Al-Azhar UNIVERSITY

Faculty of Engineering

Computers and Systems Engineering Department

EXPERIMENT 1 – GitHub Source Control

OBJECTIVES

Upon completion of this lab, you will be able to:

- Manage your source code using GitHub Source control

MATERIALS/EQUIPMENT NEEDED

1. Git program
2. GitHub account
3. Web browser

INTRODUCTION

What is Git?

Git is a free, open-source **version control software**. It was created by Linus Torvalds in 2005. This tool is a version control system that was initially developed to work with several developers on the Linux kernel.

This basically means that Git is a content tracker. So, Git can be used to store content — and it is mostly used to store code because of the other features it provides. Real life projects generally have multiple developers working in parallel. So, they need a version control system like Git to make sure that there are no code conflicts between them.

Also, the requirements in such projects change often. So, a version control system allows developers to revert and go back to an older version of their code. The branch system in Git allows developers to work individually on a task (For example: One branch -> One task OR One branch -> One developer). Basically, think of Git as a small software application that controls your code base, if you're a developer.

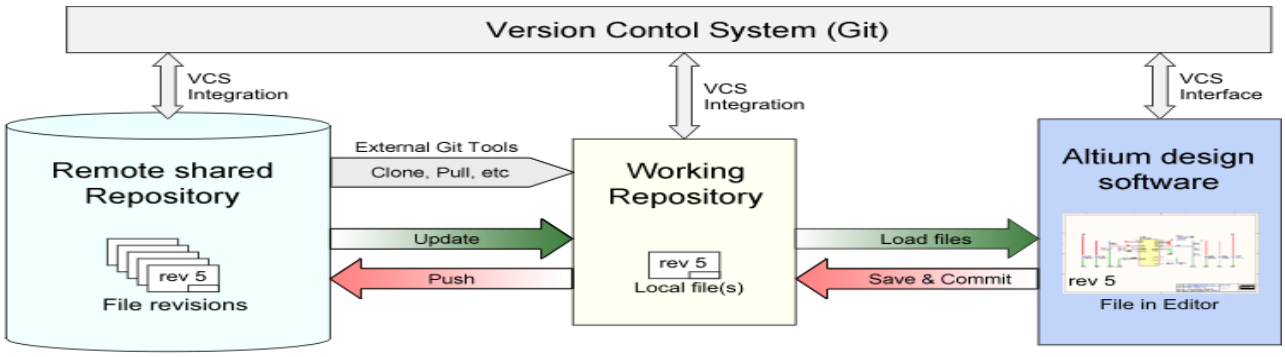


Figure 1 Shows how Git works

Git Repositories

If we want to start using Git, we need to know where to host our repositories.

A repository (or “Repo” for short) is a project that contains multiple files. In our case a repository will contain code-based files.

There are two ways you can host your repositories. One is online (on the cloud) and the second is offline (self-installed on your server).

There are three popular Git hosting services: GitHub (owned by Microsoft), GitLab (owned by GitLab) and BitBucket. We’ll use GitHub as our hosting service.

Git Commands

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline.`
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT LOG

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `"git log -5"` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author= "<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, HEAD, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

GIT PUSH

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all` flag. The `--tags` flag sends all of your local tags to the remote repo.

What is GitHub?

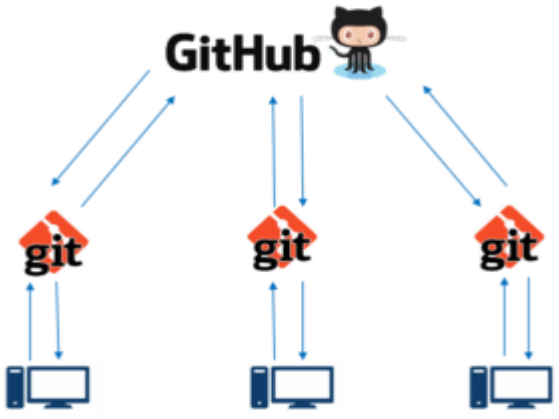
To be very crisp about what exactly is GitHub, it is a file or code-sharing service to collaborate with different people.

