

GIT

Training Material

GIT

A version control system (VCS) allows you to track the history of a collection of files. It supports creating different versions of this collection. Each version captures a snapshot of the files at a certain point in time and the VCS allows you to switch between these versions. These versions are stored in a specific place, typically called a repository

GIT is currently the most popular implementation of a distributed version control system

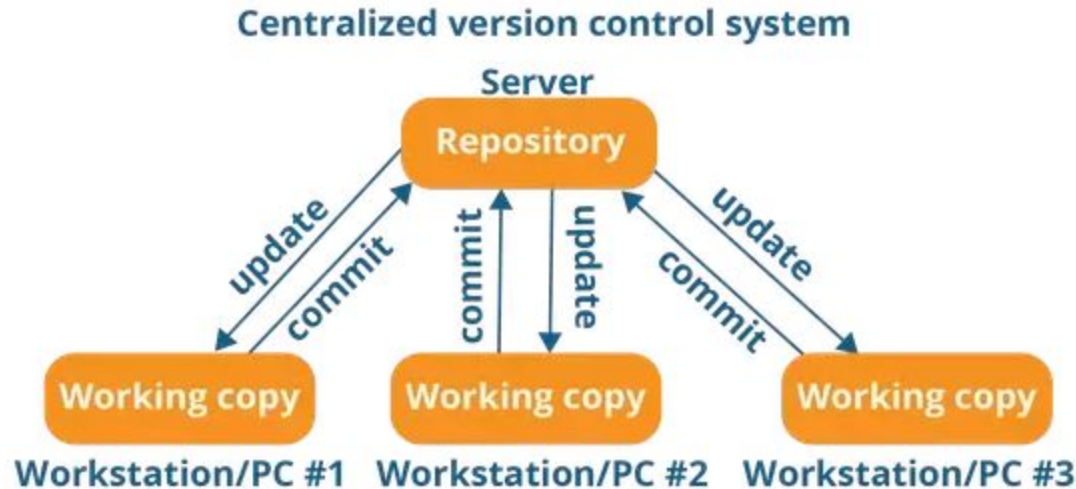
DISTRIBUTED VS. CENTRALIZED

The main difference between the two classes is that Centralized VCSs keep the history of changes on a central server from which everyone requests the latest version of the work and pushes the latest changes to. This means that everyone sharing the server also shares everyone's work. Sourceforge.net uses this type of versioning in their projects.

On the other hand, on a Distributed VCS, everyone has a local copy of the entire work's history. This means that it is not necessary to be online to change revisions or add changes to the work. "Distributed" comes from the fact that there isn't a central entity in charge of the work's history, so that anyone can sync with any other team member. This helps avoid failure due to a crash of the central versioning server

Centralized VCS

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server



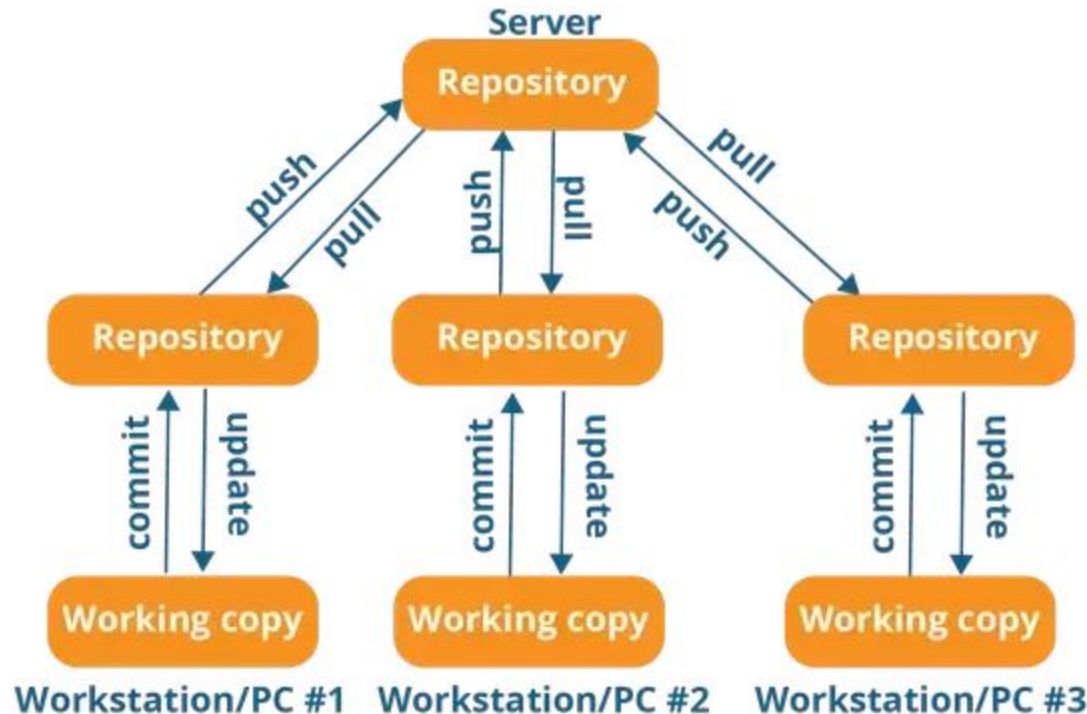
Every programmer can extract or update their workstations with the data present in the repository or can make changes to the data or commit in the repository. Every operation is performed directly on the repository.

It is not locally available; meaning you always need to be connected to a network to perform any action.

Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

Distributed VCS

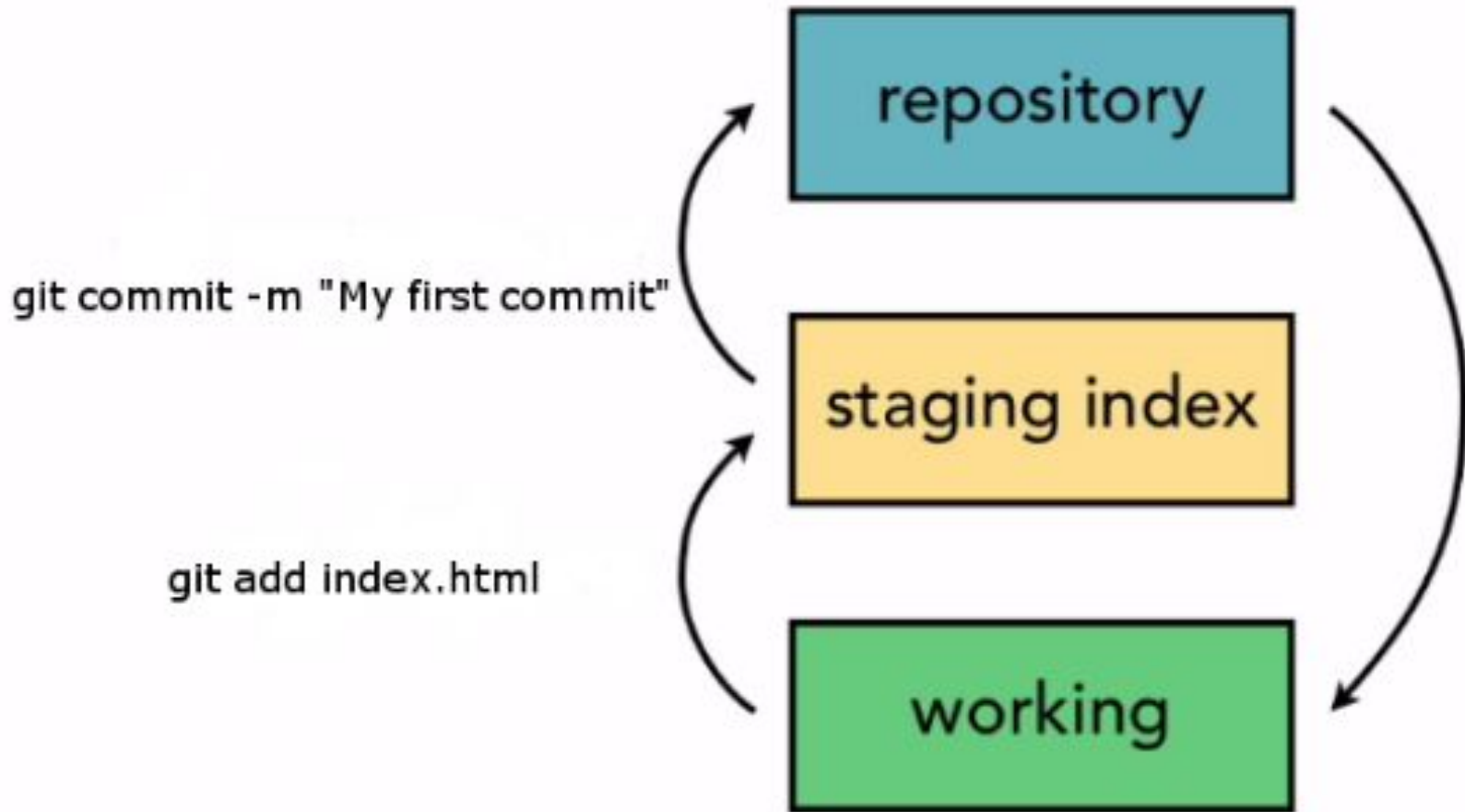
In Distributed VCS, every contributor has a local copy or “clone” of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.



They can update their local repositories with new data from the central server by an operation called **“Pull”** and affect changes to the main repository by an operation called **“Push”** from their local repository.

GIT ARCHITECTURE

three-tree architecture



GIT Installation

</SOURCEFUSE>

Ubuntu, Debian and derived systems

```
sudo apt-get install git -y
```

RedHat, CentOS and Fedora

```
sudo yum install git -y
```

GIT Configuration

```
git config --global user.name "Firstname Lastname"  
git config --global user.email "your.email@example.org"
```

Creating Repository through CLI

```
[root@localhost ~]# mkdir devops  
[root@localhost ~]# cd devops  
[root@localhost devops]# git init  
Initialized empty Git repository in /root/devops/.git/  
[root@localhost devops]#
```

git init is used to initialize the **git** on a project that's not under **git**.

Our local repository consists of three "trees" maintained by git. The first one is your Working Directory which holds the actual files. the second one is the Index which acts as a staging area and finally the HEAD which points to the last commit you've made.

```
|— HEAD
|— branches
|— config
|— description
|— hooks
|   |— pre-commit.sample
|   |— pre-push.sample
|   └─ ...
|— info
|   └─ exclude
|— objects
|   |— info
|   └─ pack
└─ refs
    |— heads
    └─ tags
```

File/directory	Type	Value	Meaning
config	text file	[See diagram]	The configuration file for the local git repository
HEAD	text file	ref: refs/heads/master	Lists a file to read that is the current HEAD branch: git branch will show the branch that HEAD refers to as the current branch.
refs	directory		Everything under the refs directory is references to a commit for a branch of some type (either a local branch or a remote tracking branch).
refs/heads	directory		Files in the refs/heads directory are branch names. For example a file named refs/heads/master means a branch named master exists in the local repository. The contents of the file is the hash of the most recent commit on that branch.
refs/heads/master	text file	6975b25be537f997c92490c7927cb6886fd5c49f	The most recent commit on this branch (master)
refs/heads/v1	text file	fd70e1d85320d12f230d8ff5e6056d86a775a6f3	The most recent commit on this branch (v1)
refs/remotes	directory	-	Everything under the refs/remotes directory is references to a commit for a remote-tracking branch.
refs/remotes/origin	directory	-	The remote tracking branches for the remote repository origin are stored in this directory.
refs/remotes/origin/v0	text file	e018ab5fe997a56bffb39350543c017684cfd721	The most recent commit on this branch. Note the hash is the different than the hash in in refs/heads/v1 which means the user's local v1 branch is behind the commit in the remote-tracking branch's v1 branch. The user needs to merge the commits from this remote-tracking branch into the user's v1 branch to catch up to the remote-tracking branch.

