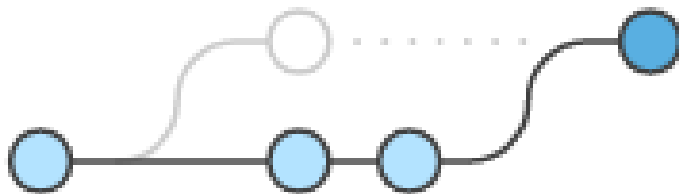


Introduction Git

What is Version Control System - VCS /Source Code Management -SCM

- *Version control is a piece of software which allows you to record and preserve the history of changes made to directories and files. If you mess things up, you can retrieve an earlier version of your project.*
- *A Version control system keeps track of a set of files and saves snapshots.*
- *Using version control allows you to confidently make changes to your code (and any other files), with the ability to roll back to any previous state.*
- *This help avoid filling our directories up with files that look like this:*
 1. *my_code.py*
 2. *my_code_version2.py*
 3. *my_code_version2B-RPA-edit.py*
 4. *my_code_FINAL_VERSION.py*
 5. *my_code_THIS_IS_ACTUALLY_THE FINAL VERSION.py*



Version Control vs. Git vs. GitHub

- **Version Control** is the general concept of tracking progress (controlling) versions of your code.
- **Git** is a specific software that allows you to do version control.
- **GitHub** is a website to host different folders that are version controlled (= repositories). Github is like google docs for programming projects.
- All of these platforms are independent of your file/code type.



The Octocat
octocat

- *By far, the most widely used modern version control system in the world today is Git.*
- *Git is a mature, actively maintained open-source project originally developed in **2005 by Linus Torvalds**, the famous creator of the Linux operating system kernel.*
- *A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers*
- *Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System).*
- *Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.*
- *In addition to being distributed, Git has been designed with performance, security and flexibility in mind.*



Why should you be using git?

1. *To track changes and progress in your code efficiently*
2. *To save earlier states of your code (e.g. versions of a code you used for a paper)*
3. *To share your code with collaborators either to trouble shoot or for a collaborative coding project (requires github)*
4. *To back-up your code (requires github)*
5. *To add it to your CV (requires github)*
6. *For collaborative projects in which several people work on the same code, version control is a must.*

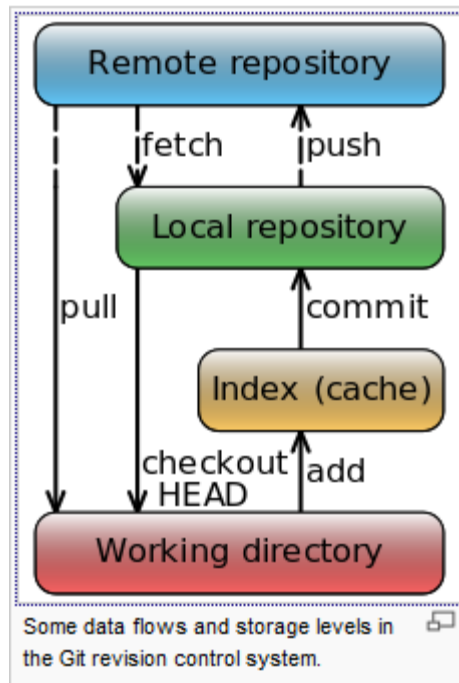
Git Process Flow

The **Working Directory** is wherever your files are on your local machine.

The **Local Repository** is the **.git/** subdirectory **inside the Working Directory**.

The **Index** is a conceptual place that also physically resides in the **.git/** subdirectory.

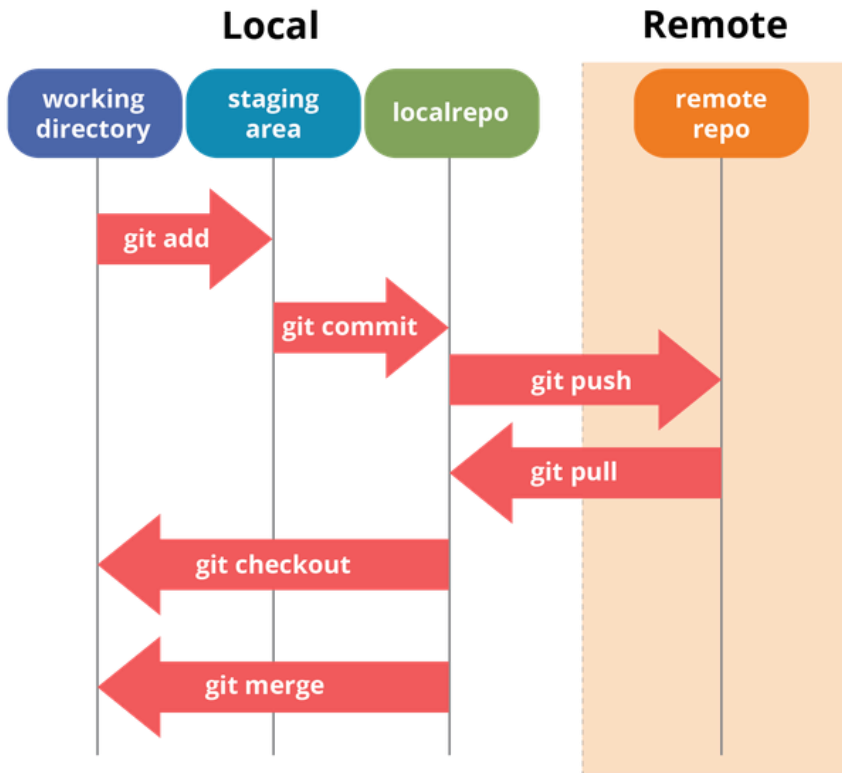
Remote repository is bare repository on server or in the filesystem.



