Using Github for project development

**Step 1 : Install git on your system and make your account on github**

* Installing on Linux

If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the basic package-management tool that comes with your distribution. If you’re on Fedora for example (or any closely-related RPM-based distro such as RHEL or CentOS), you can use dnf:

**$ sudo dnf install git-all**

**I**f you’re on a Debian-based distribution like Ubuntu, try apt-get:

**$ sudo apt-get install git-all**

* Installing on Windows

The most official build is available for download on the Git website. Just go to [http://git-scm.com/download/win](%20http://git-scm.com/download/win) and the download will start automatically.

* You can create github account here : <https://github.com/join>

Accounts are free for public repositories, but there's a charge for private repositories.

Every user account and organization on GitHub can have unlimited collaborators on any number of public repositories.

User and organization accounts with paid billing plans can also use unlimited private repositories. Anyone can view public repositories, while private repositories have their visibility limited to just you and your collaborators.

Organizations can also choose to upgrade to the Business plan for sophisticated user and identity management features and a service level agreement for uptime availability. For more information, see "[Organization billing plans](https://help.github.com/articles/organization-billing-plans)."

All of github plans and pricing for users and organization accounts can be seen at <https://github.com/pricing>. You can [upgrade or downgrade](https://help.github.com/articles/how-does-upgrading-or-downgrading-affect-the-billing-process) your billing plan at any time.

**Step 2: Global Setup**

To check whether git is installed or not and it’s version, use command:

**$ git --version**

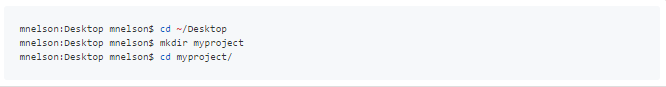
Now setup your name and email id, this helps in keeping track of who did what when we work in team :

**$ git config --global user.name “xyz”**

**$ git config --global user.email “**[**your\_email@example.com**](mailto:your_email@example.com)**”**

**Step 3: create a local repository(directory in which you will be creating your project)**

This is the folder which you want to keep as local repository(on your machine) and remote repository(git hub repository)

****

To initialize a git repository in the root of the folder, run the [**git init**](http://git-scm.com/docs/git-init)command:

**$ git init**

It will create a folder named .git in your local repository.

### Step 4: Add a new file to the repo

### You can add your project files to this repository. Once you've added or modified files in a folder containing a git repo, git will notice that changes have been made inside the repo. But, git won't officially keep track of the file (that is, put it in a commit),unless you explicitly tell it to.

### After creating the new file, you can use the [git status](http://git-scm.com/docs/git-status) command to see which files git knows exist.

### $ git status

### In this example , we have added melson .txt in the repository, myproject

### C:\Users\Vishnu\Desktop\gitdoc\Capture3.PNG

### Step 5: Add a file to the staging environment

When we make changes to our repo, git notices that a file has changed but won't do anything with it (like adding it in a commit).

A [**commit**](http://git-scm.com/docs/git-commit)is a record of what files we have changed since the last time we made a commit. Essentially, we make changes to our repo (for example, adding a file or modifying one) and then tell git to put those files into a commit.

Commits make up the essence of our project and allow us to go back to the state of a project at any point.

To add a file to a commit, we first need to add it to the staging environment. To do this, we can use the **git add <filename>**command.

### Add a file to the staging environment using the ****git add**** command.

### $ git add <file name>

### Use command git status to see the status of files in repo.

### $ git status

### C:\Users\Vishnu\Desktop\gitdoc\Capture4.PNG

### Step 6: Create a commit

### Run the command git commit -m "Your message about the commit"

### $ git commit –m “your message about the commit”

### The message at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo.

### C:\Users\Vishnu\Desktop\gitdoc\Capture5.PNG

### Step 7 : Create a new branch

### Branches allow you to move back and forth between 'states' of a project. For instance, if you want to add a new page to your website you can create a new branch just for that page without affecting the main part of the project. Once you're done with the page, you can [****merge****](http://git-scm.com/docs/git-merge) your changes from your branch into the master branch. When you create a new branch, Git keeps track of which commit your branch 'branched' off of, so it knows the history behind all the files.