Bare git Repository

A bare repository created with **git init --bare** is for... sharing. If you are collaborating with a team of developers, and need a place to share changes to a repo, then you will want to create a bare repository in centralized place where all users can push their changes (often the easy choice is github.com). Because git is a distributed version control system, no one will directly edit files in the shared centralized repository. Instead developers will clone the shared bare repo, make changes locally in their working copies of the repo, then push back to the shared bare repo to make their changes available to other users.

```
File Edit View Search Terminal Help
sundarra@sundarra:~$ git --bare init
Initialized empty Git repository in /home/sundarra/
sundarra@sundarra:~$ ls
branches      description  Documents  examples.desktop  gitdemo1  gitmergedemo  gitstash1  HEAD  mergeconflict  Pictures  refs  Templates
cherrypickdemo desktop      downloads  gitcommands.odt   gitdiff   gitrebasedemo gitstashdemo hooks  Music      Public  snap  Videos
config        Desktop     Downloads  gitdemo           gitdiff1  gitref        gitstashexample info  objects  Python-3.9.2.tar.xz tagexample
sundarra@sundarra:~$
```

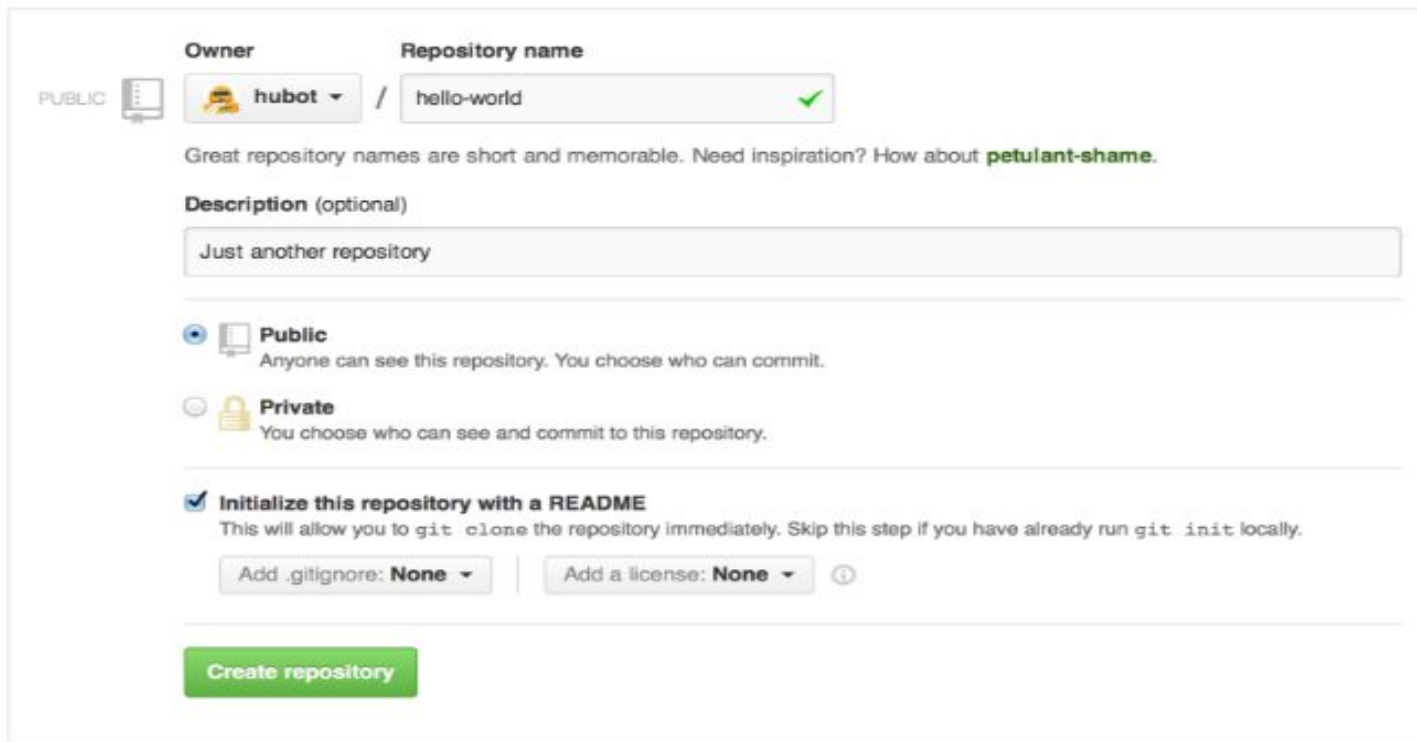
```
[root@ip-172-31-21-66 barerepo]# git add .
fatal: This operation must be run in a work tree
```

We use a working directory created with **git init** or **git clone** when I want to add, edit and delete files in myproject locally on my dev machine.

When we ready, we share my local changes with a **git push** to a bare repository myproject.git (usually on a remote server like github.com) so other developers can access my local changes.

To create a new repository

1. In the upper right corner, next to your avatar or identicon, click + and then select **New repository**.
2. Name your repository `hello-world`.
3. Write a short description.
4. Select **Initialize this repository with a README**.



The screenshot shows the GitHub 'Create new repository' form. At the top, there's a 'PUBLIC' label with a lock icon. Below it, the 'Owner' is set to 'hubot' with a dropdown arrow. The 'Repository name' is 'hello-world' with a green checkmark. A hint text says: 'Great repository names are short and memorable. Need inspiration? How about **petulant-shame**.' Below this is a 'Description (optional)' text area containing 'Just another repository'. Underneath is a section for visibility: 'Public' (selected with a radio button) and 'Private' (unselected). The 'Public' option has a subtext: 'Anyone can see this repository. You choose who can commit.' The 'Private' option has a subtext: 'You choose who can see and commit to this repository.' Below this is a checked checkbox for 'Initialize this repository with a README' with a subtext: 'This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.' At the bottom of this section are two dropdowns: 'Add .gitignore: None' and 'Add a license: None' with an information icon. At the very bottom is a large green 'Create repository' button.

Owner: hubot / Repository name: hello-world ✓

Great repository names are short and memorable. Need inspiration? How about **petulant-shame**.

Description (optional): Just another 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 allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: None | Add a license: None ⓘ

Create repository

</SOURCE**FUSE**>

Create GIT HUB Repository through CLI

```
curl -u 'USER' https://api.github.com/user/repos -d '{"name":"REPO"}'
```

USER : User name of github

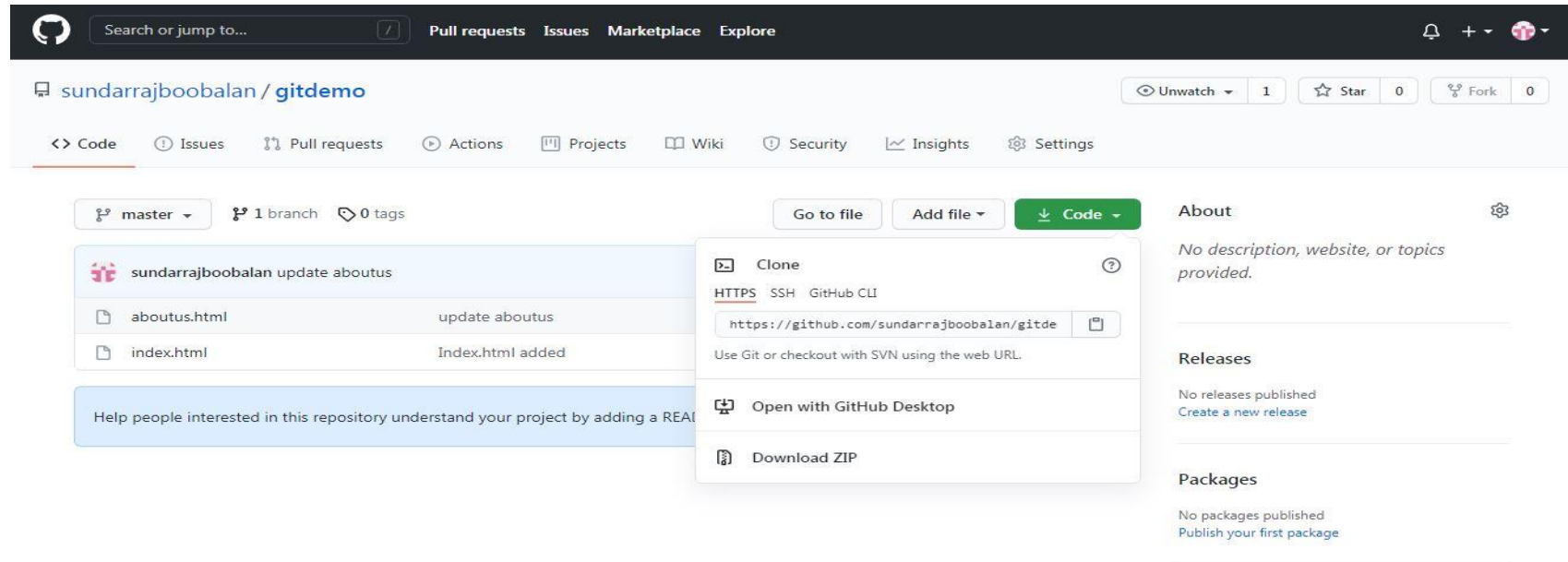
REPO : Repository name

Ignore the files which don't want to push to GITHUB

.gitignore

1. Create **.gitignore** file in the local repository
2. Open the **.gitignore** file add the files which don't want to push to GIT HUB
3. Save the file and exit
4. Push the repository to GIT HUB (**git push origin master**)

GIT Clone using HTTPS



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

sundarraajboobalan / gitdemo

Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags

sundarraajboobalan update aboutus

aboutus.html update aboutus

index.html Index.html added

Help people interested in this repository understand your project by adding a README file.

Go to file Add file Code

Clone ?

HTTPS SSH GitHub CLI

https://github.com/sundarraajboobalan/gitdemo

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

About

No description, website, or topics provided.

Releases

No releases published

Create a new release

Packages

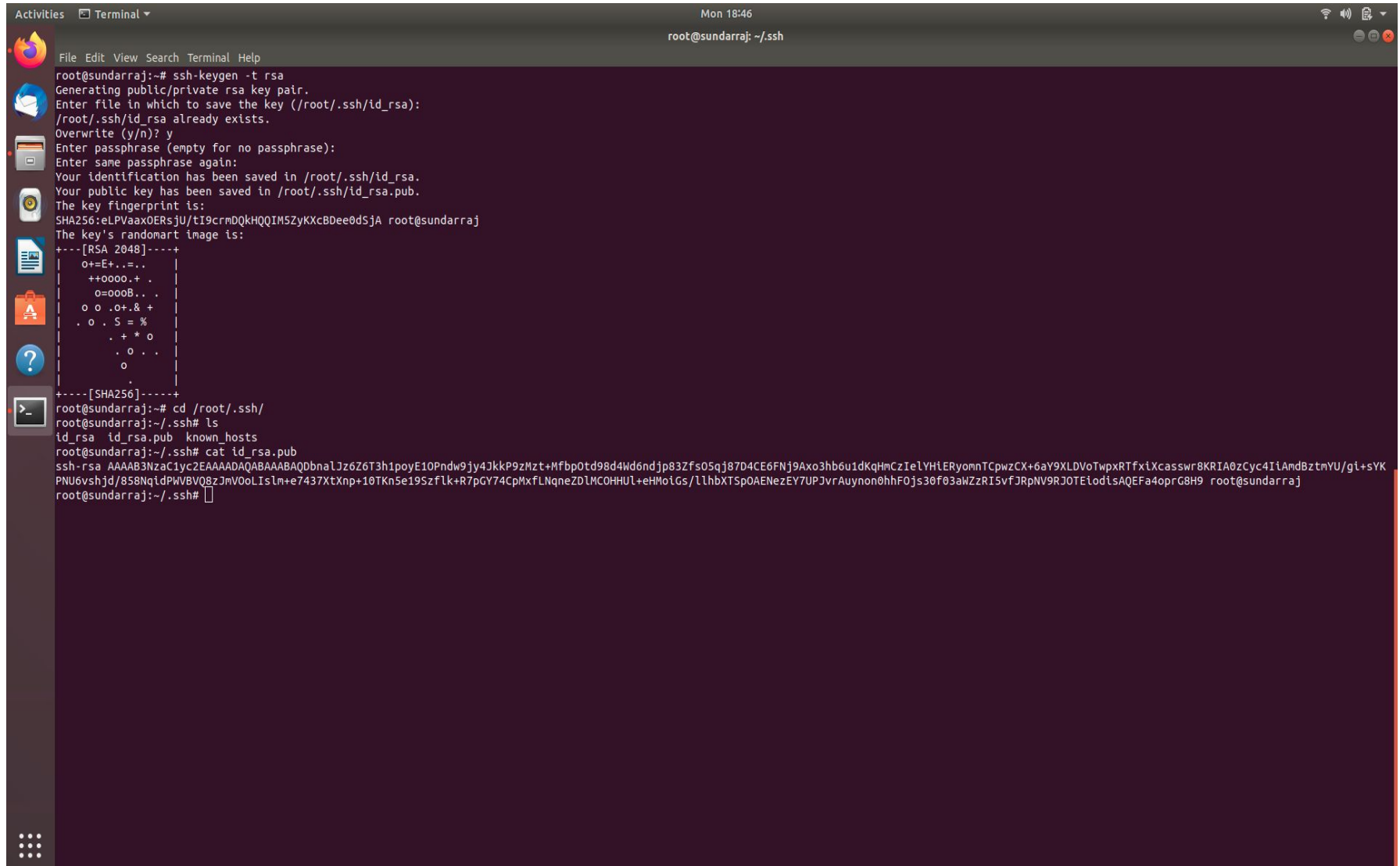
No packages published

Publish your first package

```
Dell@Dell-PC MINGW64 ~/gitclonedemo (master)
$ git clone https://github.com/sundarraajboobalan/gitdemo.git
Cloning into 'gitdemo'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 12 (delta 2), reused 5 (delta 0), pack-reused 0
Receiving objects: 100% (12/12), done.
Resolving deltas: 100% (2/2), done.
```

```
Dell@Dell-PC MINGW64 ~/gitclonedemo (master)
$
```