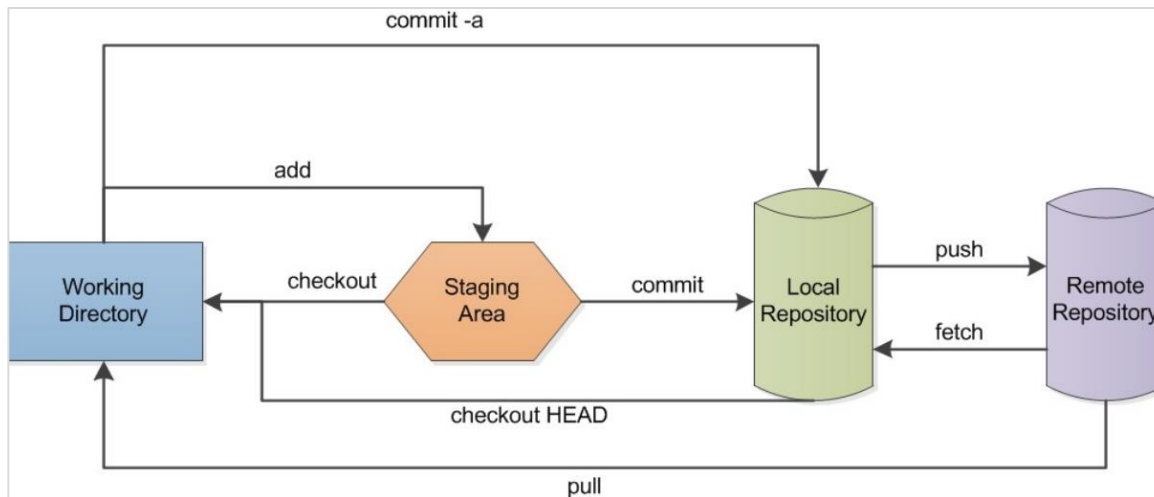
Git Process Flow

Git makes the distinction between three distinct areas/concepts:

- *The repository itself, which is stored within the .git directory, as discussed in the previous section*
- *The working tree, which corresponds to the current state of files on your filesystem*
- *The staging area (also called the index), which is the area that you can use to prepare commits / temporarily save your work*



Git Process Flow



commit -a: Directly commit modified and deleted files into the local repository (*no new files!*)

add: Add a file to the staging area.

checkout: Get a file from the staging area.

checkout HEAD: Get a file from the local repository

commit: Commit files from the staging area to the local repository

push: Send files to the remote repository

fetch: Get files from the remote repository

pull: Get files from the remote repository and put a copy in the working directory

Git Installation & Configuration

Git for Windows stand-alone installer

- *Download the latest Git for Windows installer : <https://gitforwindows.org/>*
- *When you've successfully started the installer, you should see the Git Setup wizard screen.*
- *Follow the Next and Finish prompts to complete the installation.*
- *The default options are pretty sensible for most users.*
- *Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).*
- *Run the commands to configure your Git username and email using the following commands,*

```
$ git config --global user.name <First Name Last Name since space in double quotes>  
$ git config --global user.email <emailed optionally in double quotes>  
$ git config --global user.password <Github token as applicable not in double quotes>
```

- *These details will be associated with any commits that you create:*

Git Installation & Configuration

Save the username and password globally:

```
git config --global user.name "fname lname"  
git config --global user.email "example@gmail.com"  
git config --global user.password "secret"
```

Get a specific setting,

```
git config --global --get user.name  
git config --global --get user.email  
git config --global --get user.password
```

Getting all Git settings:

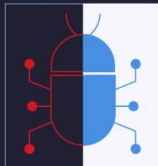
```
git config --list --show-origin
```

Note: You only have to do this once

Git Branching (Read Carefully)

- *Branching is a feature available in most modern version control systems.*
- *Instead of copying files from directory to directory, Git stores a branch as a reference to commit. The branch itself represents the HEAD of a series of commits.*
- *The default branch name in Git is **master**, which commonly represents the official, working version of your project.*
- *As you start making commits, the master branch points to the last commit you made.*
- *Every time you commit, the master branch pointer moves forward automatically.*
- *Think of a branch as a timeline of versions of a project as it progresses.*
- *Branching is a strategy that allows developers to take a snapshot of the master branch and test a new feature without corrupting the project in production.*
- *If the tests are successful, that feature can be merged back to the master branch and pushed to production.*

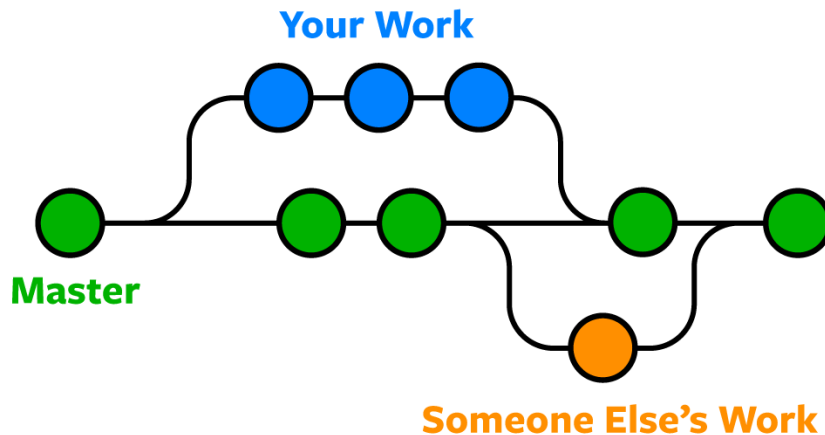
Git Branching



Git branching allows developers to **diverge from the production** version of code to **fix a bug** or add a feature.

