavlshn/update-tests](#)  
93da3a+ Replace remaining references to global routing (#8112)  
947a26f Update tests to latest compat switch  
5 more commits »

## Managing Entire Project Through GitHub

---


Working with issues

Creating a milestone

Working with projects

Setting up a wiki

## Managing Entire Project Through GitHub (Working With Issues)

- Issues are a fundamental building block for GitHub in allowing us to manage our work through the interface.
- 
- The word issues can trick in thinking that they are a negative thing.
  - In GitHub, issues is just a name to indicate everything that's going on from bugs, to work tasks, to enhancements, and even ideas.
  - All these feedback items are basically connected in one large tracker, which is called issues.
  - Each repository has an issues tracker.
  - New features that need to be created can be trained by the creation of an issue.
  - Issues can also be linked to a pull request.
  - We can then maybe merge the pull request, close the corresponding issue.
- 

## **Managing Entire Project Through GitHub (Working With Issues)**

- Closing the issue also doesn't mean that the information is gone all of the related information remains available in GitHub.
- When we use issues, we also can be notified using the notification system.

---
- For example, you have created an issue, when someone then creates a comment on that issue, you'll get notified which helps you in staying up to date regarding the state of the issue.
- Pull requests can also be associated with issues.
- In a pull request we can mention an issue number combined with a keyword.
- This will then create the association and issues can also be used to work with milestones.
- Finally, it's also possible to create issues from code in a file or a pull request as issues are a GitHub feature, the site has all the required infrastructure to work with them.
- Probably the most relevant screen in this area is of course the issues tracker itself.
- The issue tracker is linked to a repo and in the main screen of the issue tracker, we can see a list of all open issues by default and we can also set it to display other lists.
- Also from the interface we are capable of creating an issue.

# Managing Entire Project Through GitHub (Working With Issues)

Working with Issues



## GitHub Issue Tracker

aspnet / Mvc

Watch 871 Star 5,566 Fork 2,182

Code Issues 198 Pull requests 34 Projects 3 Wiki Insights

Filters issue is open Labels Milestones New issue

198 Open 4,880 Closed Author Labels Projects Milestones Assignee Sort

- InvalidOperationException thrown in tag helpers using cache in 2.2.0 preview 1  
#8330 opened 23 hours ago by martinostello
- TagHelper for <datalist>  
#8322 opened 3 days ago by Skrypt
- Rendering parts of page based on given criteria  
#8321 opened 4 days ago by JoshClose
- Test failure: VersionedApi\_CanUseOrderToDisambiguate\_OverlappingVersionRanges 1: Ready  
Branchmaster PR: 1: Required task test-failure  
#8319 opened 4 days ago by aspnet-hello 2.2.0-preview2
- ProblemDetails - Avoid serializing empty properties 2: Working PR: 1: Required cost: 5 enhancement  
#8317 opened 4 days ago by pranavkm 2.2.0-preview2
- [Discussion]: 3.0: Deprecating MvcPrecompilation tool discussion  
#8313 opened 4 days ago by pranavkm Discussions
- Unable to use IFormFile with ApiController in 2.1 investigate  
#8311 opened 5 days ago by Eilon 2.2.0-preview2

## Managing Entire Project Through GitHub (Working With Issues)

Creating an issue




### BuiltIn Labels

bug  
duplicate  
enhancement  
help wanted  
question  
wontfix  
good first issue  
invalid

- **Labels** can be applied on issues, but also on pull requests.
- They are typically used to organize and also prioritize items such as issues.
- Again, labels are a GitHub thing and they are added on issues and pull requests and they too are GitHub-specific things.
- By default, GitHub already comes with a set of built-in labels, but if these do not suffice one can create their own and even delete the built-in ones.
- By default, eight labels are built into GitHub, and so they are available on each repository for you to use out of the box.

## **Managing Entire Project Through GitHub (Creating Milestones)**

- A MileStone allows us to track if we have reached a certain point.
  - Typically, this is done using a list of issues of which we track the progress.
  - Next to issues, milestones can also be used on a group of pull requests.
  - Milestones are, again, a GitHub thing so working with them, is again, done with the interface.
  - A milestone will typically include things like a due date, a completion percentage, and a list and number of open issues and pull requests.
- 

## **Managing Entire Project Through GitHub (Working With Projects)**

- When someone talks about managing their project on GitHub, they probably are referring to a project board.
- A project board can be used to give you an at-a-glance overview of the status of the project's work.
- The main idea behind the board is of course giving us an overview to help with the organization and prioritization of the work of the project.
- For a given repository, we can create a single board, but it's perfectly possible to create multiple boards.
- For example, we can choose to create a specific board to manage the work and the related issues for a certain feature.
- A board can also be used to manage a checklist or even a roadmap.
- The main benefit is that we get a visual overview of the work in several columns.
- When someone talks about managing a project on GitHub, they are probably referring to project boards.