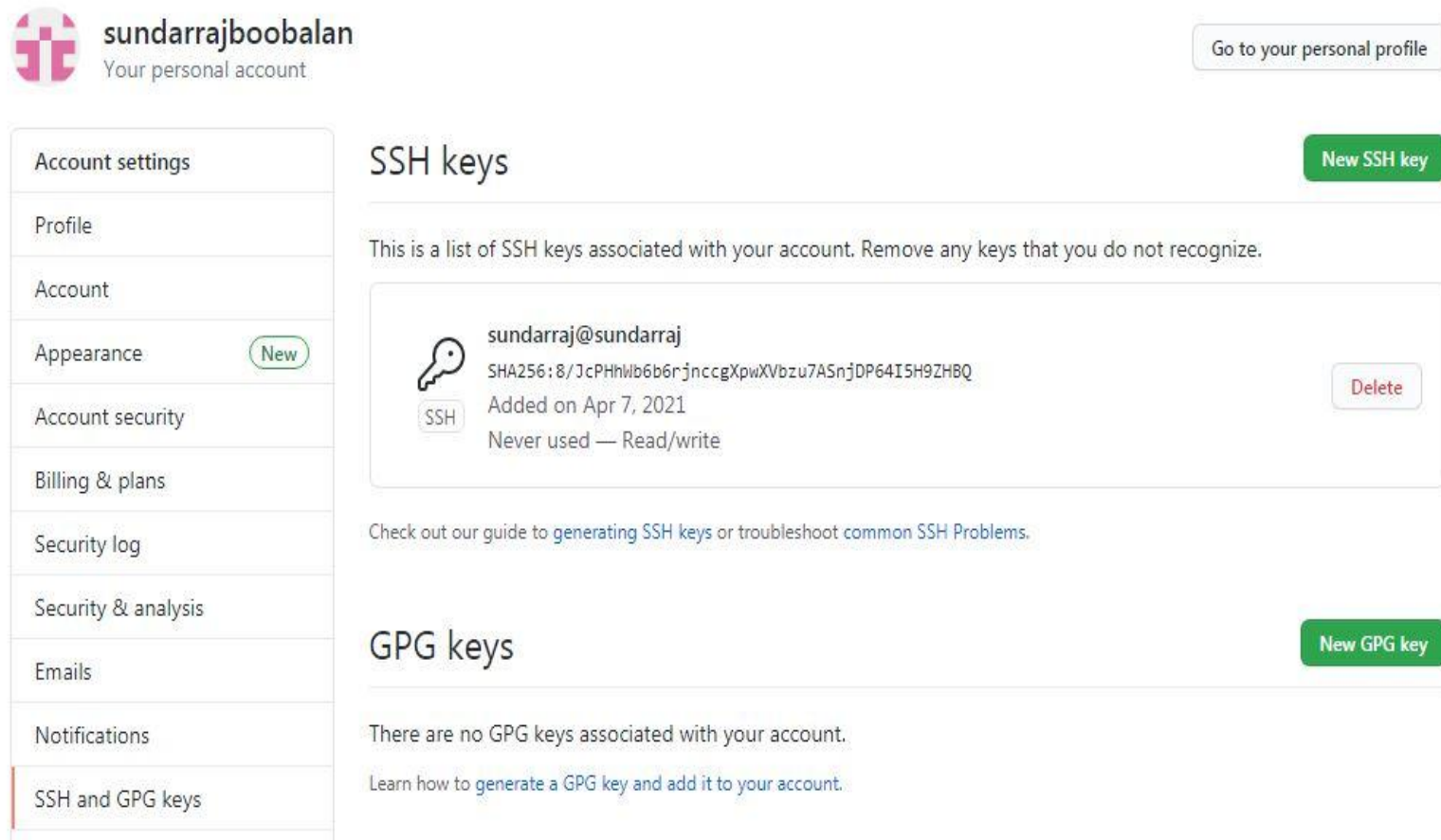
GENERATING SSH KEY



```
Activities Terminal Mon 18:46 root@sundarraj: ~/.ssh
File Edit View Search Terminal Help
root@sundarraj:~# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:eLPVaax0ERsJU/tI9crnDQkHQqIH5ZyKXcBDee0dSJA root@sundarraj
The key's randomart image is:
+---[RSA 2048]-----+
| O+=E+..=.. |
| ++0000.+ . |
| o=000B.. |
| o o .o+.& + |
| . o . S = % |
| . + * o |
| . o . |
| o |
| . |
+---[SHA256]-----+
root@sundarraj:~# cd /root/.ssh/
root@sundarraj:~/.ssh# ls
id_rsa id_rsa.pub known_hosts
root@sundarraj:~/.ssh# cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDbnalJz6Z6T3h1poyE10Pndw9jy4JkkP9zMzt+Mfbp0td98d4Wd6ndjp83Zfs05qj87D4CE6FNj9Axo3hb6u1dKqHmCzIeLYHlERyomnTCpwzCX+6aY9XLDVoTwpxRTfxiXcasswr8KRIA0zCyc4IiAmdBztmYU/gI+sYK
PNU6vshjd/858NgidPWVBVQ8zJmV0oLIsln+e7437XtXnp+10TKn5e19SzfLk+R7pCY74CpMxflNqneZDLmCOHHUL+eHMoIGs/lLhbXTSpOAEneZy7UPJvAuynon0hhF0js30f03aWZzRI5vfJRpNV9RJ0TEiodisAQEFa4oprG8H9 root@sundarraj
root@sundarraj:~/.ssh#
```

GIT Clone using SSH

1. Create Private key and Public key using SSH-KEYGEN command on the development machine for the user
2. Log in to GIT HUB , go to user settings then select SSH an GPG keys option
3. Click on New SSH Key and add the public key



The screenshot shows the GitHub user settings page for the user 'sundarraajboobalan'. The page has a sidebar on the left with navigation links: Account settings, Profile, Account, Appearance (marked with a 'New' badge), Account security, Billing & plans, Security log, Security & analysis, Emails, Notifications, and SSH and GPG keys (which is highlighted with a red bar). The main content area is titled 'SSH keys' and includes a 'New SSH key' button. Below the title, it states: 'This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.' There is one SSH key listed for the email 'sundarraaj@sundarraaj' with the SHA256 fingerprint '8/JcPHhVb6b6rjnccgXpwXVbzu7ASnjDP64I5H9ZHBQ'. It was added on Apr 7, 2021, and has permissions for 'Read/write'. A 'Delete' button is next to the key. Below the key list, there is a link to 'Check out our guide to generating SSH keys or troubleshoot common SSH Problems.' The 'GPG keys' section below it shows 'There are no GPG keys associated with your account.' and a link to 'Learn how to generate a GPG key and add it to your account.' A 'New GPG key' button is also present.

sundarraajboobalan
Your personal account

Go to your personal profile

Account settings
Profile
Account
Appearance **New**
Account security
Billing & plans
Security log
Security & analysis
Emails
Notifications
SSH and GPG keys

SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Key	SHA256	Added on	Permissions	Action
sundarraaj@sundarraaj	8/JcPHhVb6b6rjnccgXpwXVbzu7ASnjDP64I5H9ZHBQ	Apr 7, 2021	Never used — Read/write	Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

GPG keys

New GPG key

There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).

sundarrajbobalan / gitdemo

Unwatch 1

Star 0

Fork 0

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

master 1 branch 0 tags



sundarrajbobalan update aboutus



aboutus.html

update aboutus



index.html

Index.html added

Help people interested in this repository understand your project by adding a README file.

Go to file

Add file

Code

Clone

HTTPS SSH GitHub CLI

git@github.com:sundarrajbobalan/gitdemo.git

Use a password-protected SSH key.



Open with GitHub Desktop



Download ZIP

About

No description, website, or topics provided.

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

```
Dell@Dell-PC MINGW64 ~/sshdemo (master)
```

```
$ git clone git@github.com:sundarrajbobalan/gitdemo.git
Cloning into 'gitdemo'...
```


Remote Path :

</SOURCE FUSE>

git remote -v : to view pull and push path

```
sundarrajsundarraj:~$ git clone https://github.com/sundarrajboobalan/DemoGitClone.git
Cloning into 'DemoGitClone'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
sundarrajsundarraj:~$ git remote -v
sundarrajsundarraj:~$ git remote add origin https://github.com/sundarrajboobalan/DemoGitClone.git
sundarrajsundarraj:~$ git remote -v
origin https://github.com/sundarrajboobalan/DemoGitClone.git (fetch)
origin https://github.com/sundarrajboobalan/DemoGitClone.git (push)
sundarrajsundarraj:~$
```

Adding Remote Path :

git remote add <name> path

```
root@sundarraj:~$ git clone https://github.com/sundarrajboobalan/DemoGitClone.git
Cloning into 'DemoGitClone'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
sundarrajsundarraj:~$ git remote -v
sundarrajsundarraj:~$ git remote add origin https://github.com/sundarrajboobalan/DemoGitClone.git
sundarrajsundarraj:~$ git remote -v
origin https://github.com/sundarrajboobalan/DemoGitClone.git (fetch)
origin https://github.com/sundarrajboobalan/DemoGitClone.git (push)
sundarrajsundarraj:~$ git remote add sourcefuserepo https://github.com/sundarrajboobalan/DemoGitClone.git
sundarrajsundarraj:~$ git remote -v
origin https://github.com/sundarrajboobalan/DemoGitClone.git (fetch)
origin https://github.com/sundarrajboobalan/DemoGitClone.git (push)
sourcefuserepo https://github.com/sundarrajboobalan/DemoGitClone.git (fetch)
sourcefuserepo https://github.com/sundarrajboobalan/DemoGitClone.git (push)
sundarrajsundarraj:~$
```

Renaming Remote Name :

git remote rename <old name> <new Name>

```
root@sundarraj:~$ git remote rename sourcefuserepo sourcefuserepo1 https://github.com/sundarrajboobalan/DemoGitClone.git
usage: git remote rename <old> <new>

sundarrajsundarraj:~$ git remote -v
origin https://github.com/sundarrajboobalan/DemoGitClone.git (fetch)
origin https://github.com/sundarrajboobalan/DemoGitClone.git (push)
sourcefuserepo https://github.com/sundarrajboobalan/DemoGitClone.git (fetch)
sourcefuserepo https://github.com/sundarrajboobalan/DemoGitClone.git (push)
sundarrajsundarraj:~$ git remote rename sourcefuserepo sourcefuserepo1
sundarrajsundarraj:~$ git remote -v
origin https://github.com/sundarrajboobalan/DemoGitClone.git (fetch)
origin https://github.com/sundarrajboobalan/DemoGitClone.git (push)
sourcefuserepo1 https://github.com/sundarrajboobalan/DemoGitClone.git (fetch)
sourcefuserepo1 https://github.com/sundarrajboobalan/DemoGitClone.git (push)
sundarrajsundarraj:~$
```


Change Remote Path :

</SOURCE FUSE>

Git remote set-url <remote name> path

```
[root@ip-172-31-21-66 JavaProject]# git remote set-url samplerepo git@github.com:raknas999/JavaProj.git
[root@ip-172-31-21-66 JavaProject]# git remote -v
```

Delete Remote Path :

git remote rm <remote name>

```
[root@ip-172-31-21-66 JavaProject]# git remote rm samplerepo
```

Add or Modify data :

```
[root@ip-172-31-21-66 JavaProj]# touch sample.txt
[root@ip-172-31-21-66 JavaProj]# git status
On branch master
Your branch is up-to-date with 'origin/master'.

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

    sample.txt

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

</SOURCE FUSE>

Untracked: the file is not tracked by the Git repository. This means that the file never staged nor committed.

Git add :

Git add file-name (to add particular file)

Git add . (to add all the files)

```
[root@ip-172-31-21-66 JavaProj]# git add .
```

```
[root@ip-172-31-21-66 JavaProj]# git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   sample.txt
```

Git commit :

Git commit -m "message"

Or

Git commit -am "message" (in a single shot add and commit will be done)

After every commit new unique SHA value generated

```
[root@ip-172-31-21-66 JavaProj]# git commit -m "sample file created"
[master 2f509b4] sample file created
```

untracked: the file is not tracked by the Git repository. This means that the file never staged nor committed

```
[root@ip-172-31-21-66 JavaProj]# git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Git Push with HTTPS : Push the changes to remote repository (Git Hub)

If the repository is cloned using HTTPS then it will ask user name and password at the time of pushing the change to remote repo.

```
root@sundarraj:~/samplerepo# git push origin master
Username for 'https://github.com': sundarrajboobalan
Password for 'https://sundarrajboobalan@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 246 bytes | 246.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/sundarrajboobalan/DemoGitClone/pull/new/master
remote:
To https://github.com/sundarrajboobalan/DemoGitClone.git
 * [new branch]   master -> master
root@sundarraj:~/samplerepo#
```

2 commits
1 branch
0 releases
1 contributor

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

This branch is 1 commit ahead of rchelikani:master.
Pull request
Compare

root sample file created
Latest commit 2f509b4 7 minutes ago

README.md	Initial commit	9 days ago
sample.txt	sample file created	7 minutes ago

Git Push with SSH : Push the changes to remote repository (Git Hub)

If the repository is cloned using SSH, it will push the changes to remote repo without asking user name and password

```
root@sundarraj:~/samplerepo# git push origin master
Username for 'https://github.com': sundarrajboobalan
Password for 'https://sundarrajboobalan@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 246 bytes | 246.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/sundarrajboobalan/DemoGitClone/pull/new/master
remote:
To https://github.com/sundarrajboobalan/DemoGitClone.git
 * [new branch]      master -> master
root@sundarraj:~/samplerepo#
```

Git Pull : git pull runs **git fetch** with the given parameters and calls **git merge** to merge the retrieved branch heads into the current branch.

3 commits

1 branch

0 releases

1 contributor

Branch: master ▾

New pull request


Create new file

Upload files

Find file

Clone or download ▾

This branch is 2 commits ahead of rchelikani:master.