Git Branching Commands

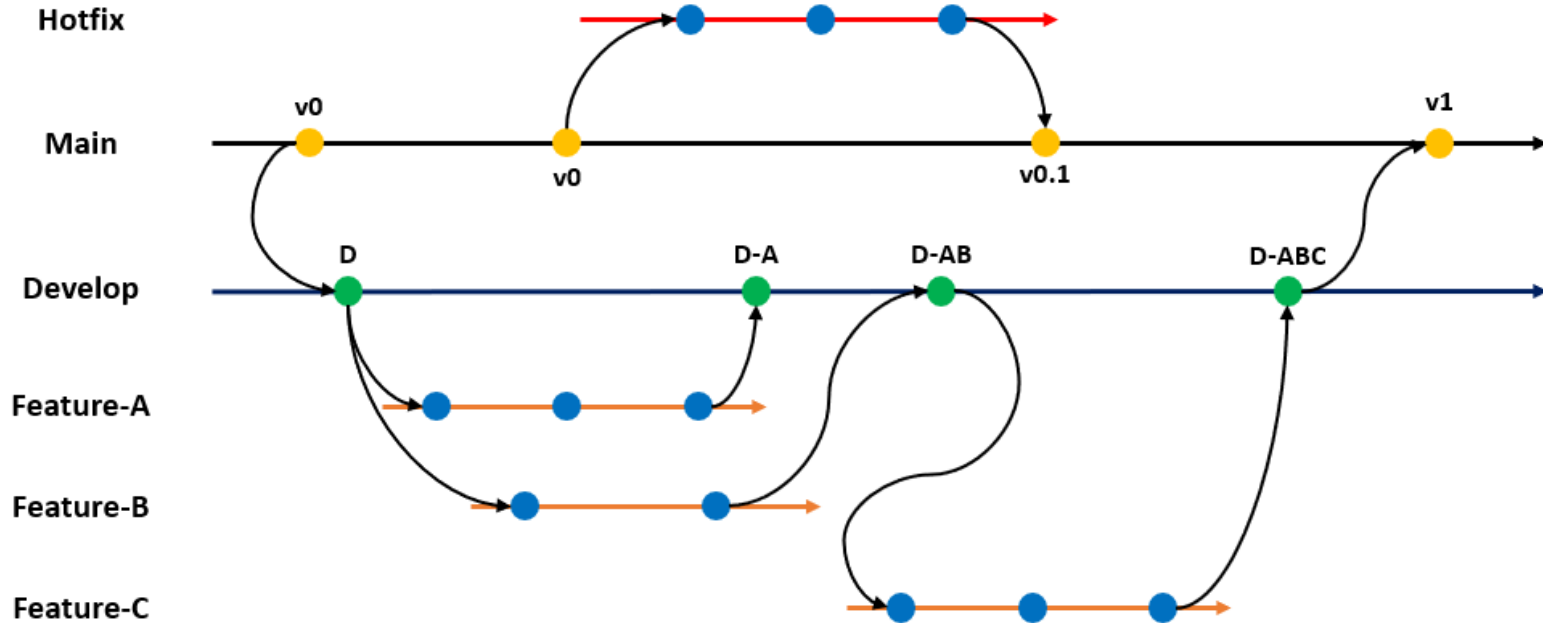
- `git branch -a` - shows you all branches that exist(local as well as remote)
- `git branch branch_name` – creates a branch based on current master with name `branch_name`
- `git checkout branch_name` – switch over to the branch called `branch_name` to work in that make any changes, commits etc
- `git branch -d branch_name` – removes that branch
- `git push -u origin <branch-name>` – to push the newly created branch in to the remote repository
- Merge branch into master
 - `git checkout master` – takes you back to the master branch
 - `git merge branch_name` – merges any changes you did into the master branch



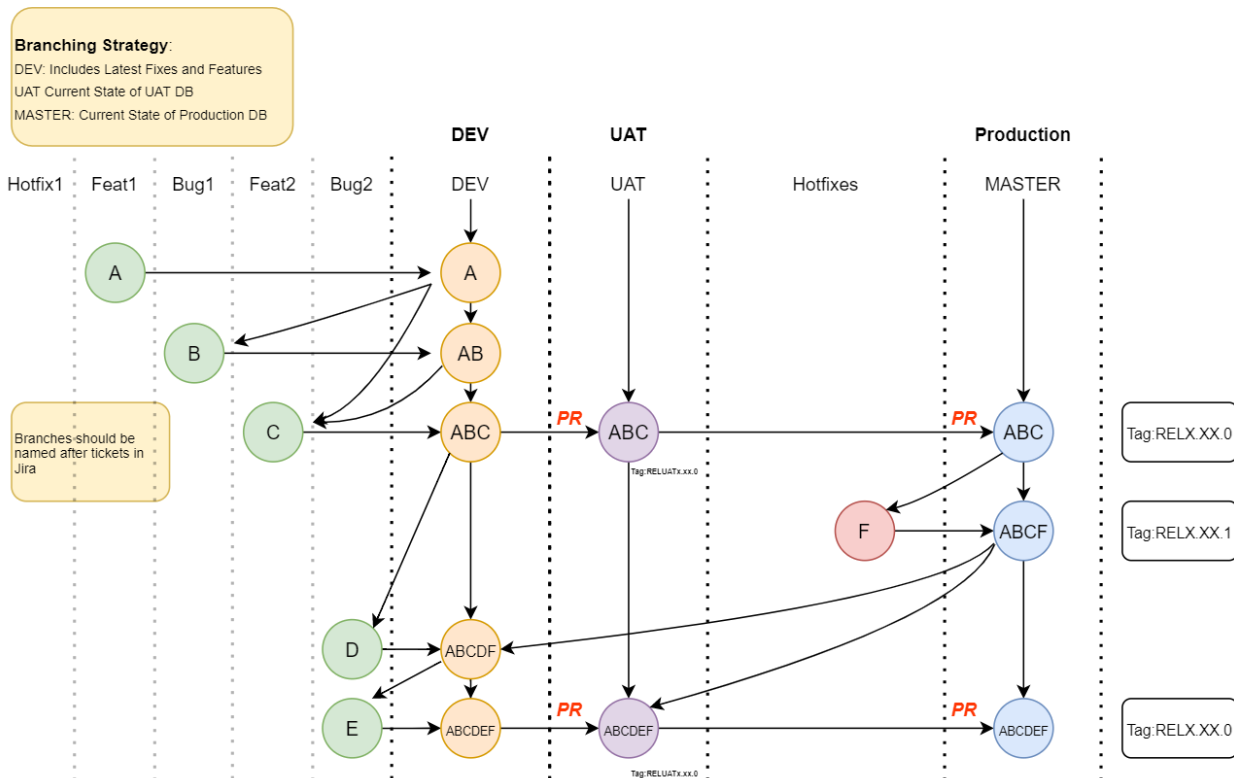
If master branch is ahead than feature branch how to fix issue?

