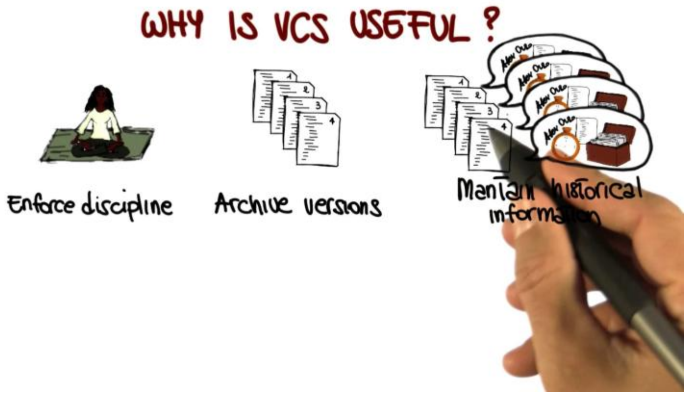
Version Control System

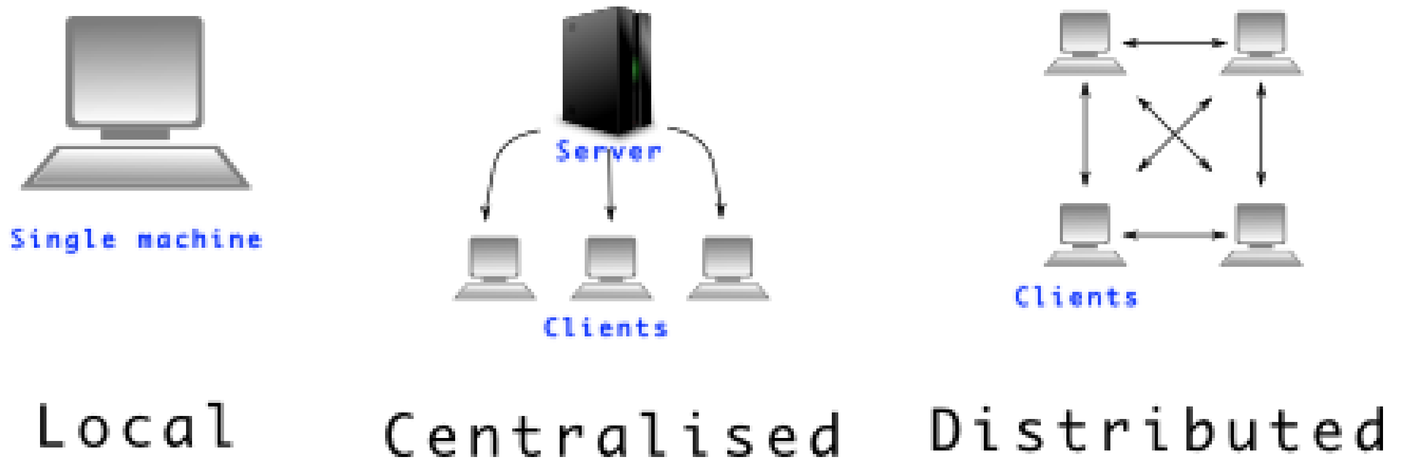
Why it is useful ?

- Collaboration
- Storing versions (Properly)
- Restoring Previous versions
- Understanding what happened
- Backup