[Pull request](#) [Compare](#) root sshfile created

Latest commit 31f6632 4 minutes ago

 README.md

Initial commit

9 days ago

 sample.txt

sample file created

14 minutes ago

 sshfile.txt

sshfile created

4 minutes ago

Git Diff : Compare the changes (between branches, commits, local and remote)


```
[root@ip-172-31-21-66 JavaProj]# vi sample.txt
[root@ip-172-31-21-66 JavaProj]# git diff
diff --git a/sample.txt b/sample.txt
index e69de29..a391866 100644
--- a/sample.txt
+++ b/sample.txt
@@ -0,0 +1 @@
+First line added
```


</SOURCE FUSE>

sundarrajboobalan / gitdemo Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master gitdemo / aboutus.html Go to file ...

 sundarrajboobalan update aboutus Latest commit 8f8e5a9 6 hours ago History

 1 contributor

9 lines (8 sloc) | 154 Bytes Raw Blame Download Edit Delete

```
1 <html>
2 <head>
3 <title> Git push demo </title>
4 <P> this is demo for paragraph</P>
5 <h1> example for heading1</h1>
6 <body bgcolor="red">
7 </head>
8 </html>
```

```
* [new branch]      master -> master
root@sundarraj:~/samplerepo# git fetch
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/sundarrajboobalan/DenoGitClone
* [new branch]      main -> origin/main
root@sundarraj:~/samplerepo#
```


Gitfetch : **Fetch** branches and/or tags (collectively, "refs") from one or more other repositories, along with the objects necessary to complete their histories but not merged with local repository

```
* [new branch]      master -> master
root@sundarraj:~/samplerepo# git fetch
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/sundarrajboobalan/DenoGitClone
* [new branch]      main -> origin/main
root@sundarraj:~/samplerepo#
```

```
[root@ip-172-31-21-66 JavaProj]# cat sample.txt
First line added
```

Gitmerge : Merge the changes to local repository.

```
[root@ip-172-31-21-66 JavaProj]# git merge
Updating 945125b..5af7884
Fast-forward
 sample.txt | 1 +
 1 file changed, 1 insertion(+)
You have new mail in /var/mail/root
```

```
[root@ip-172-31-21-66 JavaProj]# cat sample.txt
First line added
Second line added
```

</SOURCE FUSE>

Git Log : To check the log information of Committed Files in local repo.

```
root@sundarraj:~/samplerepo# git log
commit 3caabe7b9bd5e6c1b5f955e5052447dd8c4f3443 (HEAD -> master, origin/master)
Author: sundarrajboobalan <sundarrajboobalan@gmail.com>
Date: Fri Apr 9 14:34:17 2021 +0530

    First commit
root@sundarraj:~/samplerepo#
```

Comparing difference between commits

```
[root@ip-172-31-21-66 JavaProj]# git diff 5af7884c4f1652 945125b5969be
diff --git a/sample.txt b/sample.txt
index 61dfe5e..a391866 100644
--- a/sample.txt
+++ b/sample.txt
@@ -1,2 +1 @@
    First line added
-Second line added
```


</SOURCE FUSE>

Git commit --amend : To modify the commit message

git commit --amend -m “message”

```
root@sundarraj:~/samplerepo# git commit -am "Second Commit"
[master 1142bcc] Second Commit
1 file changed, 2 insertions(+)
root@sundarraj:~/samplerepo#
```

```
root@sundarraj:~/samplerepo# git log
commit 3caabe7b9bd5e6c1b5f955e5052447dd8c4f3443 (HEAD -> master, origin/master)
Author: sundarrajboobalan <sundarrajboobalan@gmail.com>
Date: Fri Apr 9 14:34:17 2021 +0530

    First commit
root@sundarraj:~/samplerepo#
```

Git Branch :

Git supports branching which means that you can work on different versions of your collection of files. A branch allows the user to switch between these versions so that he can work on different changes independently from each other.

</SOURCE FUSE>

Create Branch :

git branch (branch name)

```
[root@ip-172-31-21-66 JavaProj]# git branch development
```

Verify Branch :

```
[root@ip-172-31-21-66 JavaProj]# git branch
development
* master
```

Switch to Branch :

git checkout branch

```
[root@ip-172-31-21-66 JavaProj]# git checkout development
Switched to branch 'development'
```

Modify and verify the changes in branch

```
[root@ip-172-31-21-66 JavaProj]# cat sample.txt
First line added
Second line added
Third line added thru development branch
```

Verify the changes in Master branch

```
[root@ip-172-31-21-66 JavaProj]# cat sample.txt
First line added
Second line added
```

</SOURCE FUSE>

Merging Branch :

git merge branch

```
[root@ip-172-31-21-66 JavaProj]# git merge development
Updating 470fdeb..a1f6b00
Fast-forward
 sample.txt | 1 +
 1 file changed, 1 insertion(+)
```

Delete Branch :

git branch -d <Branch Name>

git branch -D <Branch Name> (To delete forcefully)

```
[root@ip-172-31-21-66 JavaProj]# git branch
  development
* master
[root@ip-172-31-21-66 JavaProj]# git branch -d development
Deleted branch development (was a1f6b00).
[root@ip-172-31-21-66 JavaProj]# git branch
* master
```

</SOURCE FUSE>

Create Pull Request:

1. Update the local repository (git pull)

```
[root@ip-172-31-21-66 JavaProj]# git pull
Already up-to-date.
```

2. Create Branch and switch (git checkout -b branch (it will create and switch))

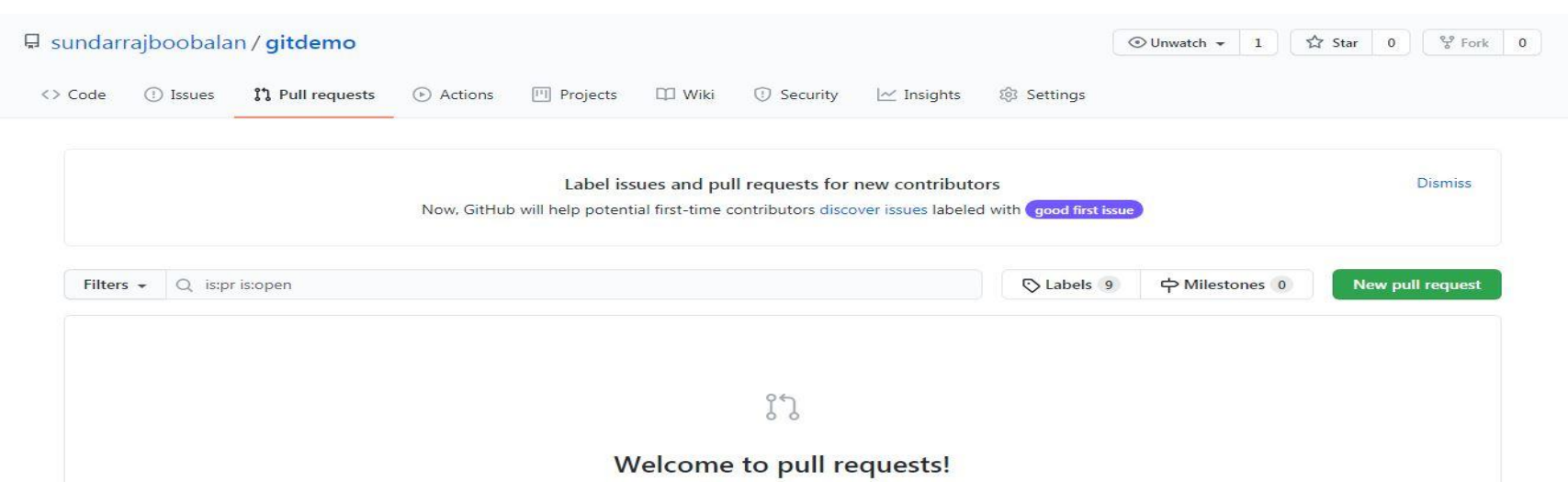
```
[root@ip-172-31-21-66 JavaProj]# git checkout -b feature
Switched to a new branch 'feature'
```

```
[root@ip-172-31-21-66 JavaProj]# git branch
* feature
master
```

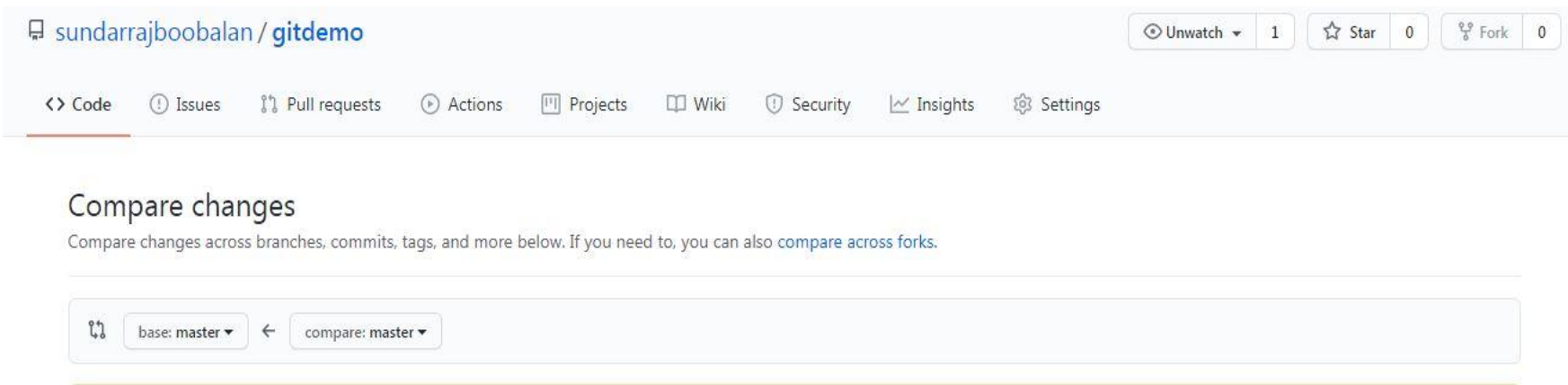
3. Push Branch to Remote (git push origin <branch name>)

```
root@sundarraj:~/samplerepo# git push origin master
Username for 'https://github.com': sundarrajboobalan
Password for 'https://sundarrajboobalan@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 246 bytes | 246.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/sundarrajboobalan/DenoGitClone/pull/new/master
remote:
To https://github.com/sundarrajboobalan/DenoGitClone.git
 * [new branch]      master -> master
root@sundarraj:~/samplerepo#
```

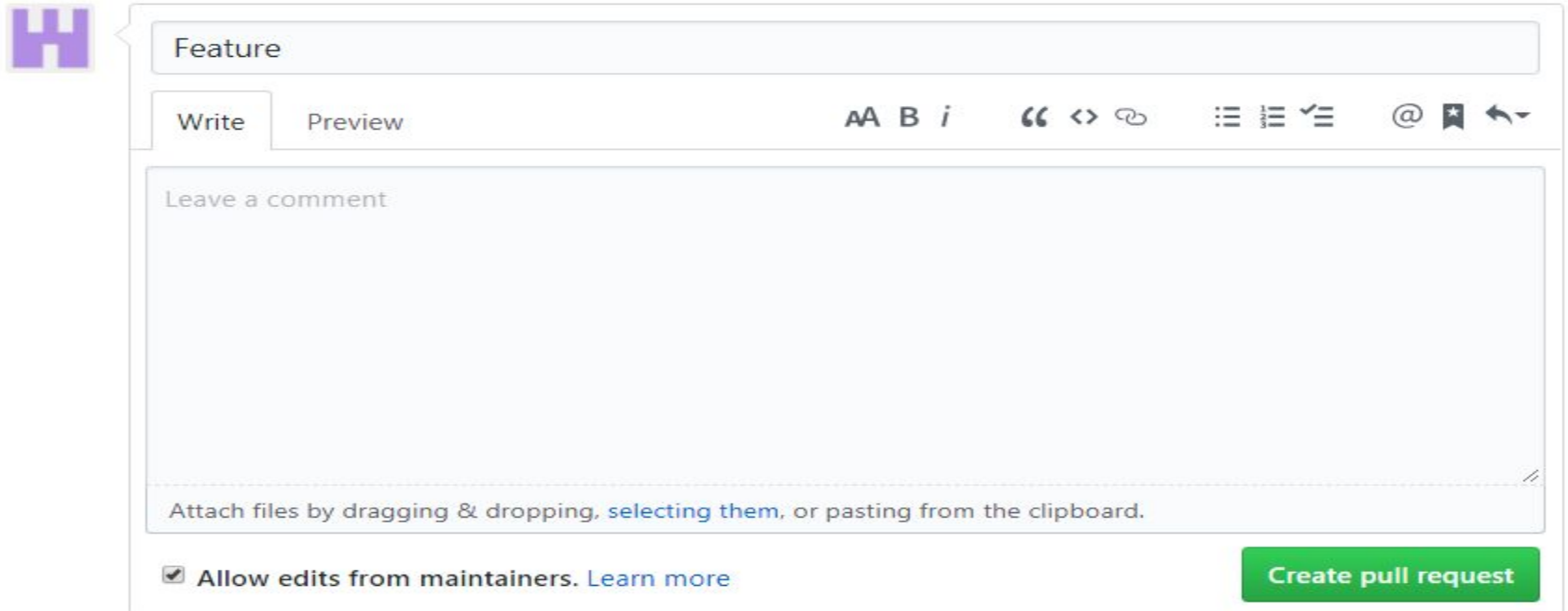
4. Verify in github, compare and pull request button will appear



5. Compare the changes by clicking the button



6. Create Pull request



The screenshot shows a web interface for creating a pull request. On the left is a purple icon with a white 'H'. The main area has a title field containing 'Feature'. Below the title are two tabs: 'Write' (active) and 'Preview'. To the right of the tabs is a rich text editor toolbar with icons for bold (AA), italic (i), bulleted list, numbered list, link, unlink, quote, code, and other formatting options. Below the toolbar is a large text area with the placeholder text 'Leave a comment'. At the bottom of the text area is a dashed line and the text 'Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.' Below this is a checkbox labeled 'Allow edits from maintainers.' with a blue link 'Learn more' to its right. In the bottom right corner is a green button with the text 'Create pull request'.

Feature

Write Preview

AA B i “ < > @ ★ ↶

Leave a comment

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

☒ Allow edits from maintainers. [Learn more](#)

Create pull request

7. Close pull request

Add more commits by pushing to the **feature** branch on `raknas999/JavaProj`.



This branch has no conflicts with the base branch

Only those with [write access](#) to this repository can merge pull requests.