- *Run below command in under master branch*
- `git checkout master`
- `git pull`
- `git checkout <branch_name>`
- `git merge master` - this will merge code of master into feature branch
- `git push origin <branch_name>`

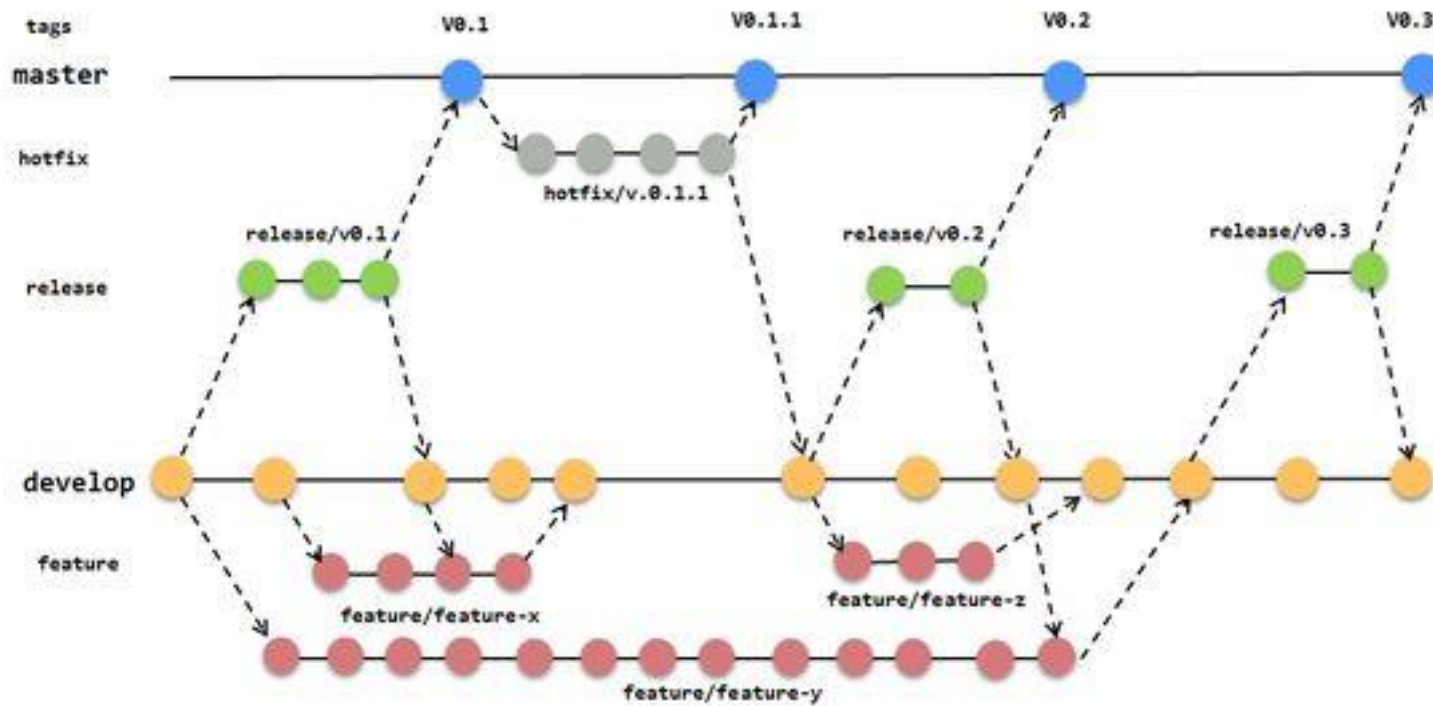
Git Branching Strategy in Companies



Git Branching Strategy in Companies



Git Branching Strategy in Companies



In Action | Creating branch is easy

```
MINGW64:/c/Users/GangadharParde/OneDrive - revature.com/Desktop/Demo/MyApp

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$ git branch feature/login

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$ git checkout -b feature/report
Switched to a new branch 'feature/report'

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (feature/report)
$ git branch -a
  feature/login
* feature/report
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (feature/report)
```

In Action | Smart branch only show its own files..

```
MINGW64/c/Users/GangadharParde/OneDrive - revature.com/Desktop/Demo/MyApp
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (feature/report)
$ ls
mvnw mvnw.cmd pom.xml src

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (feature/report)
$ touch report.html

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (feature/report)
$ git add .

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (feature/report)
$ git commit -m "Added report.html"
[feature/report 8b7be71] Added report.html
1 file changed, 1 insertion(+)
create mode 100644 report.html

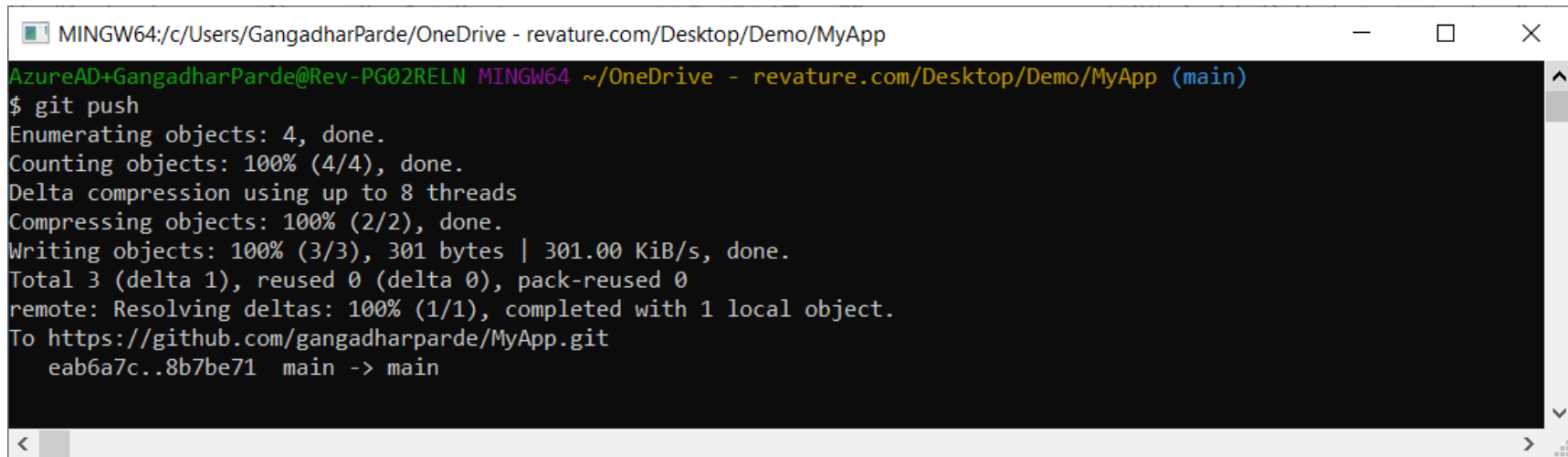
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (feature/report)
$ ls
mvnw mvnw.cmd pom.xml report.html src

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (feature/report)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$ ls
mvnw mvnw.cmd pom.xml src

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$
```

In Action | Pushing current branch to GitHub



```
MINGW64:/c/Users/GangadharParde/OneDrive - revature.com/Desktop/Demo/MyApp
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 301.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/gangadharparde/MyApp.git
   eab6a7c..8b7be71  main -> main
```