### To create new branch and move to that branch:

### $ [git checkout -b <my branch name>](http://git-scm.com/docs/git-checkout)

### **Above command is shorthand for this,**

### ****$ git branch <my branch name>****

### ****$ git checkout <my branch name>****

### **To confirm that the new branch is created:**

### ****$ git branch****

### The branch name with the asterisk next to it indicates which branch you're pointed to at that given time.

### C:\Users\Vishnu\Desktop\gitdoc\Capture6.PNG

### Now, if you switch back to the master branch and make some more commits, your new branch won't see any of those changes until you [**merge**](http://git-scm.com/docs/git-merge) those changes onto your new branch.

### To merge changes made in new branch, you need to commit the changes on new branch, checkout to master branch and merge the new branch with the master branch.

### $ git commit –a –m ‘changes in new branch’

### git commit –a adds the files to staging area and commits them.

### $ git checkout master

### $ git merge my new branch

### Now since the use of new branch is over, it can be deleted.

### $ git branch –d my new branch

### Step 8: Create a new repository on GitHub

If you only want to keep track of your code locally, you don't need to use GitHub. But if you want to work with a team, you can use GitHub to collaboratively modify the project's code.

### To create a new repo on GitHub, log in and go to the GitHub home page. You should see a green '+ New repository' button:

### C:\Users\Vishnu\Desktop\gitdoc\Capture7.PNG

### After clicking the button, GitHub will ask you to name your repo and provide a brief description:

### C:\Users\Vishnu\Desktop\gitdoc\Capture8.PNG

When you're done filling out the information, press the 'Create repository' button to make your new repo.

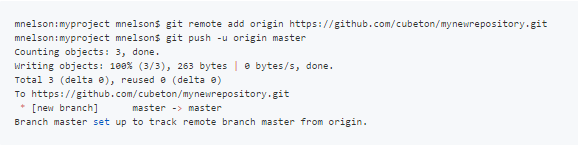
GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, we have already created a new repo locally, we want to push that onto GitHub so follow the **'....or push an existing repository from the command line'** section:

**$ git remote add origin URL** \*URL will be given to us when we create a new repository

**$ git push –u origin master**

We can also use :

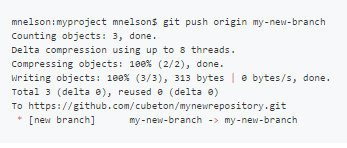
**git push git@github.com:git/git.git master \*(not used much)**

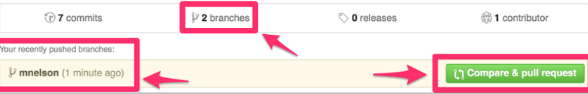


### Step 9: Push a branch to GitHub

Now we can **push** the commit in your branch to our new GitHub repo. This allows other people to see the changes we've made. If they're approved by the repository's owner, the changes can then be merged into the master branch. GitHub will automatically create the branch on the remote repository:

**$ git push origin <your branch name>**





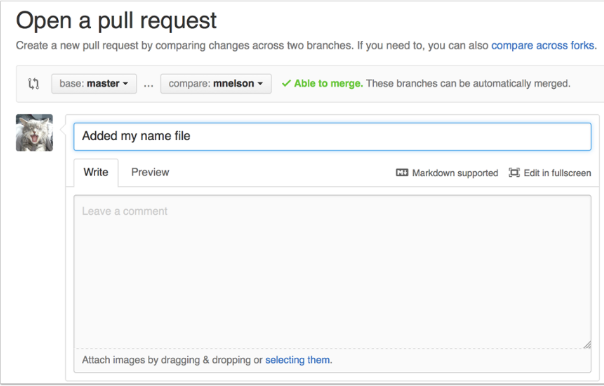
You can see in the screenshot, there are two branches, one “Master” and the other

“my new branch”.