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

Close pull request

Comment

8. Delete Branch



Closed with unmerged commits

This pull request is closed, but the `raknas999:feature` branch has unmerged commits.

Delete branch

</SOURCE FUSE>

Compare two branches:

```
[root@ip-172-31-21-66 JavaProj]# git diff feature master
diff --git a/sample.txt b/sample.txt
index 5283ed1..14abc2c 100644
--- a/sample.txt
+++ b/sample.txt
@@ -1,4 +1,3 @@
 First line added
 Second line added
 Third line added thru development branch
- Fourth line added thru feature branch
```

Branches merged with Master

git branch --merged master

```
[root@ip-172-31-21-66 JavaProj]# git branch
* feature
  master
[root@ip-172-31-21-66 JavaProj]# git branch --merged master
  master
```

Branches not merged with Master

git branch --no-merged master

```
[root@ip-172-31-21-66 JavaProj]# git branch
* feature
  master
[root@ip-172-31-21-66 JavaProj]# git branch --merged master
  master
[root@ip-172-31-21-66 JavaProj]# git branch --no-merged master
* feature
```


</SOURCE FUSE>

Remote Branches list :

git branch -r

```
[root@ip-172-31-21-66 JavaProj]# git branch -r
origin/HEAD -> origin/master
origin/feature
origin/master
```

All Branches list(local and remote) :

git branch -a

```
[root@ip-172-31-21-66 JavaProj]# git branch -a
* feature
master
remotes/origin/HEAD -> origin/master
remotes/origin/feature
remotes/origin/master
```

Rename Local Branch :

git branch -m (old name) (new name)

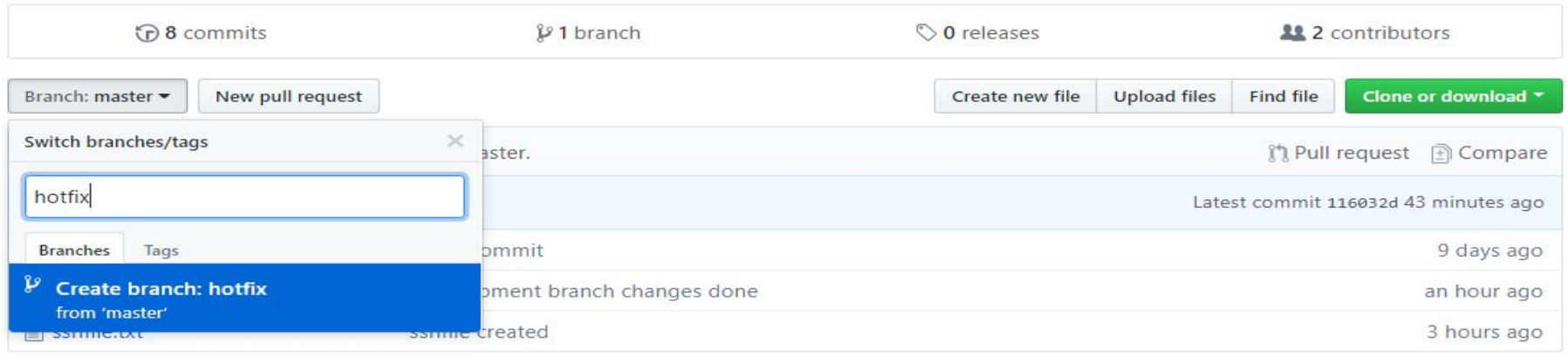
git branch -M (old name) (new name) -□ To rename forcefully

```
[root@ip-172-31-21-66 JavaProj]# git branch -m feature featurebr
```

```
[root@ip-172-31-21-66 JavaProj]# git branch -m feature featurebr
You have new mail in /var/spool/mail/root
[root@ip-172-31-21-66 JavaProj]# git branch
* featurebr
master
```

Rename Remote Branch :

1. Create Branch on GIT HUB



2. Update Local repository (git pull)

3. Rename the branch (git branch -m <old name> <new name>)

4. Delete the branch one Remote (git push origin :branch name)

5. Push the branch to remote (git push origin <branch name>)

Delete Branch Locally :

git branch -d branch name

git branch -D branch name (forceful delete)

Delete all the local Branches :

`git branch --merged | grep -v * | xargs git branch -D`

```
[root@ip-172-31-21-66 JavaProj]# git branch | grep -v \* | xargs git branch -D
Deleted branch featurebr (was 8ade03b).
Deleted branch hotfix (was 8ade03b).
```

Delete Remote Branch

`git push origin :<branch name>`

```
[root@ip-172-31-21-66 JavaProj]# git push origin :hotfix
```

git fetch origin --prune : this removes all obsolete local remote-tracking branches for any remote branches that no longer exist on the remote

```
[root@ip-172-31-21-66 JavaProj]# git fetch origin --prune
```

</SOURCE FUSE>

Tagging or Versioning

It has the ability to tag specific points in history as being important. Typically people use this functionality to mark release points (v1.0, and so on)

Tagging Types :

Git supports two types of tags: **lightweight** and **annotated**.

1. Lightweight Tags

git tag <tag name> (create tag on latest commit)

```
[root@ip-172-31-21-66 JavaProj]# git tag v1.0
```

Tags List :

```
[root@ip-172-31-21-66 JavaProj]# git tag  
v1.0
```

Create tag for particular commit :

git tag <tag name> <commit id>

```
[root@ip-172-31-21-66 JavaProj]# git tag v2.0 945125b5969b  
[root@ip-172-31-21-66 JavaProj]# git tag  
v1.0  
v2.0
```

</SOURCE FUSE>

To view the tag :

git show <tag version>

```
[root@ip-172-31-21-66 JavaProj]# git show v2.0
commit 945125b5969be396ffb347fb29a6a418b5c9da2d (tag: v2.0)
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sat Aug 25 15:53:03 2018 +0000

    first line added

diff --git a/sample.txt b/sample.txt
index e69de29..a391866 100644
--- a/sample.txt
+++ b/sample.txt
@@ -0,0 +1 @@
+First line added
```

Modify particular version :

git checkout <version>

```
[root@ip-172-31-21-66 JavaProj]# git checkout v2.0
Note: checking out 'v2.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

    git checkout -b <new-branch-name>

HEAD is now at 945125b... first line added
```

Create branch for the version

```
[root@ip-172-31-21-66 JavaProj]# git checkout -b v2.0-br  
Switched to a new branch 'v2.0-br'
```

Now the new branch will contain the particular version code

Push the tags to remote from local

git push origin <tag name>

```
[root@ip-172-31-21-66 JavaProj]# git push origin v1.0  
Total 0 (delta 0), reused 0 (delta 0)  
To github.com:raknas999/JavaProj.git  
* [new tag]          v1.0 -> v1.0
```

Push all the tags to remote from local

git push origin --tags

```
[root@ip-172-31-21-66 JavaProj]# git push origin --tags  
Total 0 (delta 0), reused 0 (delta 0)
```


Releases Tags

Draft a new release

3 hours ago

v2.0 ...

945125b zip tar.gz

3 hours ago

v1.2 ...

945125b zip tar.gz

2 hours ago

v1.0 ...

116032d zip tar.gz

Delete local tag :

git tag -d (tag name)

```
[root@ip-172-31-21-66 JavaProj]# git tag
v1.0
v1.2
v2.0
[root@ip-172-31-21-66 JavaProj]# git tag -d v1.0
Deleted tag 'v1.0' (was 116032d)
[root@ip-172-31-21-66 JavaProj]# git tag
v1.2
v2.0
```

</SOURCE FUSE>

Delete all the local tags :

`git tag -d $(git tag -l)`

```
[root@ip-172-31-21-66 JavaProj]# git tag -d $(git tag -l)
Deleted tag 'v1.2' (was 945125b)
Deleted tag 'v2.0' (was 945125b)
You have new mail in /var/spool/mail/root
[root@ip-172-31-21-66 JavaProj]# git tag
[root@ip-172-31-21-66 JavaProj]#
```

To view remote tags :

`git ls-remote --tags`

```
[root@ip-172-31-21-66 JavaProj]# git ls-remote --tags
```

Delete remote tag :

`git push origin :<tag name>`

```
[root@ip-172-31-21-66 JavaProj]# git push origin :v1.0
```


</SOURCE FUSE>

Delete all the remote tag :

```
git ls-remote --tags --refs origin | cut -f2 | xargs git push origin --delete
```

```
[root@ip-172-31-21-66 JavaProj]# git ls-remote --tags --refs origin | cut -f2 | xargs git push origin --delete
To github.com:raknas999/JavaProj.git
- [deleted]          v1.2
- [deleted]          v2.0
```

2. Annotated Tags :

```
git tag -a v1.4 -m "message"
```

```
[root@ip-172-31-21-66 JavaProj]# git tag -a v1.4 -m "my version 1.4"
You have new mail in /var/spool/mail/root
[root@ip-172-31-21-66 JavaProj]# git tag
v1.4
```

Undo the changes before add :

```
git checkout <file>
```

```
[root@ip-172-31-21-66 JavaProj]# git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working director

       modified:   sshfile.txt

no changes added to commit (use "git add" and/or "git commit -a")
[root@ip-172-31-21-66 JavaProj]# cat sshfile.txt
First line added in SSH File
You have new mail in /var/spool/mail/root
```

</SOURCE FUSE>

```
[root@ip-172-31-21-66 JavaProj]# git checkout sshfile.txt
[root@ip-172-31-21-66 JavaProj]# cat sshfile.txt
[root@ip-172-31-21-66 JavaProj]#
```

Undo the changes after add :

1. Do the changes
2. Add the file (git add .)
3. Git reset file

```
[root@ip-172-31-21-66 JavaProj]# cat sshfile.txt
First line added in SSH File
```

```
[root@ip-172-31-21-66 JavaProj]# git add .
You have new mail in /var/spool/mail/root
[root@ip-172-31-21-66 JavaProj]# git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   sshfile.txt
```

```
[root@ip-172-31-21-66 JavaProj]# git reset sshfile.txt
Unstaged changes after reset:
M       sshfile.txt
```

</SOURCE FUSE>

Undo the changes after commit :

Types of reset

1. git reset soft
2. git reset mixed (Default)
3. git reset hard

Git reset soft

1. Take the commit id which we want to keep
2. Apply the command (git reset --soft paste the Commit id)

```
commit 8847cb7957c7cfe9493d079ecf18d7cf25ac8cd8 (HEAD -> master)
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sat Aug 25 20:17:20 2018 +0000

    first line

commit 116032dac2111e51230d2ff82424e1deba58a33b (origin/master, origin/HEAD)
Merge: alf6b00 5af7884
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sat Aug 25 17:21:45 2018 +0000

    ssave

commit alf6b00686353c66be061fb8b8b375a9c47bd284
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sat Aug 25 17:09:41 2018 +0000

    Development branch changes done
```

```
[root@ip-172-31-21-66 JavaProj]# git reset --soft 116032dac211
commit 116032dac2111e51230d2ff82424e1deba58a33b (HEAD -> master, origin/master, origin/HEAD)
Merge: a1f6b00 5af7884
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sat Aug 25 17:21:45 2018 +0000

    ssave

commit a1f6b00686353c66be061fb8b8b375a9c47bd284
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sat Aug 25 17:09:41 2018 +0000

    Development branch changes done
```

Changes will come back to staging (after add)

```
[root@ip-172-31-21-66 JavaProj]# git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   sshfile.txt
```

```
[root@ip-172-31-21-66 JavaProj]# cat sshfile.txt
First line added
```


Git reset mixed (Default)

Changes will come back to staging (before add)

```
[root@ip-172-31-21-66 JavaProj]# git reset a1f6b0068635
Unstaged changes after reset:
M      sshfile.txt
[root@ip-172-31-21-66 JavaProj]# git status
On branch master
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   sshfile.txt
```

Git reset hard

```
[root@ip-172-31-21-66 JavaProj]# git reset --hard a1f6b0068635
HEAD is now at a1f6b00 Development branch changes done
[root@ip-172-31-21-66 JavaProj]# git status
On branch master
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

</SOURCE FUSE>

Bring back deleted data:

git reflog --> will show all the commits history even after reset

```
[root@ip-172-31-21-66 JavaProj]# git reflog
a1f6b00 (HEAD -> master) HEAD@{0}: reset: moving to a1f6b0068635
5cdf850 HEAD@{1}: commit: first line added
a1f6b00 (HEAD -> master) HEAD@{2}: reset: moving to a1f6b0068635
12591f3 HEAD@{3}: commit: first line added
116032d (origin/master, origin/HEAD) HEAD@{4}: reset: moving to 116032dac211
8847cb7 HEAD@{5}: commit: first line
116032d (origin/master, origin/HEAD) HEAD@{6}: pull: Fast-forward
a1f6b00 (HEAD -> master) HEAD@{7}: reset: moving to a1f6b00686353c
116032d (origin/master, origin/HEAD) HEAD@{8}: clone: from git@github.com:raknas999/JavaProj.git
```

git checkout <commit id> --> checkout the deleted commit id

```
[root@ip-172-31-21-66 JavaProj]# git checkout 5cdf850
Note: checking out '5cdf850'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

    git checkout -b <new-branch-name>

HEAD is now at 5cdf850... first line added
```

```
[root@ip-172-31-21-66 JavaProj]# git log
commit 5cdf850f4b54e1177d091cd01d24cb5abcae9017 (HEAD)
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sat Aug 25 20:26:23 2018 +0000

    first line added
```



```
[root@ip-172-31-21-66 JavaProj]# git status  
HEAD detached at 5cdf850
```

```
[root@ip-172-31-21-66 JavaProj]# git checkout -b deletedbr  
Switched to a new branch 'deletedbr'  
[root@ip-172-31-21-66 JavaProj]# git status  
On branch deletedbr  
nothing to commit, working tree clean
```

```
[root@ip-172-31-21-66 JavaProj]# git branch  
* deletedbr  
  master  
You have new mail in /var/spool/mail/root  
[root@ip-172-31-21-66 JavaProj]# ls  
README.md  sample.txt  sshfile.txt  
[root@ip-172-31-21-66 JavaProj]# cat sshfile.txt  
First line added
```

Revert commit :

git revert <commit id>

It won't delete the commit id instead creates new commit on top of existing one.