In Action | Pushing other branch to GitHub

```
MINGW64:/c:/Users/GangadharParde/OneDrive - revature.com/Desktop/Demo/MyApp
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$ git push -u origin feature/report
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature/report' on GitHub by visiting:
remote:   https://github.com/gangadharparde/MyApp/pull/new/feature/report
remote:
To https://github.com/gangadharparde/MyApp.git
 * [new branch]      feature/report -> feature/report
branch 'feature/report' set up to track 'origin/feature/report'.

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$ git push -u origin feature/login
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature/login' on GitHub by visiting:
remote:   https://github.com/gangadharparde/MyApp/pull/new/feature/login
remote:
To https://github.com/gangadharparde/MyApp.git
 * [new branch]      feature/login -> feature/login
branch 'feature/login' set up to track 'origin/feature/login'.

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Demo/MyApp (main)
$
```


How to create & push project in Github

1. *Cloning the remote repository from GitHub using Git then push the code*
2. *Using eclipse IDE*
3. *Adding a local repository to GitHub using Git (Last preference)*

Note: In all above three cases you first need to create new repository into Github.

Create Blank Repository From GitHub will help with basic commands like below

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

 Set up in Desktop

or

HTTPS

SSH

`https://github.com/[redacted]/MyApp.git`



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

...or create a new repository on the command line

```
echo "# MyApp" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/gangadharparde/MyApp.git
git push -u origin main
```



-M is a flag (shortcut) for --move --force. It renames the branch main (since the default branch name for repositories created using the command line is master, while those created in GitHub [starting in Oct. 2020] have a default name of main) and forces it (allows renaming of the branch even if the new branch name already exists)

...or push an existing repository from the command line

```
git remote add origin https://github.com/gangadharparde/MyApp.git
git branch -M main
git push -u origin main
```

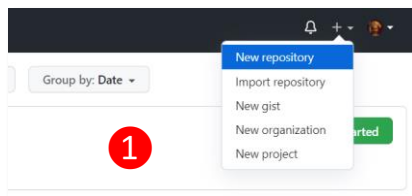


...or import code from another repository

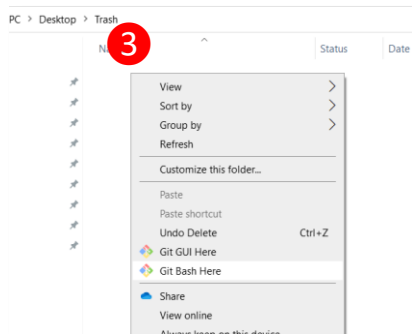
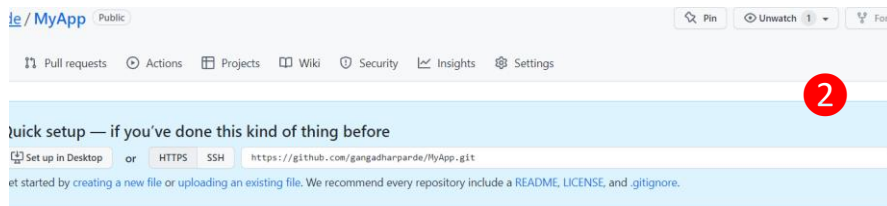
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

1. Cloning the remote repository from GitHub using Git then push the code



1. Cloning the remote repository from GitHub using Git then push the code
2. Inside GitHub you will see the repository URL and few git commands for reference.




3. In windows, inside empty folder of your choice right click and select **Git Bash Here** (This menu comes post installation of Git in you machine)
4. Now clone the project using the git clone command and go the repository check you are on main branch



Create springboot application from <https://start.spring.io/>

5



☒ Maven Project

☐ Gradle Project

☒ Java

☐ Kotlin

☐ Groovy

Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M2) ☐ 2.7.0 (SNAPSHOT) ☐ 2.7.0 (RC1) ☐ 2.6.8 (SNAPSHOT) ☒ 2.6.7 ☐ 2.5.14 (SNAPSHOT) ☐ 2.5.13

Project Metadata

Group

com.gd

Artifact

hotel-18.degree

Name

hotel-18.degree

Description

Demo project for Spring Boot

Package name

com.gd.hotel-18.degree

Packaging

☒ Jar ☐ War

Java

☐ 18 ☐ 17 ☒ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL

MySQL JDBC and R2DBC driver.

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot Actuator OPS

Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Consul Discovery SPRING CLOUD DISCOVERY

Service discovery with Hashicorp Consul.

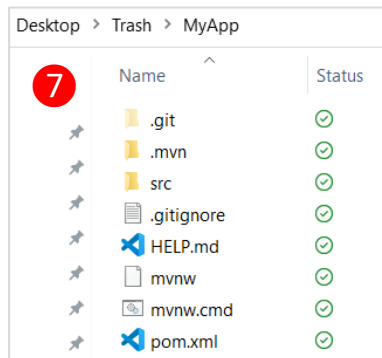
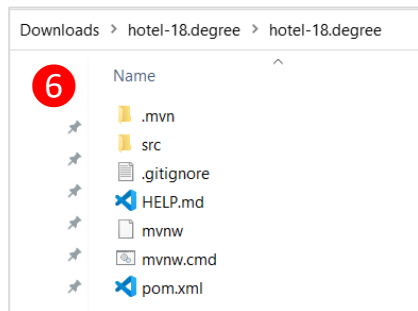
Choose Correct Dependencies & Click on Generate

GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

Extract project folder and copy all files to our GitHub cloned empty repo



```
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Trash/MyApp (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .gitignore
  .mvn/
  mvnw
  mvnw.cmd
  pom.xml
  src/

nothing added to commit but untracked files present (use "git add" to track)
```