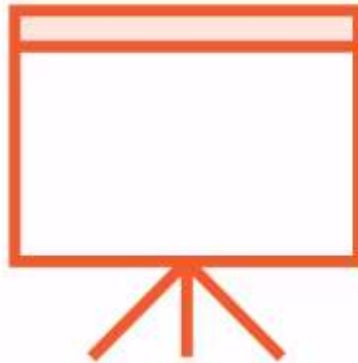
## **Managing Entire Project Through GitHub (Working With Projects)**

- As the name implies, a project board can be used to give an at-a-glance overview of the status of one's work.
- The main idea behind the board is of course giving an overview to help with the organization and prioritization of the work of the project.
- The currency on GitHub to manage work are issues.
- Since the project board is all about managing the work within a project, it's quite logical that on the project board we'll typically list issues and pull requests.
- On the project board, these items are shown as cards and we drag the cards between the different columns on the project board.
- These cards also contain other related data such as to who it is assigned or the assigned labels.
- GitHub comes with some templates baked in that we can select to set up our new board.
- Based on the selected template, the configured board comes with some columns and some base cards default.
- The available templates include basic kanban, automated kanban, automated kanban with review, and bug triage.

## Managing Entire Project Through GitHub (Working With Projects)

### Project Boards

Organization of work



Work on a specific feature

Checklists

Roadmap

# Managing Entire Project Through GitHub (Working With Projects)

## A Sample Project Board

The screenshot displays a GitHub Project Board for the repository `aspnet/Mvc`. The board is organized into four columns: **Design**, **Ready**, **Working**, and **Done**. Each column contains a list of project items, which are GitHub issues or pull requests, each with a title, progress bar, and status labels.

**Design Column:**

- Improvements to Swagger generation  
#6877 opened by rymowak  
Labels: 1 - Ready, PRI: 1 - Required, needs design
- Track template and scaffolding updates for new Web API conventions and updates  
#6915 opened by danroth27  
Label: task
- Figure out what to do about `ProblemDetails.Type`  
#6913 opened by rymowak  
Labels: PRI: 3 - Optional, needs design

**Ready Column:**

- Convert some "basic" 4XX responses into 'problem' responses  
#6786 opened by rymowak  
Labels: 3 - Done, PRI: 1 - Required, cost: 5, enhancement
- prototype build-time code generation using swagger  
#6788 opened by rymowak  
Labels: 3 - Done, PRI: 1 - Required, task
- Add swagger tool for build-time swagger generation  
#6767 opened by rymowak  
Labels: 1 - Ready, PRI: 1 - Required, enhancement

**Working Column:**

- (Empty)

**Done Column:**

- Report good error messages when JSON fails to deserialize  
#4862 opened by DaveDDE  
Labels: 3 - Done, bug
- `JsonInputFormatter`, `ModelState` and error response message.  
#4607 opened by MicahZoltu  
Label: 3 - Done
- For types annotated with `ApiControllerAttribute`, consider making complex type parameters `FromBody` by default  
#6647 opened by pranavkm  
Label: 3 - Done
- Add support for validation attributes on parameters  
#6648 opened by pranavkm  
Label: 3 - Done

## **Managing Entire Project Through GitHub (Setting Up Wiki)**

- In GitHub, we have the ability to add a wiki to our project.
- A project without good documentation is a regular project.
- While it's often the least interesting part, good documentation on things such as coding guidelines or best practices are valuable for a project.
- GitHub comes with the ability built in to add a wiki to a repository, giving a central location to manage all the information about a project.
- Wikis are linked on GitHub with a repository, thus describing the project.
- Project documentation can contain information for end users about the project, such as how to use the code, why certain things work the way that they were created, and much more.
- One can choose of course what to put in there.
- The wiki can contain many pages and can be edited online directly from GitHub or offline and then the changes can be pushed .

## **Managing Entire Project Through GitHub (Setting Up Wiki)**

- In that view, it's different from the README file, which is also a form of documentation for the project.
- The README will typically contain an overview or a getting started while the wiki can be used to provide more complete information about the project.
- Of course, we can reference the wiki from within the read me but WIKI can be done in mark down syntax.