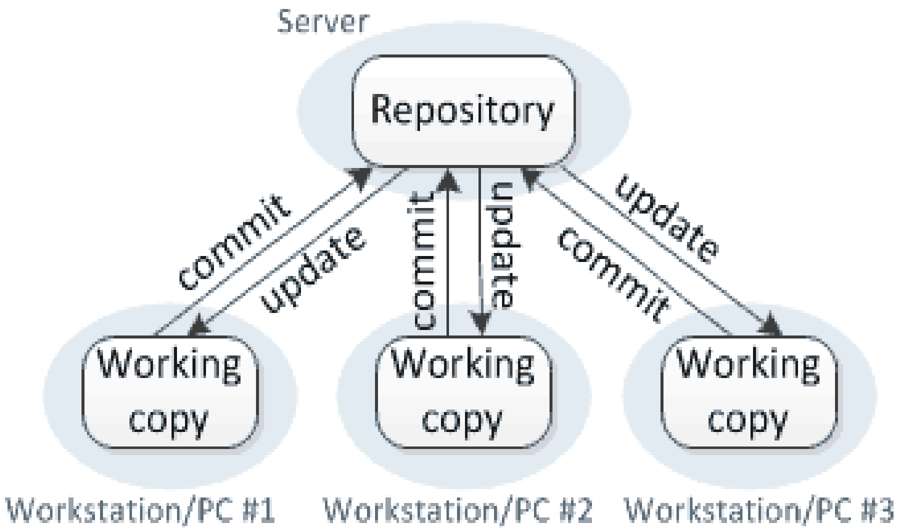
Types of Version Control System

- Centralized Version Control System
- Distributed / Decentralized Version Control System



Centralized Version Control System

Use central server to store all files and enables team collaboration

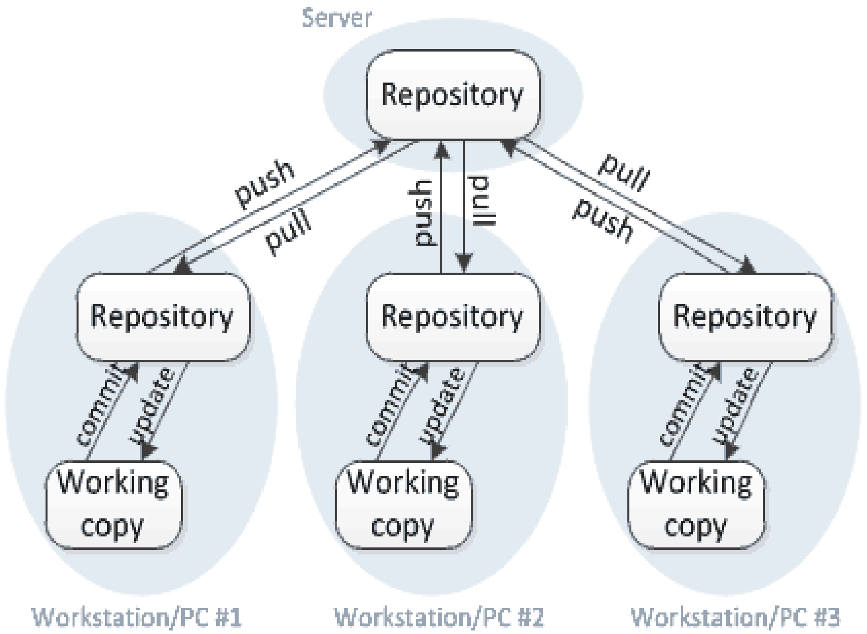


Issues in Centralized Version Control System

- If central server goes down, during that time no one can collaborate at all.
- If the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project

Decentralized Version Control System

- Clients not only check out the latest snapshot of the directory but they also fully mirror the repository.
- If server goes down , then the repository from any client can be copied back to the server to restore it.
- Every checkout is the full backup of the repository.