8

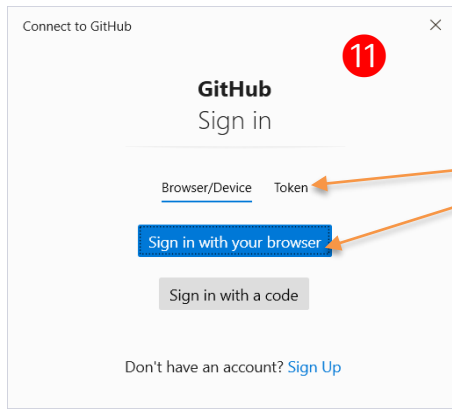
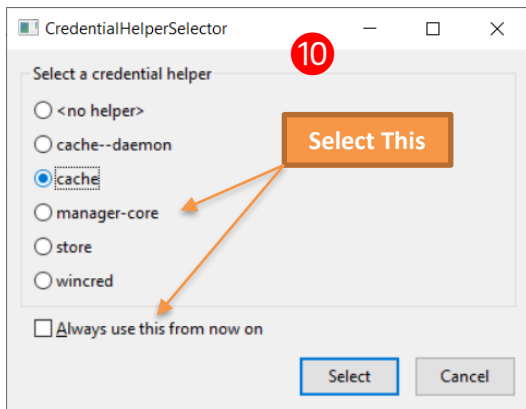
```
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Trash/MyApp (main)
$ git add .
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in .mvn/wrapper/maven-wrapper.properties.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in mvnw.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in mvnw.cmd.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in pom.xml.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/main/java/com/gd/hotel18/degree/Application.java.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/main/resources/application.properties.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/test/java/com/gd/hotel18/degree/ApplicationTests.java.
The file will have its original line endings in your working directory
```

9

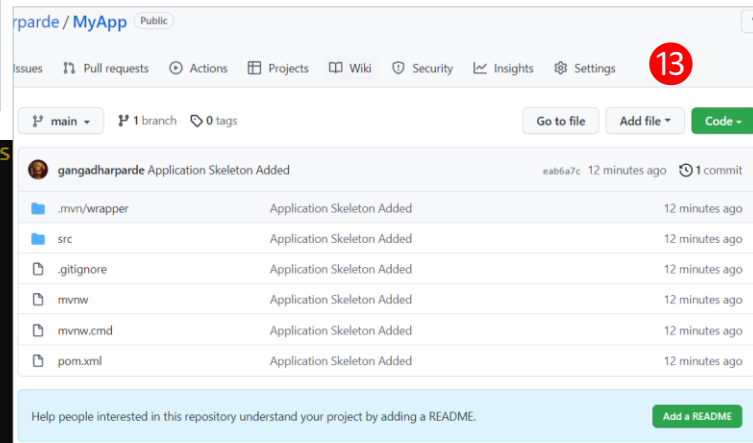
```
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Trash/MyApp (main)
$ git commit -m "Application Skeleton Added"
[main (root-commit) eab6a7c] Application Skeleton Added
 9 files changed, 642 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .mvn/wrapper/maven-wrapper.jar
 create mode 100644 .mvn/wrapper/maven-wrapper.properties
 create mode 100644 mvnw
 create mode 100644 mvnw.cmd
 create mode 100644 pom.xml
 create mode 100644 src/main/java/com/gd/hotel18/degree/Application.java
 create mode 100644 src/main/resources/application.properties
 create mode 100644 src/test/java/com/gd/hotel18/degree/ApplicationTests.java
```

```
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/Trash/MyApp (main)
$ git status
On branch main
Your branch is based on 'origin/main', but the upstream is gone.
(use "git branch --unset-upstream" to fixup)
```

Finally Push Code to GitHub from local repository to remote repository



```
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Des
$ git push
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (27/27), 59.02 KiB | 9.84 MiB/s, done.
Total 27 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/gangadharparde/MyApp.git
* [new branch]      main -> main
```



2. Using eclipse IDE

Steps:

1. *Open github.com and sign in and create a new repository.*
2. *Copy URL of the new repository.*
3. *Open Eclipse.*
4. *Select Project which you want to push on GitHub->right click.*
5. *select Team->share Project->Git-> "push branch " ...".*
6. *Follow the wizard until it finishes.*

<https://www.youtube.com/watch?v=Huwf0TgWrOw>

<https://www.geeksforgeeks.org/how-to-export-eclipse-projects-to-github/#:~:text=Step%201%3A%20Open%20Eclipse%20IDE,go%20to%20Team%2D%3Ecommit.>

Adding a local repository to GitHub using Git

- 1 Create a new repository on GitHub.com. To avoid errors, do not initialize the new repository with README, license, or gitignore files. You can add these files after your project has been pushed to GitHub.
- 2 Open Git Bash.
- 3 Change the current working directory to your local project.
- 4 Initialize the local directory as a Git repository.

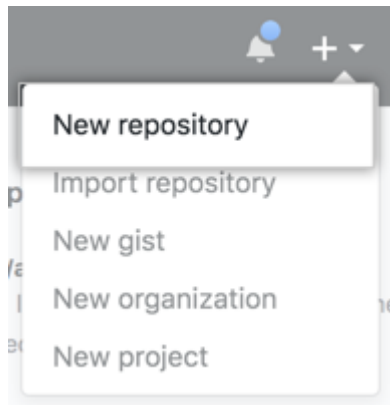
```
git init -b main
```

- 5 Add the files in your new local repository. This stages them for the first commit.

```
git add .  
# Adds the files in the local repository and stages them for commit.  
# To unstage a file, use 'git reset HEAD YOUR-FILE'.
```

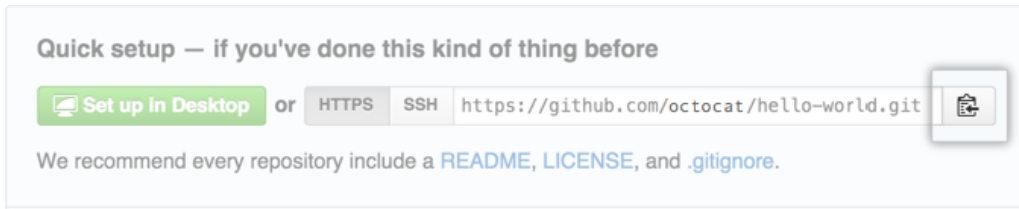
- 6 Commit the files that you've staged in your local repository

```
git commit -m "First commit"  
# Commits the tracked changes and prepares them to be pushed to a remote repository.  
# To remove this commit and modify the file, use 'git reset --soft HEAD~1' and commit and  
add the file again.
```



Adding a local repository to GitHub using Git

- 7 At the top of your repository on GitHub.com's Quick Setup page, click to copy the remote repository URL.