```
[root@ip-172-31-21-66 JavaProj]# git revert 5cdf850f4b5  
[deletedbr 153ec5d] Revert "first line added"  
Committer: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
```

Cherry-pick

Choose a commit from one branch and apply it onto another

1. Make sure you are on the branch you want to apply the commit to.

```
git checkout <branch>
```

2. Execute the following:

```
git cherry-pick <commit-hash>
```

```
[root@ip-172-31-21-66 JavaProj]# vi sshfile.txt
[root@ip-172-31-21-66 JavaProj]# cat sshfile.txt
Line added in master
```

Take the commit id of the branch

```
[root@ip-172-31-21-66 JavaProj]# git log
commit 666e3daa41e24e788126dc60f6948dffdfeb8d64 (HEAD -> master)
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sat Aug 25 20:49:11 2018 +0000

    line added thru master
```

Switch the branch which you want to copy the commit and apply cherry-pick

```
[root@ip-172-31-21-66 JavaProj]# git cherry-pick 666e3daa41e2
[deletedbr d0f5ec9] line added thru master
Date: Sat Aug 25 20:49:11 2018 +0000
```

</SOURCE FUSE>

Commit will be copied into the Destination branch

```
[root@ip-172-31-21-66 JavaProj]# cat sshfile.txt  
Line added in master
```

```
[root@ip-172-31-21-66 JavaProj]# git branch  
* deletedbr  
master
```

Stashing :

Often, when you've been working on part of your project, things are in a messy state and you want to switch branches for a bit to work on something else and don't want to commit half done work.

Git stash -□ to save the changes in buffer

Git stash list -□ to check the stashed list

Git stash apply -□ To jump into latest one

Git stash apply stashid □ to go to the particular stash block

Git stash apply pop □ to go to the particular block and remove the entry from stash list

</SOURCE FUSE>

Stash process:

- create branch(child1) on master do some changes to file, then add the file then stash
- create branch(child2) on master do some changes to file, then add the file then stash
- create branch(child3) on master do some changes to file, then add the file then stash
- git stash list (it will show you from 0 to 2)
- git stash apply stash@{1} --> it will go to second one
- Now undo the changes or commit (git checkout file or git commit)
- git stash apply stash@{2} --> it will go to third one
- now undo the changes or commit (git checkout file or git commit)

GIT Rebase :

Rebasing is the process of moving or combining a sequence of commits to a new base commit.

Example :

- Commit A: we add a.txt file in the 'master' branch
- Commit B: we add b.txt file in the 'master' branch
- At this stage, we create the branch 'feature' which means it will have a.txt and b.txt
- Commit C: we add c.txt file in the 'master' branch
- We go to the 'feature' branch
- Commit E: we modify a.txt in 'feature' branch
- Commit F: we modify b.txt in 'feature' branch

</SOURCE**FUSE**>

```
git init
touch a.txt
git add -A
git commit -m "Commit A: added a.txt"
touch b.txt
git add -A
git commit -m "Commit B: added b.txt"
git branch feature
touch c.txt
git add -A
git commit -m "Commit C: added c.txt"
git status
git checkout feature
echo aaa > a.txt
git add -A
git commit -m "Commit E: modified a.txt"
echo bbb > b.txt
git add -A
git commit -m "Commit F: modified b.txt"
```

</SOURCE FUSE>

git checkout master

```
[root@ip-172-31-21-66 rebaseexample]# git log --oneline
2e18749 (HEAD -> master) Commit C: added c.txt
d534761 Commit B: added b.txt
d525e14 Commit A: added a.txt
fffb91 (origin/master, origin/HEAD) Initial commit
```

git checkout feature

```
[root@ip-172-31-21-66 rebaseexample]# git log --oneline
cc097be (HEAD -> feature) Commit F: modified b.txt
376a209 Commit E: modified a.txt
d534761 Commit B: added b.txt
d525e14 Commit A: added a.txt
fffb91 (origin/master, origin/HEAD) Initial commit
```

Merge the feature branch to master

```
[root@ip-172-31-21-66 rebaseexample]# git branch
feature
* master
[root@ip-172-31-21-66 rebaseexample]# git merge feature
Merge made by the 'recursive' strategy.
a.txt | 1 +
b.txt | 1 +
2 files changed, 2 insertions(+)
```


Check the commit history after merge

```
[root@ip-172-31-21-66 rebaseexample]# git log --oneline
2d79d7b (HEAD -> master) Merge branch 'feature'
cc097be (feature) Commit F: modified b.txt
376a209 Commit E: modified a.txt
2e18749 Commit C: added c.txt
d534761 Commit B: added b.txt
d525e14 Commit A: added a.txt
fffbb91 (origin/master, origin/HEAD) Initial commit
```

After merge log of feature branch

```
[root@ip-172-31-21-66 rebaseexample]# git log --oneline
cc097be (HEAD -> feature) Commit F: modified b.txt
376a209 Commit E: modified a.txt
d534761 Commit B: added b.txt
d525e14 Commit A: added a.txt
fffbb91 (origin/master, origin/HEAD) Initial commit
```

In the 'master' branch, you will notice there is a new commit G that has merged the changes from 'feature' branch. Basically, the following action has taken place:

```
A - B - C - G (master)
      \   /
      E - F (feature)
```

</SOURCE FUSE>

In the Commit G(new), all the changes from 'feature' branch have been brought into the master branch. But the 'feature' branch itself has remained untouched due to the merge process. Notice the hash of each commit. After the merge, E (376a209) and F (cc097be) commit has the same hash on the 'feature' and 'master' branch.

Merging with Rebasing

After done the changes on both master and feature branches

git checkout master

```
[root@ip-172-31-21-66 rebaseexample]# git log --oneline
e5d583e (HEAD -> master) Commit C: added c.txt
6848f65 Commit B: added b.txt
7baa1d6 Commit A: added a.txt
ffffbb91 (origin/master, origin/HEAD) Initial commit
[root@ip-172-31-21-66 rebaseexample]#
```

git checkout feature

```
[root@ip-172-31-21-66 rebaseexample]# git log --oneline
0db8061 (HEAD -> feature) Commit F: modified b.txt
caae8ae Commit E: modified a.txt
6848f65 Commit B: added b.txt
7baa1d6 Commit A: added a.txt
ffffbb91 (origin/master, origin/HEAD) Initial commit
[root@ip-172-31-21-66 rebaseexample]#
```

git rebase master

```
[root@ip-172-31-21-66 rebaseexample]# git rebase master
First, rewinding head to replay your work on top of it...
Applying: Commit E: modified a.txt
Applying: Commit F: modified b.txt
```

Merge 'feature' into 'master'

Git checkout master

git merge feature

Verify the log of master branch

```
[root@ip-172-31-21-66 rebaseexample]# git log --oneline
0852f8a (HEAD -> master, feature) Commit F: modified b.txt
8da5a57 Commit E: modified a.txt
e5d583e Commit C: added c.txt
6848f65 Commit B: added b.txt
7baa1d6 Commit A: added a.txt
fffb91 (origin/master, origin/HEAD) Initial commit
```

Verify the log of feature branch

```
[root@ip-172-31-21-66 rebaseexample]# git log --oneline
0852f8a (HEAD -> feature, master) Commit F: modified b.txt
8da5a57 Commit E: modified a.txt
e5d583e Commit C: added c.txt
6848f65 Commit B: added b.txt
7baa1d6 Commit A: added a.txt
fffb91 (origin/master, origin/HEAD) Initial commit
```

</SOURCE FUSE>

Notice that after the rebase and merge both branches are the same. Also, the hashes for E and F have changed in both branches. Basically, in the rebase scenario, this is what happened:

```
A - B - C
      |
      E' - F' (feature, master)
```

That's why there is no new commit. The E and F commits have been recalculated and latched to the end of the 'master' branch.

Re arranging commit using rebase

```
[root@ip-172-31-21-66 rebaseexample]# git log
commit 48e0f21d1bf2b8f30f5d060a2bf644b3208883cf (HEAD -> feature)
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sun Aug 26 11:33:23 2018 +0000

    Commit F: modified b.txt

commit felb48ead62b28e11d50b097abce893bb2429c2f
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sun Aug 26 11:32:58 2018 +0000

    Commit C: added c.txt

commit 36fd5c2dd51b39e545be6a46bfb9788004412e30
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date: Sun Aug 26 11:33:23 2018 +0000

    Commit E: modified a.txt
```


git rebase -i HEAD~3 : Will open latest three commits

```
pick 36fd5c2 Commit E: modified a.txt
pick felb48e Commit C: added c.txt
pick 48e0f21 Commit F: modified b.txt

# Rebase 6848f65..48e0f21 onto 6848f65 (3 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Now change the order and save the file

```
pick fe1b48e Commit C: added c.txt
pick 36fd5c2 Commit E: modified a.txt
pick 48e0f21 Commit F: modified b.txt

# Rebase 6848f65..48e0f21 onto 6848f65 (3 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```


</SOURCE FUSE>

Check the log , now we can see the changed order

```
[root@ip-172-31-21-66 rebaseexample]# git log
commit 7916dad75a1da8411f935a0f64e08ca265200d9a (HEAD -> feature)
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date:    Sun Aug 26 11:33:23 2018 +0000

    Commit F: modified b.txt

commit 231ece4ac5ece16179fb6496257b3e2b4a569078
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date:    Sun Aug 26 11:33:23 2018 +0000

    Commit E: modified a.txt

commit 3536ddc1451cf744e4e5fd28424dd0d44e16b291
Author: root <root@ip-172-31-21-66.ap-south-1.compute.internal>
Date:    Sun Aug 26 11:32:58 2018 +0000

    Commit C: added c.txt
```

GIT Conflict

No sync between remote repository and local repository conflicts will happen

GIT

```
[root@ip-172-31-21-66 rebaseexample]# cat sample.txt
First line added thru GIT
Second line added thru GIT
[root@ip-172-31-21-66 rebaseexample]#
```

GIT HUB



Try update the remote with local , will face conflict due to non sync

```
[root@ip-172-31-21-66 rebaseexample]# git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:raknas999/rebaseexample
   294c1df..5d84e2e  master    -> origin/master
Auto-merging sample.txt
CONFLICT (content): Merge conflict in sample.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Resolving Conflicts : Open the file and Modify accordingly

```
[root@ip-172-31-21-66 rebaseexample]# cat sample.txt
First line added thru GIT
<<<<<< HEAD
Second line added thru GIT
=====
Thrid line added thru GIT HUB
>>>>>> 5d84e2ed1b14cf4ae7742b42f4d1c382825f520c
```

</SOURCE FUSE>

```
[root@ip-172-31-21-66 rebaseexample]# cat sample.txt
First line added thru GIT
Second line added thru GIT
Thrid line added thru GIT HUB
```

```
[root@ip-172-31-21-66 rebaseexample]# git pull
Already up-to-date.
```

GIT FORK

Github makes a separate copy for you, independent of the original repository

The screenshot shows the GitHub interface for a repository named 'gitdemo' by user 'sundarrajboobalan'. The top navigation bar includes the GitHub logo, a search bar, and links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The repository header shows the name 'sundarrajboobalan / gitdemo' and interaction buttons: 'Unwatch' (1), 'Star' (0), and 'Fork' (0). Below the header is a tabbed interface with 'Code' selected. The main content area shows the 'master' branch with 1 branch and 0 tags. It lists two commits: 'sundarrajboobalan update aboutus' (8f8e5a9, 6 hours ago, 4 commits) and 'update aboutus' (6 hours ago). Below the commits is a table of files: 'aboutus.html' (update aboutus, 6 hours ago) and 'index.html' (Index.html added, 6 hours ago). At the bottom, there is a prompt to 'Add a README' to help people understand the project. On the right side, there is an 'About' section with the text 'No description, website, or topics provided.' and a 'Releases' section with the text 'No releases published. Create a new release'.

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

sundarrajboobalan / gitdemo Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags Go to file Add file Code

sundarrajboobalan update aboutus 8f8e5a9 6 hours ago 4 commits

aboutus.html	update aboutus	6 hours ago
index.html	Index.html added	6 hours ago

Help people interested in this repository understand your project by adding a README. Add a README

About No description, website, or topics provided.

Releases No releases published. Create a new release

Open the project which you want to be fork and click on fork



</SOURCE FUSE>

GIT Merge Conflict Resolution

Launch Merge Tool : `git mergetool`

```
root@ip-172-31-38-249:~/devopsrepo

Hello this is srini
added second line
added thrid line
added fifth line

Hello this is srini
added second line
added thrid line

Hello this is srini
added second line
added thrid line
added fourth line

./hello LOCAL 3528.txt      1,1      All ./hello BASE 3528.txt      1,1      All ./hello REMOTE 3528.txt      1,1      All
Hello this is srini
added second line
added thrid line
<<<<<< HEAD
added fifth line
=====
added fourth line
>>>>>> test
```

Changing Merge Tool : `git config --global merge.tool <tool name>`

You can incorporate the changes by manually editing the MERGED split, or use Vim shortcuts pull from one of the LOCAL, BASE ad REMOTE versions.

```
:diffg RE # get from REMOTE  
:diffg BA # get from BASE  
:diffg LO # get from LOCAL
```

save the changes then quit with :wqa to close all the splits. Remember to commit the merge.