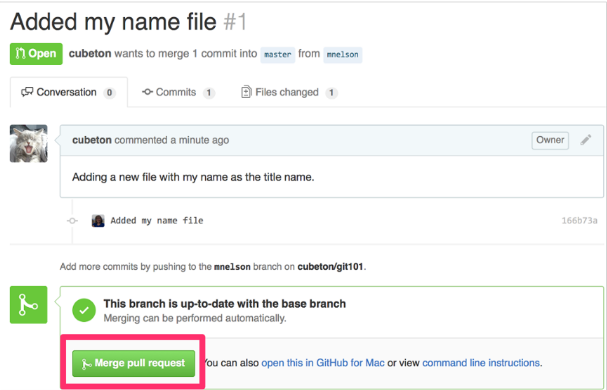
### Step 9: Create a Pull Request (PR)

A pull request (or PR) is a way to alert a repo's owners that you want to make some changes to their code. It allows them to review the code and make sure it looks good before putting your changes on the master branch.

This is what the PR page looks like before you've submitted it:



And this is what it looks like once you've submitted the PR request:



You might see a big green button at the bottom that says 'Merge pull request'. Clicking this means you'll merge your changes into the master branch.

Note that this button won't always be green. In some cases it'll be grey, which means you're faced with a **merge conflict**. This is when there is a change in one file that conflicts with a change in another file and git can't figure out which version to use. You'll have to manually go in and tell git which version to use.

Sometimes you'll be a co-owner or the sole owner of a repo, in which case you may not need to create a PR to merge your changes. However, it's still a good idea to make one so you can keep a more complete history of your updates and to make sure you always create a new branch when making changes.

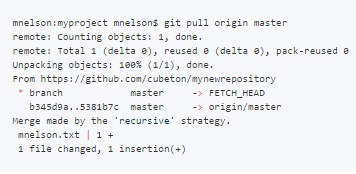
### Step 10: Merge a PR

### Go ahead and click the green 'Merge pull request' button. This will merge your changes into the master branch.

### Step 11: Get changes on GitHub back to your computer

The repo on GitHub looks a little different than what you have on your local machine. For example, the commit you made in your branch and merged into the master branch doesn't exist in the master branch on your local machine.To get those changes, use the command,

**$ git pull origin master** (when working on master branch)



Now we can use the [**git log**](http://git-scm.com/docs/git-log) command again to see all new commits.

**$ git log**

**List of github commands with their use:**

1. **$ git init :** to initialize the git repository
2. **git –version :** to find the version of git installed
3. **$ git config –global user.name “your\_name” :** to configure your name
4. **$ git config –global user.email** [**your\_name@example.com**](mailto:your_name@example.com) **:** to configure your email id, hence further it will be easy to track who made what changes, while working in a team.
5. **$ git status :** to check the status of files, i.e which files are ready to be commited and which are untracked
6. **$ git add –a :** It will bring all file to staging area
7. **$ git add <filename> :** It will add particular file to staging area
8. **$ git reset <file name> :** It removes the file from staging area
9. **$ git commit –m “first commit” :** It commits all the files in staging area and a record is maintained for commit.
10. **$ git branch < new branch name> :** create a new branch
11. **$ git checkout <new branch name> :** move to the new branch
12. **$ git checkout –b <new branch name> :** create a new branch and move to that branch.
13. **$ git checkout master ; $ git merge <branch name> :** It will merge the branch with master branch

**$ git remote add origin URL :** To create the remote repository for already existing local repository, URL is given by github when we create repository

**$ git push origin master :** push master to remote repository

$ **git push origin <your branch name> :** push a branch to remote repository, so that others can see the changes.

**$ git pull origin master :** To get the changes done on remote repository back to the local repository

$ **git log** : To maintain the log of all new commits

### C:\Users\Vishnu\Desktop\gitdoc\Capture4.PNG

### We can see that changes are not yet commited, they are about to be commited.