- 8 In the Command prompt, add the URL for the remote repository where your local repository will be pushed.

```
git remote add origin <REMOTE_URL> # Sets the new remote
git remote -v # Verifies the new remote URL
```

- 9 Push the changes in your local repository to GitHub.com.

```
git push origin main
# Pushes the changes in your local repository up to the remote repository you specified as
the origin
```

```
MINGW64:/c/Users/GangadharParde/OneDrive - revature.com/Desktop/New folder
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder
$ git init -b main
Initialized empty Git repository in C:/Users/GangadharParde/OneDrive - revature.com/Desktop/New folder/.git/

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files present (use "git add" to track)

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder (main)
$ git add .

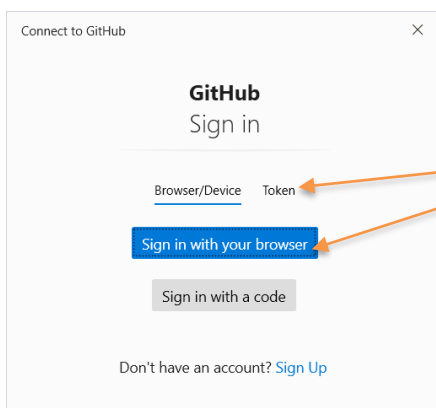
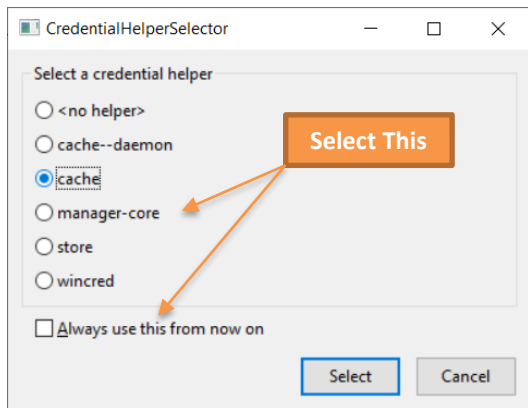
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder (main)
$ git commit -m "Added index.html"
[main (root-commit) 69c6a65] Added index.html
 1 file changed, 9 insertions(+)
 create mode 100644 index.html

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder (main)
$ git remote add origin https://github.com/gangadharparde/MyApp.git

AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder (main)
```

Now once you try to push the code to origin it will show two pop ups

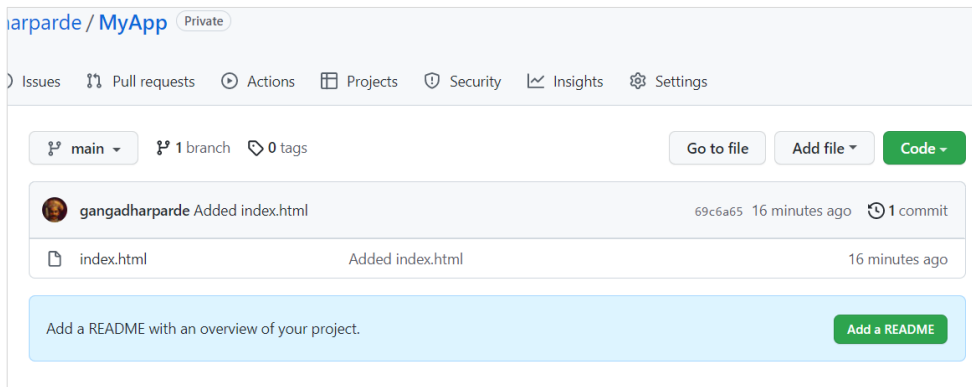
```
MINGW64:/c/Users/GangadharParde/OneDrive - revature.com/Desktop/New folder
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder (main)
$ git remote -v
origin https://github.com/gangadharparde/MyApp.git (fetch)
origin https://github.com/gangadharparde/MyApp.git (push)
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder (main)
$ git push origin main
```



Successful Push will show below log:

```
AzureAD+GangadharParde@Rev-PG02RELN MINGW64 ~/OneDrive - revature.com/Desktop/New folder (main)
$ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 291 bytes | 291.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/gangadharparde/MyApp.git
 * [new branch]      main -> main
```

Verify the code is pushed to GitHub Repo-



The screenshot shows the GitHub interface for a repository named 'MyApp' (marked as Private) by the user 'gangadharparde'. The repository has 1 branch (main) and 0 tags. A commit by 'gangadharparde' is shown, titled 'Added index.html', with commit hash '69c6a65' and timestamp '16 minutes ago'. Below the commit, a file 'index.html' is listed with the description 'Added index.html' and timestamp '16 minutes ago'. At the bottom, there is a light blue box with the text 'Add a README with an overview of your project.' and a green button labeled 'Add a README'.

Summary of Commands

- `git clone https://github.com/xxxx/xxxx.git` – clone/download project from GitHub to start development.
- `git status` – shows you the files that are tracked
- `git add .` – to add all files to the staging area (note dot after add). adds the file to be tracked. adds / stages all of the files in the current directory.
- `git commit -m <"commit message">` – commit to store the first version of your code.
- `git push` – used to upload local repository content to a remote repository
- Work on your file. Whenever you are at a new point where you want to save your current state (version) add and commit.
- `git pull` – any changes done by other developers to your branch are also added to your repo.
- `git merge`

[Git - Branches in a Nutshell \(git-scm.com\)](https://git-scm.com)