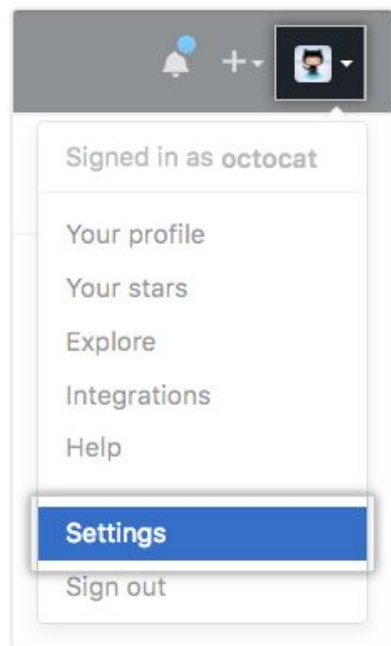
```
$ git commit -am 'merged from several branches'
```

Creating organizations

You can choose to set up a new organization or convert an existing personal account into an organization.

An organization is a collection of user accounts that owns repositories. Organizations have one or more owners, who have administrative privileges for the organization. Organizations can also be used for namespacing—for example, `http(s)://[hostname]/[organization name]/` takes you to an organization's profile on GitHub Enterprise Server, while `http(s)://[hostname]/[organization name]/[repository name]/` takes you to a repository's profile.

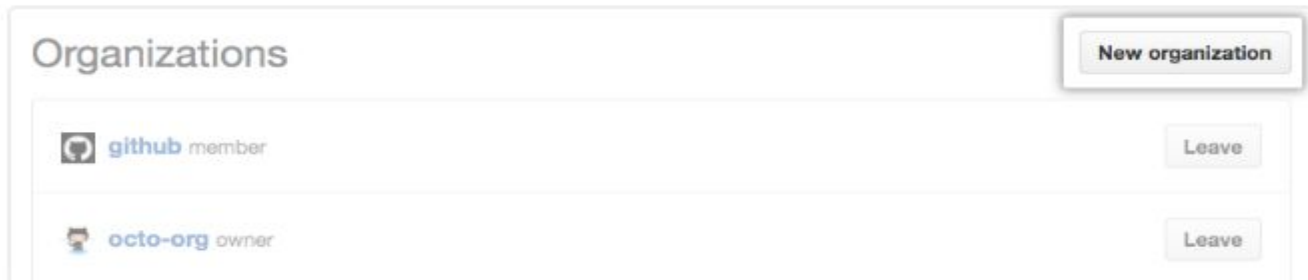
- 1 In the upper-right corner of any page, click your profile photo, then click **Settings**.



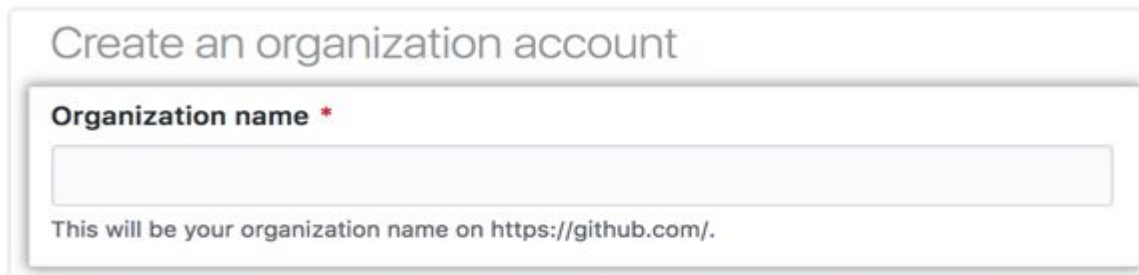
- 2 In your user settings sidebar, click **Organizations**.



- 3 In the "Organizations" section, click **New organization**.



- 4 Under "Organization name", give the organization a name.

A screenshot of a form titled 'Create an organization account'. Inside the form, there is a label 'Organization name *' followed by a text input field. The input field is highlighted with a black border. Below the input field, there is a note: 'This will be your organization name on https://github.com/'.

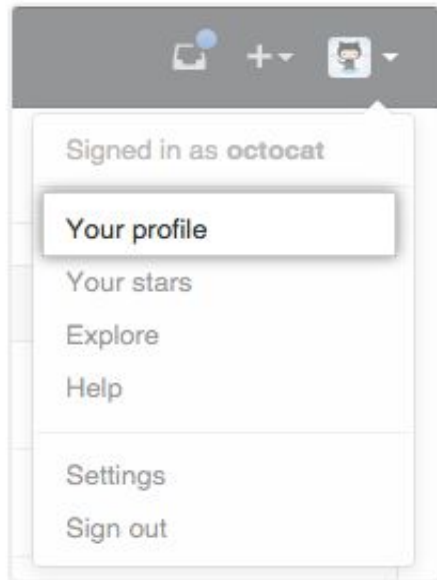
- 5 Under "Contact email," type the email address of a person who can be contacted for more information about the organization.
- 6 Click **Create organization**.

Creating teams


Teams give organizations the ability to create groups of members and control access to repositories. Team members can be granted read, write, or admin permissions to specific repositories.

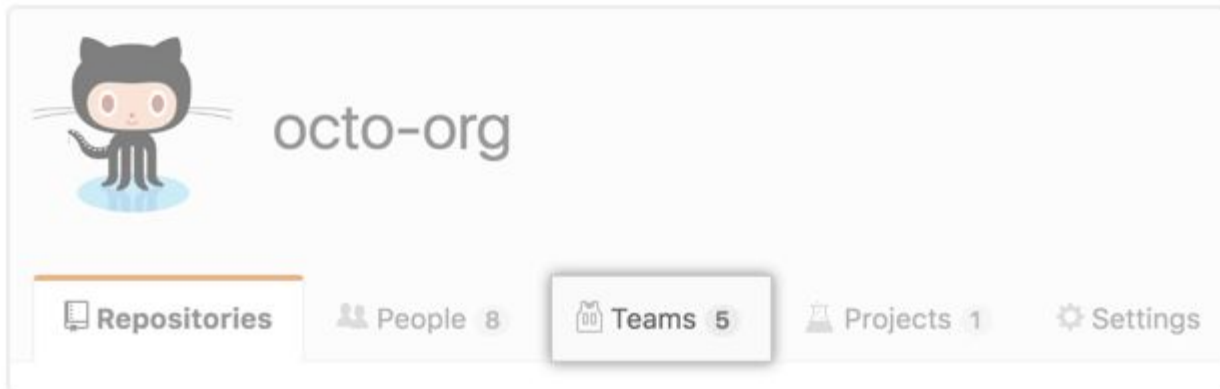
Teams are central to many of GitHub's collaborative features, such as team @mentions to notify appropriate parties that you'd like to request their input or attention. For more information, see "[Permission levels for an organization repository](#)".

- 1 In the top right corner of GitHub Enterprise Server, click your profile photo, then click **Your profile**.

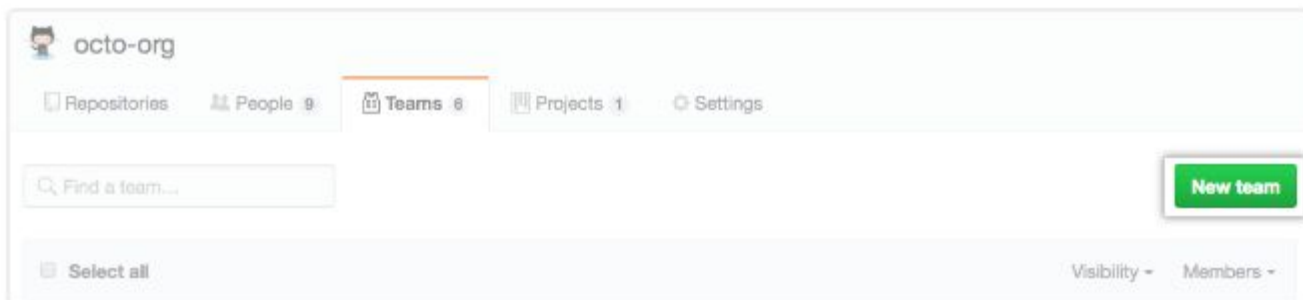


- 2 On the left side of your profile page, under "Organizations", click the icon for your organization.

- 3 Under your organization name, click  **Teams**.



- 4 On the right side of the Teams tab, click **New team**.



</SOURCEFUSE>

- 5 Under "Create new team", type the name for your new team.

Create new team

Team name

Authors

✓

Mention this team in conversations as **@octo-org/authors**.

Description


Writers for the octo-project

- 6 Optionally, in the "Description" field, type a description of the team.

- 6 Optionally, in the "Description" field, type a description of the team.

Create new team

Team name

Authors 

Mention this team in conversations as **@octo-org/authors**.

Description

Writers for the octo-project

</SOURCE FUSE>

- 7 Decide whether the team will be visible or secret.

Create new team

Team name

Authors ✓

Mention this team in conversations as @octo-org/authors.

Description

Writers for the octo-project

Parent team

Select parent team ▾

Team visibility

☒ **Visible** Recommended

A visible team can be seen and @mentioned by every member of this organization.

☐ **Secret**

A secret team can only be seen by its members.

Create team

- 8 Optionally, if you're creating a [child team](#), use the drop-down menu to choose a parent team for your new team.
- 9 Click **Create team**.

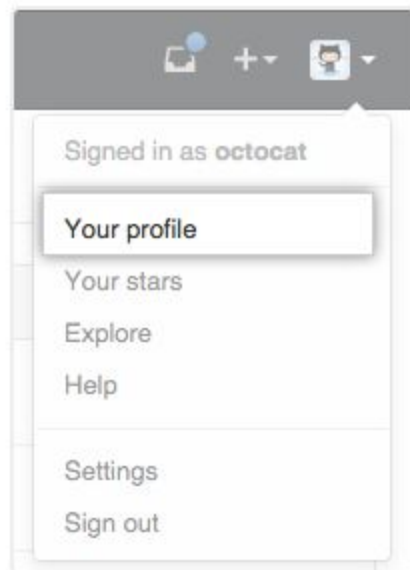
Adding people to teams


Once a team has been created, organization admins can add users from your GitHub Enterprise Server instance to the team and determine which repositories they have access to.

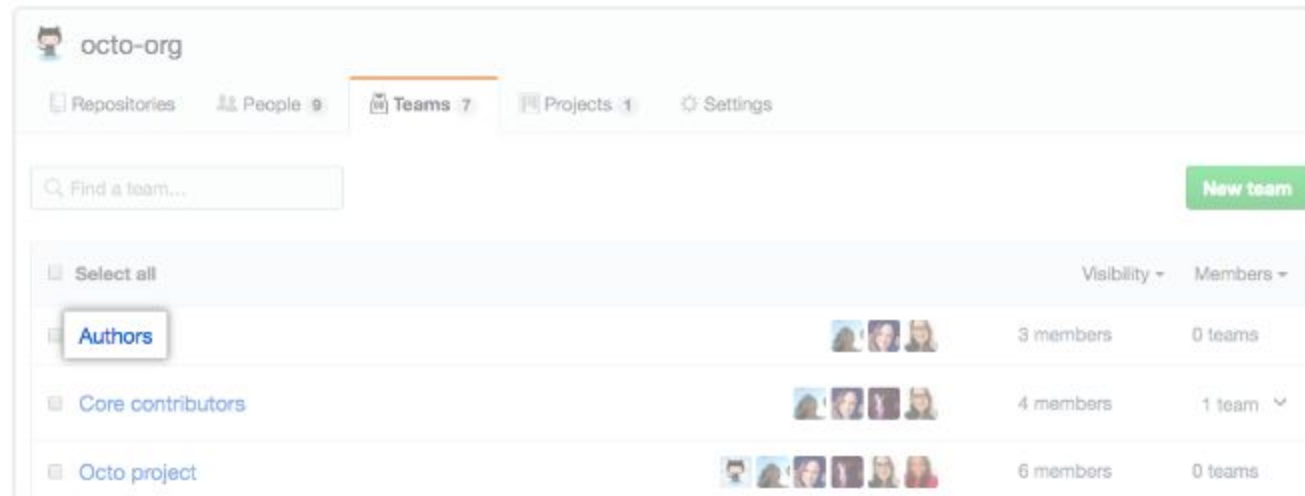
Each team has its own individually defined [access permissions for repositories owned by your organization](#).

Setting up a team

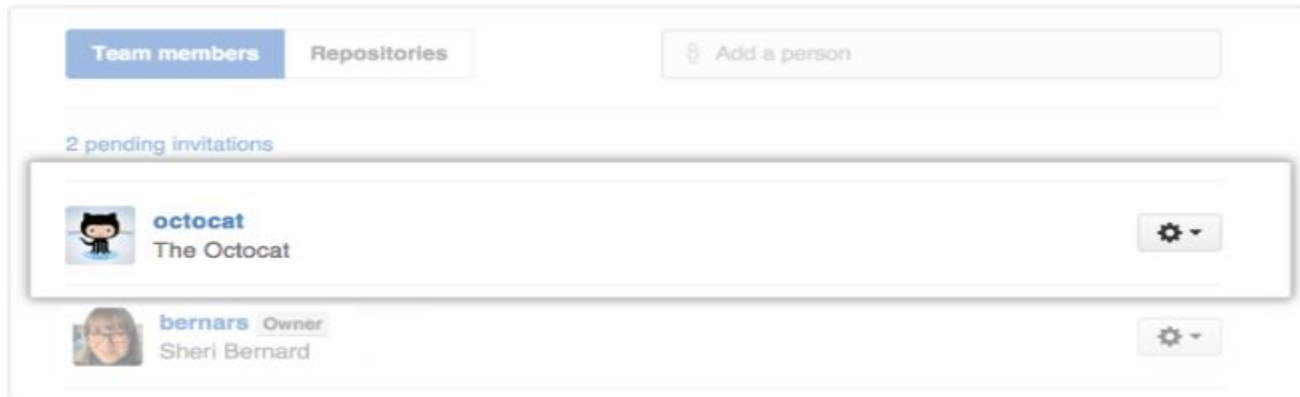
- 1 In the top right corner of GitHub Enterprise Server, click your profile photo, then click **Your profile**.