Examples for Decentralized Version Control System

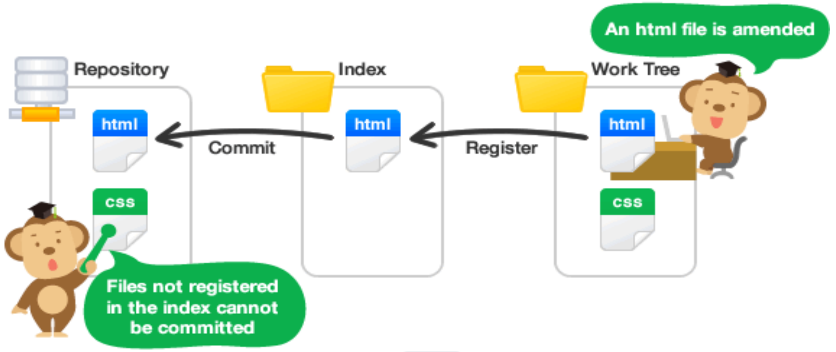
- GitHub
- Bitbucket
- GitLab

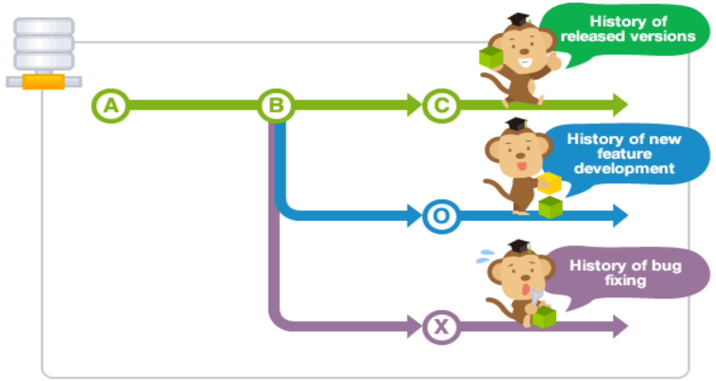
Git

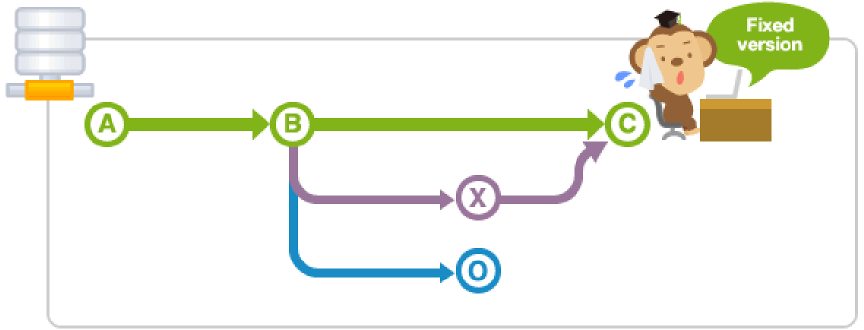
- In Git you can commit changes, create branches , view logs, etc. when you are offline.
- You require network connection only to publish your changes and take the latest changes.

Advantages of Git

- Fast and small
- Implicit backup
- Security
- No need of powerful hardware
- Easier branching




History of released versions
A
B
C
History of new feature development
O
History of bug fixing
X



Installation

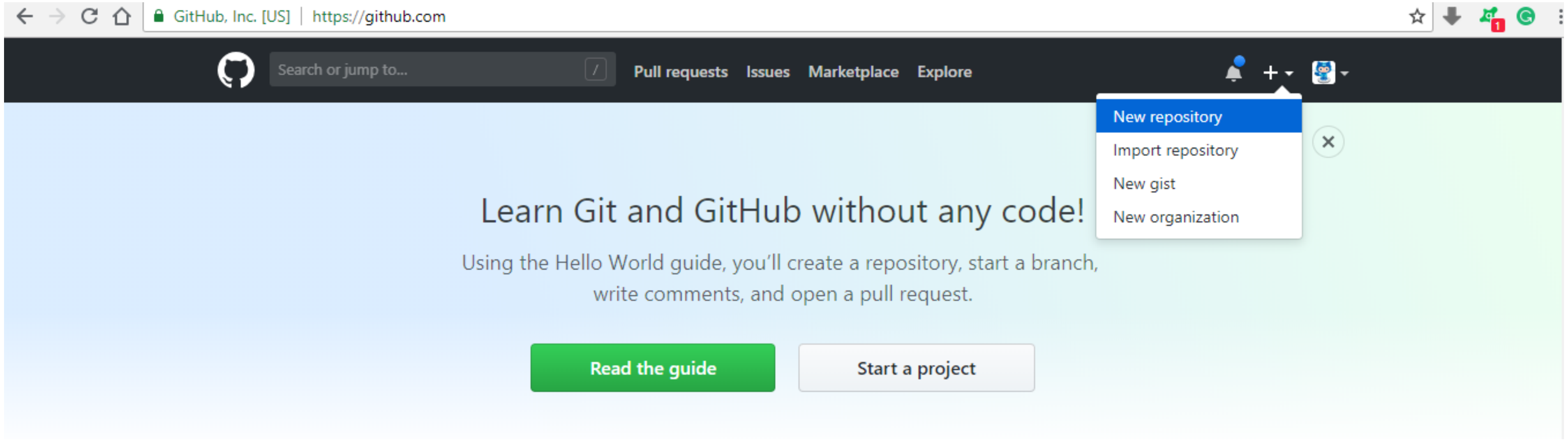
Linux: sudo apt install git-all

Windows: [Download]