<https://www.nobledesktop.com/learn/git/git-branches>

Help document of any git commands

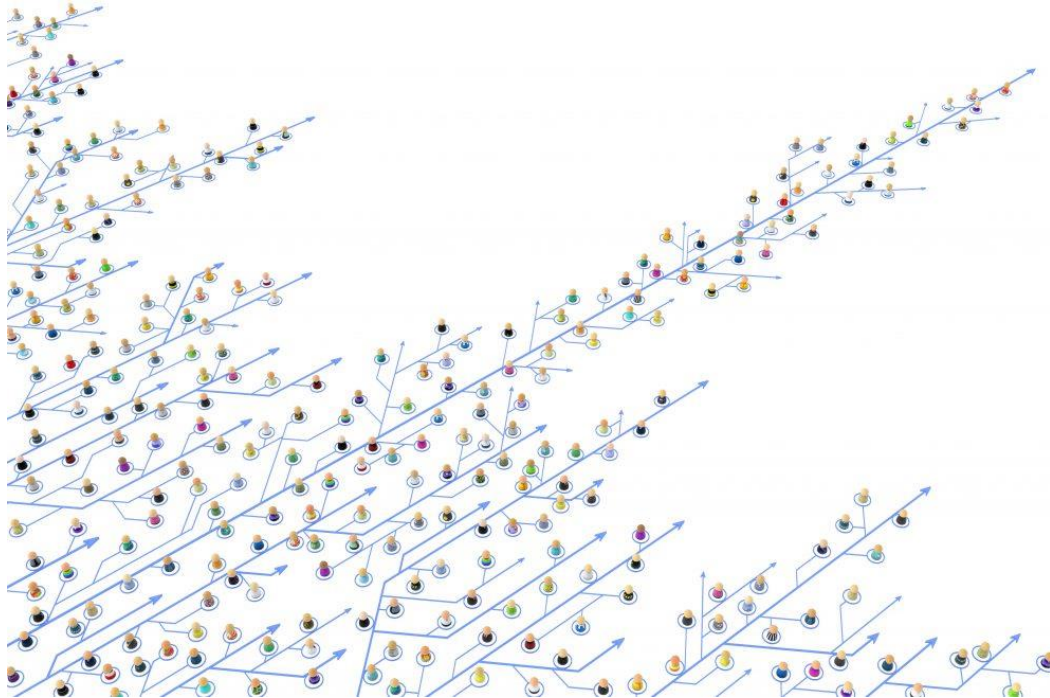
- You can get help html document of any command using below command note `--help` switch.
`git checkout --help`

Additional References | Optional

- *You can also use SSH key to connect with GitHub (Optional)*
- *What is an SSH KEY? - An SSH key is an access credential for the SSH (secure shell) network protocol. This authenticated and encrypted secure network protocol is used for remote communication between machines on an unsecured open network. SSH is used for remote file transfer, network management, and remote operating system access.*
- *Official documentation: <https://try.github.io/>*
- *Nice to read: <https://swcarpentry.github.io/git-novice/>*
- *Interactive learning: <https://learngitbranching.js.org/>*

Pull Request, Approval by Technical SME or Product Owner

- <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/about-pull-request-reviews>



Optional commands Using git – checking differences

- `git log` – shows you all your commits
- `git diff` – shows you any current changes
- `git diff commit#1 commit#2` – shows you changes between commits
- `git checkout commit#1 file_name` – reverts your file back to what it was at `commit#1` stage
- `git stash` – Enables you to save changes that you don't want to immediately commit. An alternative to this is creating a new branch, but in any case stash will record changes you have made since your last commit.
- A good case for using this is when you want to switch branches, but don't want to commit what you've been working on yet. – `git stash`
- You can see which stashes you've stored by using – `git stash list`
- and to restore the stash you can run the command – `git stash apply`

<https://stevenpcurtis.medium.com/common-git-commands-4663bab829c6>

<https://javascript.plainenglish.io/git-commands-that-make-your-life-easier-1f285653449d>

<https://www.deployinc.com/blog/branching-strategy/>

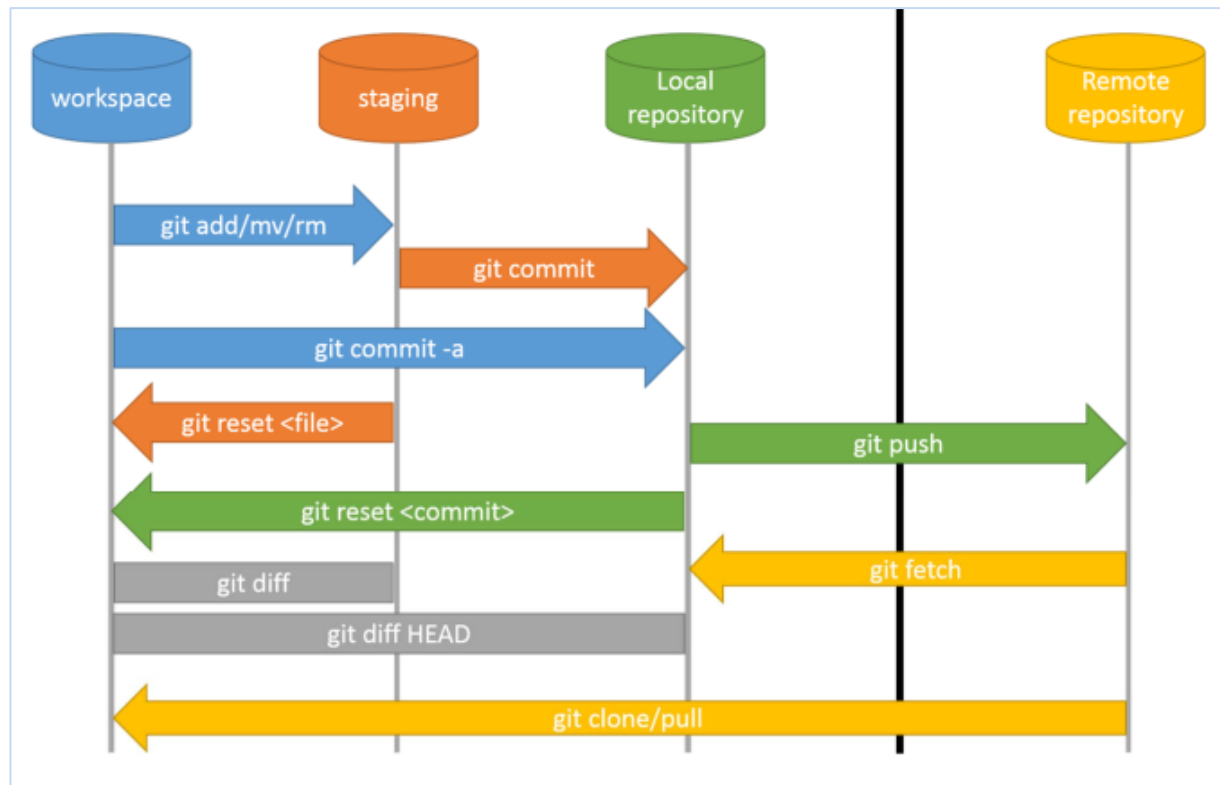
<https://towardsdatascience.com/how-to-structure-your-git-branching-strategy-by-a-data-engineer-45ff96857bb>

<https://pradeep1.com/blog/git-branching-strategies/>

<https://devcenter.heroku.com/articles/gitignore>

Git Process Flow

1. Generally, the code is submitted in three steps:
2. *git add* commit from workspace to staging area
3. *git commit* from staging area to local warehouse
4. *git push* submit from local warehouse to remote warehouse



Git Process Flow