## Working With Organizations and Teams(Creating Organizations)

- GitHub organizations are broader than just being useful for a company.
- GitHub defines an organization more in general as a shared account which can be used to work and collaborate on one or more projects, it could be that the company creates an organization within GitHub and assigns its members to that organization.
- We can also create an organization to bring together a number of developers to work together on a project.
- Organizations are owned by the owner, and owners and administrators can manage the members and the projects, and can also manage the access.
- Organizations can be used both using a free or a paid account.
- If we use an organization account just as with a personal account, we can, using this account, create an unlimited number of public repositories, and manage within that free organization and unlimited amount of collaborators.
- If we decide we want to create an organization, we can create one from scratch.
- GitHub offers an interface where we can create an organization.

## Working With Organizations and Teams(Creating Organization)

- Next to creating one from scratch, it's also possible to convert an existing account, so an individual account into an organization.
- Doing so will give you more fine-grained options to manage who within the organization gets access to which repositories.
- Through GitHub, one can simply add new members and they'll get an invite to become part of the organization.
- Within an organization, there are just a couple of roles, let's say, built-in, the owner and the member.
- The owner role is used to manage the organization.


Organization ≠ Company

## Working With Organizations and Teams(Working With Teams)


- Once we have the organization created, we'll probably want to manage the people within the organization and assign them into roles.
- That is possible in GitHub using the concept of teams.
- A team is defined by GitHub as a way to create groups of members that reflect the structure of a company or group.
- Using teams, we can manage permissions, and the system comes with a cascading structure to make it easier, and it is possible to nest teams.
- If it is a large organization to manage, this will definitely make things easier.
- Members can be part of one or more team.
- The organization owner can, through the system, manage which teams get access to which repositories.



## ■ Working With Organizations and Teams(Working With Teams)

- Other interesting features are also enabled to work with teams.
  - For example, it's possible to mention a team name, add a comment on a pull request, and then all members within that team will get a notification.
  - Again, we see how the system of notifications runs through GitHub entirely, and it's accessible from multiple places.
  - Teams also get like a home page for the team, the team page.
- 

## ■ Working With Organizations and Teams(Working With Teams)

- Here we see an example of one of those team pages in GitHub.
  - It contains the team's picture, which is the avatar for that particular team, and it'll also list out the members within that team.
  - GitHub offers a fine-grained way to manage teams and through the GitHub interface, we can add or remove members in a team.
  - Teams have their own profile and teams can be nested within a hierarchy.
- 

## Working With Organizations and Teams(Working With Teams)

### Teams Page

The screenshot displays the GitHub Teams interface. At the top, a dark navigation bar contains the GitHub logo, a search bar, and links for Pull requests, Issues, Marketplace, and Explore. Below this, a breadcrumb trail shows 'sample-organization1233' and 'developers'. The main content area is divided into two columns. The left column features the team's profile, including a header image of two stylized figures, the team name 'developers', the organization handle '@sample-organization1233/developers', a description 'Dev team @Snowball - Edit', and a list of 2 members. The right column shows the team's discussion feed. It starts with a prompt to 'Start a discussion with @sample-organization1233/developers'. Below this are tabs for 'Recent' and 'Pinned'. A discussion titled 'Colors don't look great' by user 'gillcleerendemo' is visible, posted '2 minutes ago'. The discussion content reads 'We need to improve the site's colors dramatically,' and includes a 'Reply...' input field.

sample-organization1233 / developers

Discussions Members (2) Teams (2) Repositories (2) Projects (0) Settings

Start a discussion with @sample-organization1233/developers

Recent Pinned (0)

**Colors don't look great**

gillcleerendemo  
2 minutes ago

We need to improve the site's colors dramatically,

Reply...