Getting Start to work with git

Step 1

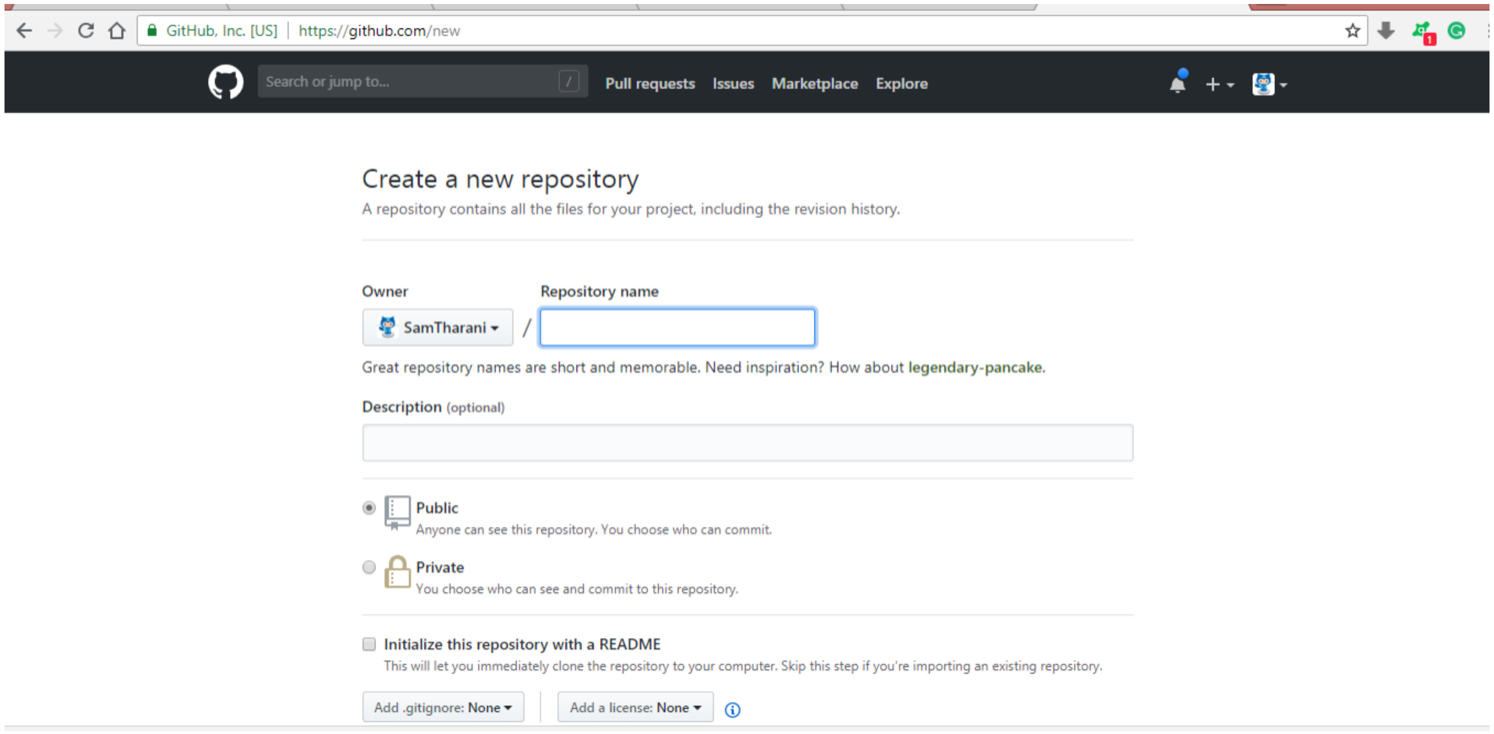
Create new repository in gitHub.



Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Fill necessary details.