- 2 On the left side of your profile page, under "Organizations", click the icon for your organization.
- 3 Under your organization name, click  Teams.
- 4 On the Teams tab, click the name of the team.




- 5 Type the username of the person you want to add and click **Enter**. The user will immediately be added to the team.



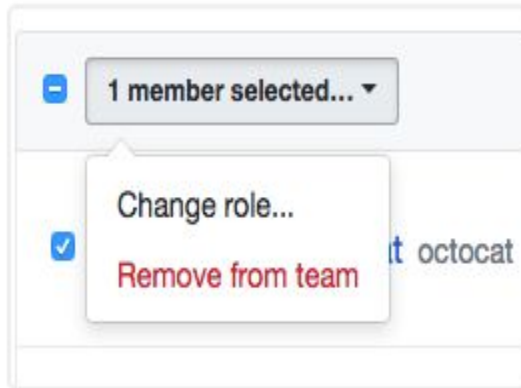
- 6 Review the list of repositories that the new team member will have access to, then click **Add USERNAME to TEAMNAME**.

Removing users from teams and organizations

If a member of your organization no longer requires access to certain repositories, you can remove them from the team that allows that access. If a member of your organization no longer requires access to any repositories owned by the organization, you can remove them from the organization.

- 1 In the top right corner of GitHub Enterprise Server, click your profile photo, then click **Your profile**.
- 2 On the left side of your profile page, under "Organizations", click the icon for your organization.
- 3 Under your organization name, click  **Teams**.
- 4 On the Teams tab, click the name of the team.
- 5 Select the person or people you'd like to remove.

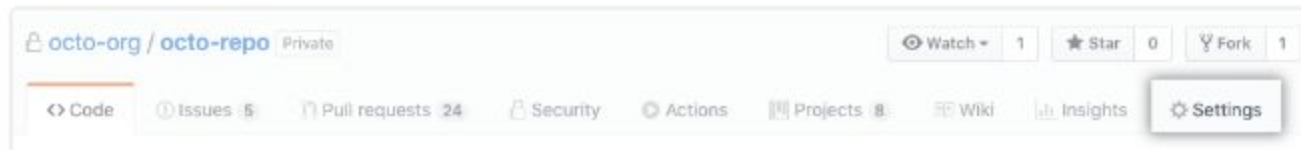
- 6 Above the list of team members, use the drop-down menu and click **Remove from team**.



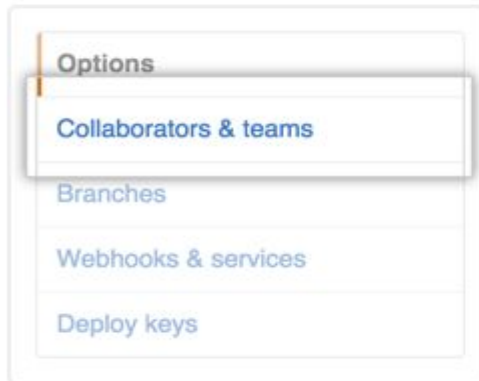
Adding outside collaborators to repositories in your organization

An *outside collaborator* is a person who isn't explicitly a member of your organization, but who has Read, Write, or Admin permissions to one or more repositories in your organization.

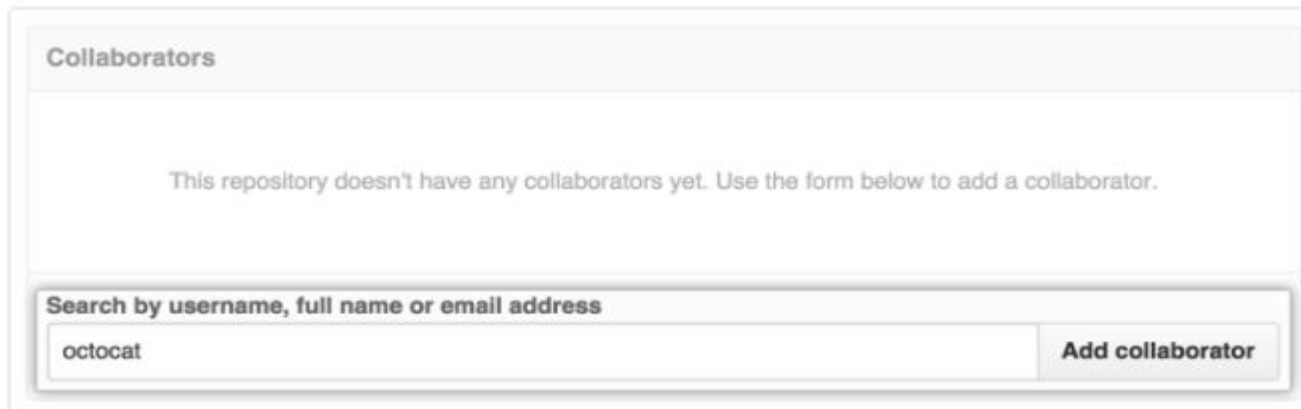
- 1 In the top right corner of GitHub, click your profile photo, then click **Your profile**.
- 2 On the left side of your profile page, under "Organizations", click the icon for your organization.
- 3 In the organization's **Repositories** tab, click the name of a repository, or search for the repository by name.
- 4 Under your repository name, click ⚙ **Settings**.



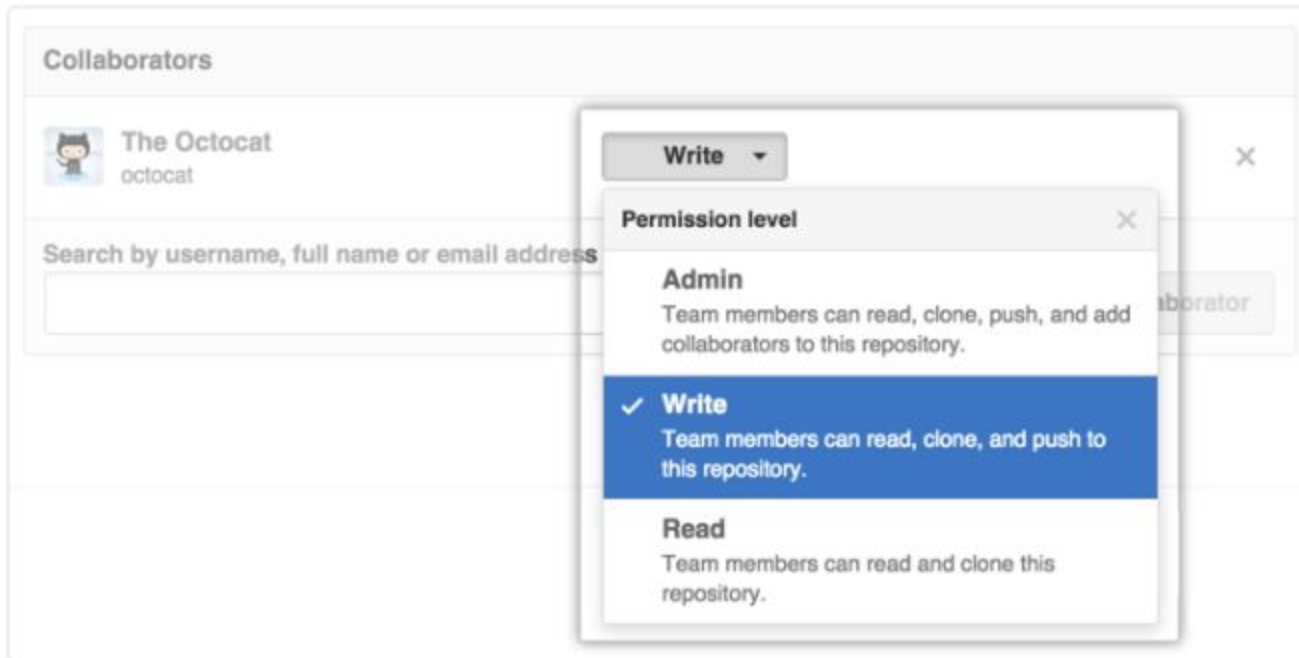
- 5 In the left sidebar, click **Collaborators & teams**.



- 6 Under "Collaborators", type the name of the person you'd like to give access to the repository, then click **Add collaborator**.

A screenshot of the GitHub 'Collaborators' page. The page has a header 'Collaborators' and a message: 'This repository doesn't have any collaborators yet. Use the form below to add a collaborator.' Below the message is a search bar with the placeholder text 'Search by username, full name or email address'. The search bar contains the text 'octocat'. To the right of the search bar is a button labeled 'Add collaborator'.

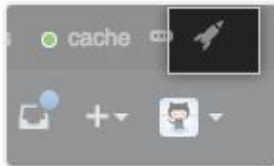
- 7 Next to the new collaborator's name, choose the appropriate permission level: *Write*, *Read*, or *Admin*.



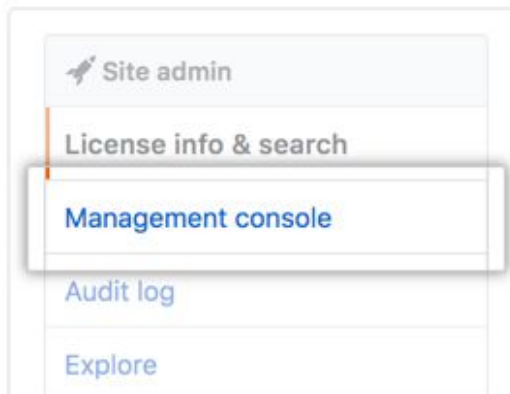
Upgrading GitHub Enterprise Server

Upgrade GitHub Enterprise Server to get the latest features and security updates.

- 1 Enable automatic updates. For more information, see "[Enabling automatic updates.](#)"
- 2 In the upper-right corner of any page, click 🚀.



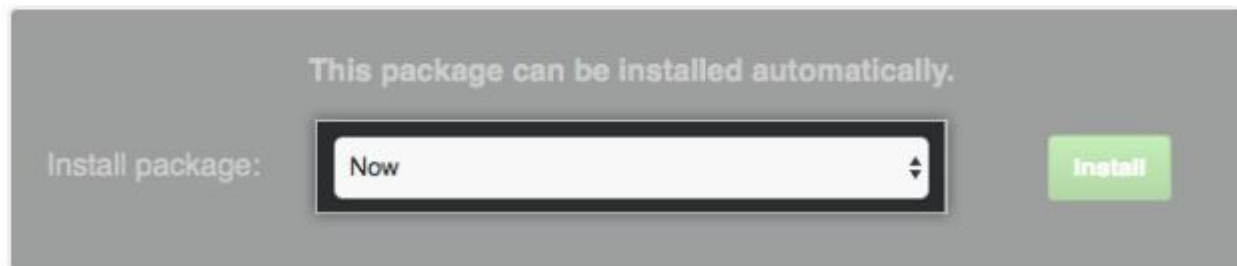
- 3 In the left sidebar, click **Management Console**.



- 4 At the top of the Management Console, click **Updates**.



- 5 When a new hotpatch has been downloaded, use the Install package drop-down menu:
 - o To install immediately, select **Now**:
 - o To install later, select a later date.



- 6 Click **Install**.

Installing a hotpatch using the administrative shell

Note: If you've enabled automatic update checks, you don't need to download the upgrade package and can use the file that was automatically downloaded. For more information, see "[Enabling automatic update checks](#)."

- 1 SSH into your GitHub Enterprise Server instance.

```
$ ssh -p 122 admin@HOSTNAME
```

- 2 Browse to the [GitHub Enterprise Server Releases](#) page. Next to the release you are upgrading to, click **Download**, then click the **Upgrading** tab. Copy the URL for the upgrade hotpackage (.hpkg file).

- 3 Download the upgrade package to your GitHub Enterprise Server instance using `curl` :

```
admin@HOSTNAME:~$ curl -L -O UPGRADE-PKG-URL
```

- 4 Run the `ghe-upgrade` command using the package file name:

```
admin@HOSTNAME:~$ ghe-upgrade GITHUB-UPGRADE.hpkg  
*** verifying upgrade package signature...
```

</SOURCE FUSE>

<https://enterprise.github.com/releases>



https://enterprise.github.com/releases



GitHub Enterprise

Features

Case Studies

Pricing

Resources

Contact

Sign in

Try it for free

Releases



2.17.0 May 23, 2019

Notes · Download

2.16.9 May 21, 2019

Notes · Download

2.15.14 May 21, 2019

Notes · Download

2.14.21 May 21, 2019

Notes · Download

Download GitHub Enterprise

Version 2.16.9

Installing

Upgrading

Metadata

⚠ This is not the [latest release](#) of GitHub Enterprise.



Universal hotpatch upgrade package

Choose this option if you are already running GitHub Enterprise 2.16.0 (or newer). This patch can be applied to any supported hypervisor and cloud platform.

<https://github-enterprise.s3.amazonaws.com/hotpatch/2.16/github-enterprise-2.16.9.hpkg>