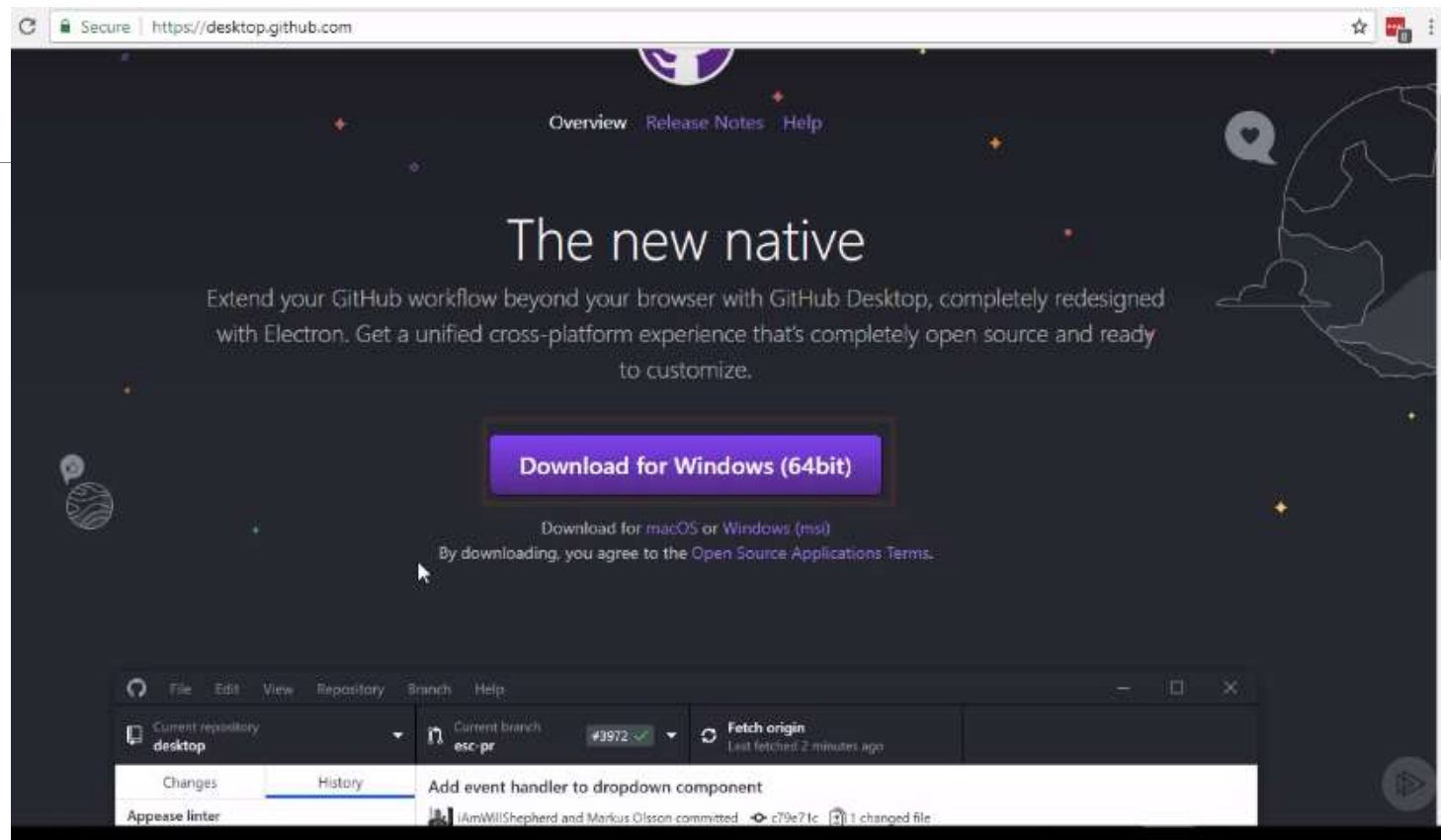
## Interacting With GitHub Through the GitHub Desktop Client(What is Github Desktop)

- GitHub Desktop is a tool developed by the people behind GitHub, allowing to perform some, not all, interactions with GitHub through a desktop client.
- GitHub Desktop is an Electron app.
- Most developers using Git will keep on using the command line to work with GitHub, and that's perfectly fine.
- The main purpose of the tool is bringing down the learning curve for people not really experienced in working with Git.
- Also, according to the creators, we can use the desktop client in combination with your current workflow.
- Whether or not we're open to use a GUI to interact with GitHub ultimately depends on each one.
- The GitHub Desktop is not the only client-side tool which allows interacting with GitHub, the others tools are, which offer similar functionality, include Sourcetree, GitKraken, and the Git Extensions.

## ■ Interacting With GitHub Through the GitHub Desktop Client(Installation)

- GitHub Desktop is available only for Windows and Mac.
- There's no Linux version at the time of the creation of this course.
- However, a Linux client was created by the community.
- The tool is available to download for free from the URL <https://desktop.github.com/>
- Once the tool is installed we can sign in with the GitHub account.
- It may be so that on GitHub you have enabled two-factor authentication, if so, this way of authenticating with the tool is also required.

## ■ Interacting With GitHub Through the GitHub Desktop Client(Installation)



## Core Concepts : What is Git?

Git is a decentralized or distributed version control system.

---

---





---



---



---



---



---



---



---



---





---



---



---



---



---



---



---



---





---



---



---



---



---



---



---



---





---



---



---



---



---



---



---