GitHub, Inc. [US] | https://github.com/new

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: SamTharani / Repository name:

Great repository names are short and memorable. Need inspiration? How about legendary-pancake.

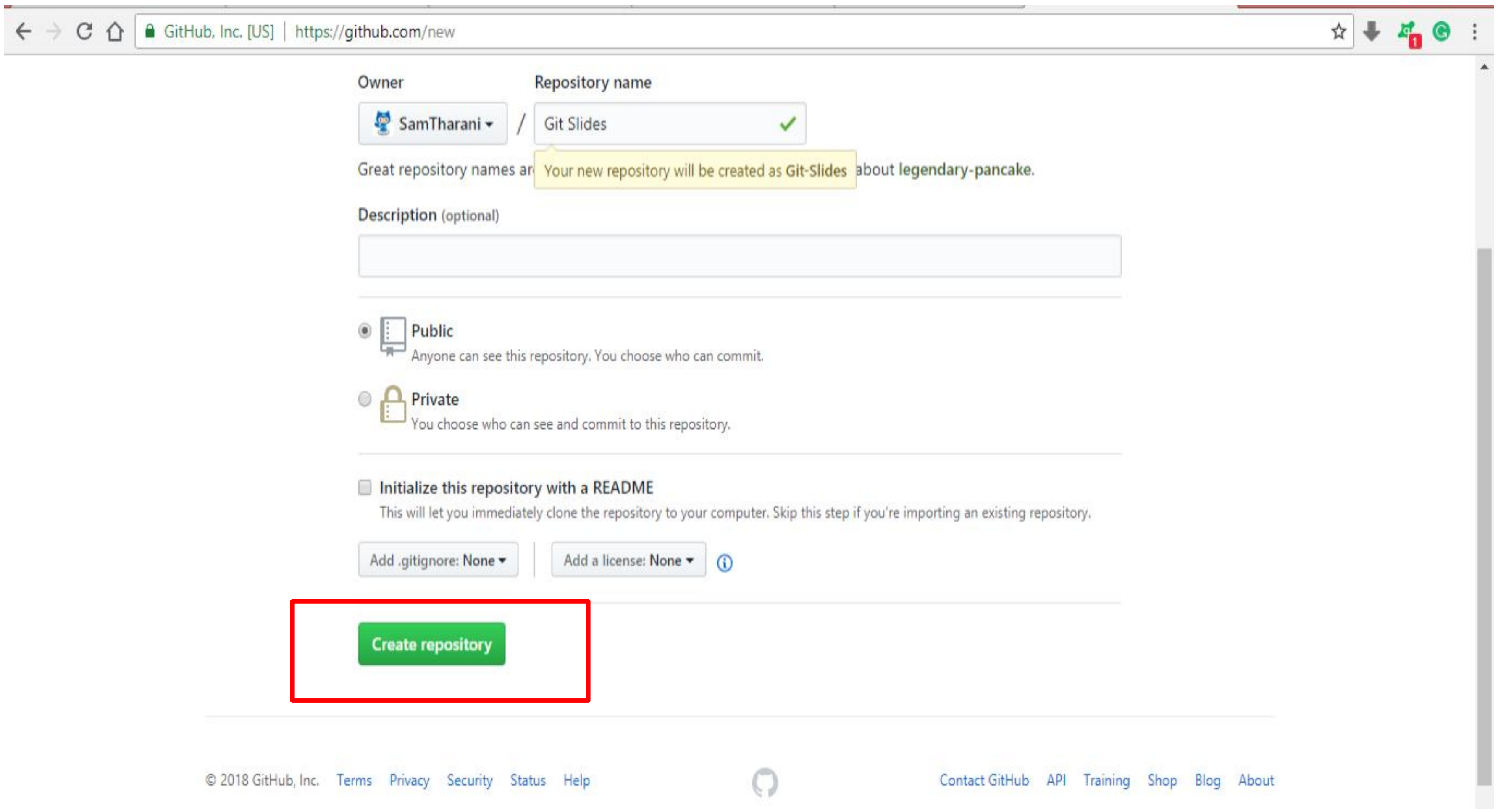
Description (optional)

[✓ Public] Anyone can see this repository. You choose who can commit.

Private: You choose who can see and commit to this repository.

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

Add .gitignore: None | Add a license: None



GitHub, Inc. [US] | https://github.com/new

Owner
SamTharani ▾
/
Repository name
Git Slides ✓

Great repository names ar Your new repository will be created as **Git-Slides** about **legendary-pancake**.

Description (optional)

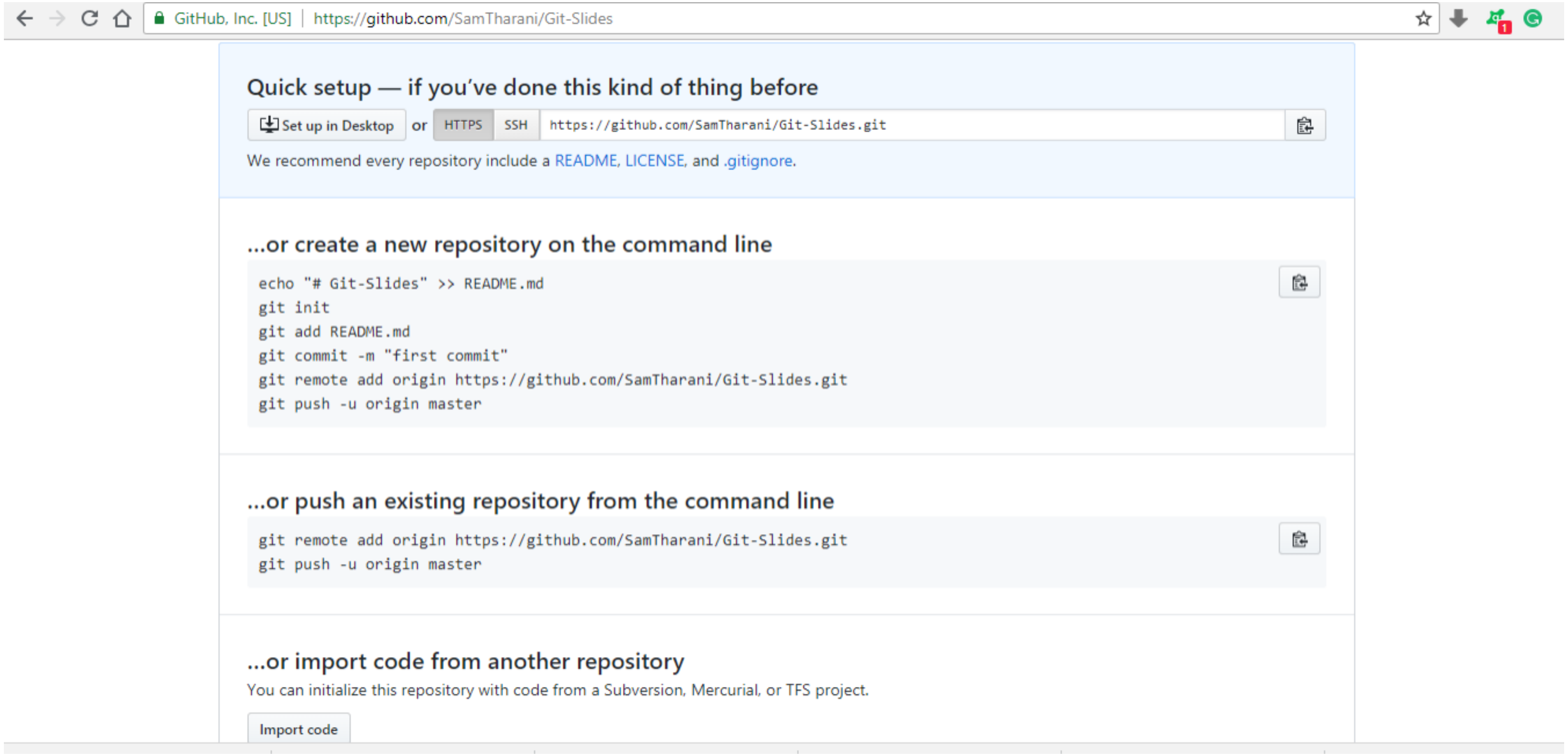
● **Public**
Anyone can see this repository. You choose who can commit.

○ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾ | Add a license: **None** ▾

© 2018 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About



GitHub, Inc. [US] | https://github.com/SamTharani/Git-Slides

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

Set up in Desktop or HTTPS SSH https://github.com/SamTharani/Git-Slides.git

We recommend every repository include a README, LICENSE, and .gitignore.

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

```
echo "# Git-Slides" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/SamTharani/Git-Slides.git
git push -u origin master
```

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

```
git remote add origin https://github.com/SamTharani/Git-Slides.git
git push -u origin master
```

...or import code from another repository

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

Import code

Staging environment in git

Step 3

- Create a file.

```
  $ touch <fileName>
```

- Create a new folder to the repository

```
 $ mkdir <folderName>
```

- Use the **git status** command to see which files”git knows exist.

```
$ git status
```

Step 4

- Add a file to a commit, you first need to add it to the staging environment.
  `$ add . or git add <newFolderName> then git add <filename>`

- Add a file in a folder
  `$ git add <newFolderName>\fileName`

Step 5

- Package them into a commit using the **git commit**
  `$ git commit -m “Your message about the commit”`

Step 6

- Configure the usename and password

```
$ git config --global user.name “<username>”

$ git config --global user.password “<password>”
```

Step 7

- Upload the file into Github repository **git push**

```
$ git push –u origin master
```

Create branch in git

Step 5

- To create new branch .

```
$ git checkout -b <my branch name>
```

- To check available branches **git branch**

```
$ git branch
```

Git Conflict

- Conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it.
- In these cases, Git cannot automatically determine what is correct.
- Conflicts only affect the developer conducting the **merge**, the rest of the team is unaware of the conflict.
- Git will mark the file as being conflicted and halt the merging process.
- It is then the developers' responsibility to resolve the conflict.

Git Conflict

Identify conflict

```
$ git status
```

```
On branch master
You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)
Unmerged paths:
(use "git add <file>..." to mark resolution)
both modified:   merge.txt
```

Git Conflict

Identify conflict

```
$ vi merge.txt
```

```
<<<<<<< HEAD
this is some content to mess with
content to append
=======
totally different content to merge later
>>>>>>> new_branch_to_merge_later
```

Git Conflict

Identify conflict

- These new lines as "conflict dividers".
- The ======= line is the "center" of the conflict.
- All the content between the center and the <<<<<<< HEAD line is content that exists in the current branch master which the HEAD ref is pointing to.
- Alternatively all content between the center and >>>>>>> new_branch_to_merge_later is content that is present in our merging branch.

Git stash

- Stashing your work
- Re-applying your stashed changes
- Stashing untracked or ignored files
- Managing multiple stashes
- Viewing stash diffs
- Partial stashes
- Creating a branch from your stash
- Clean up your stash

Git stash

- **Stashing your work**

  ```
  $ git stash
  ```

  takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy

Note:

`stash` is local to your Git repository; stashes are not transferred to the server when you push.

Git stash

- **Re-applying your stashed changes**

  ```
  $ git stash pop
  ```

  Popping your stash removes the changes from your stash and re-applies them to your working copy.

  ```
  .$ git stash apply
  ```

  This is useful if you want to apply the same stashed changes to multiple branches

Git stash

- **Stashing untracked or ignored files**

  ```
  $ git stash -u
  ```

  tells git stash to also stash your untracked files.

  ```
  .$ git stash -a
  ```

  tells git stash to also stash your ignored files.

Git stash

- **Managing multiple stashes**

```
$ git stash list
```

View multiple stashes on top of the branch.

```
$ git stash save "message"
```

Annotate the stash with description.

```
$ git stash pop stash_identifier
```

re-apply the given stash into the current working copy

Git stash

- **Viewing stash diffs**

  ```
  $ git stash show
  ```

  View a summary of a stash.

  ```
  $ git stash show -p
  ```

  View the full difference of the stash.

Git stash

- **Creating a branch from your stash**

```
$ git stash branch
```

Git stash

- **Clean up your stash**

```
$ git stash clear
```

Delete all of your stashes.