GitHub is a highly used software that is typically used for version control. It is helpful when more than just one person is working on a project. Say for example, a software developer team wants to build a website, and everyone has to update their codes simultaneously while working on the project. In this case, GitHub helps them to build a centralized repository where everyone can upload, edit, and manage the code files.

Why is Github so popular?

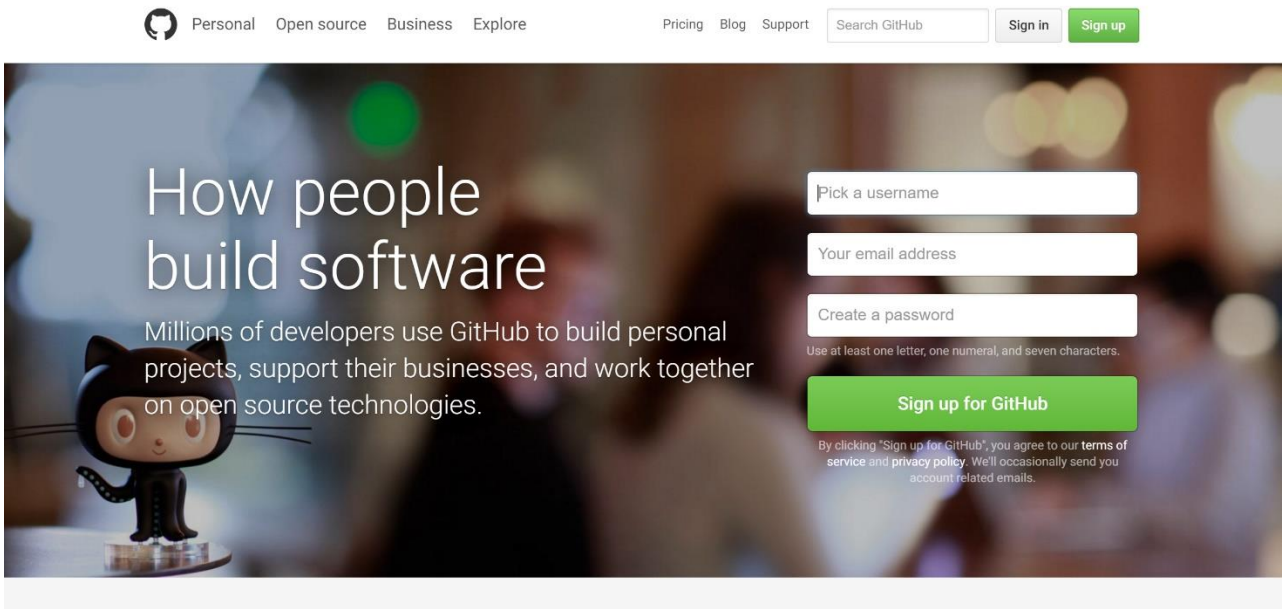
GitHub has various advantages, but many people often have a doubt as to why not use dropbox or any cloud-based system? Let me take the same example forward to answer this question. Say more than two software developers are working on the same file and they want to update it simultaneously. Unfortunately, the person who save the file first will get precedence over the others. While in Github, this is not the case. Github document the changes and reflect them in an organized manner to avoid any chaos between any of the files uploaded.

Therefore, using GitHub centralized repository, it avoids all the confusion and working on the same code becomes very easy. If you look at the image below, GitHub is a central repository and Git is a tool which allows you to create a local repository. Now people usually get confused between git and GitHub but its actually very different. Git is a version control tool that will allow you to perform all kinds of operations to fetch data from the central server or push data to it whereas GitHub is a core hosting platform for version control collaboration. GitHub is a company that allows you to host a central repository in a remote server.



PROCEDURE PREREQUISITE

Step 1: Create GitHub account



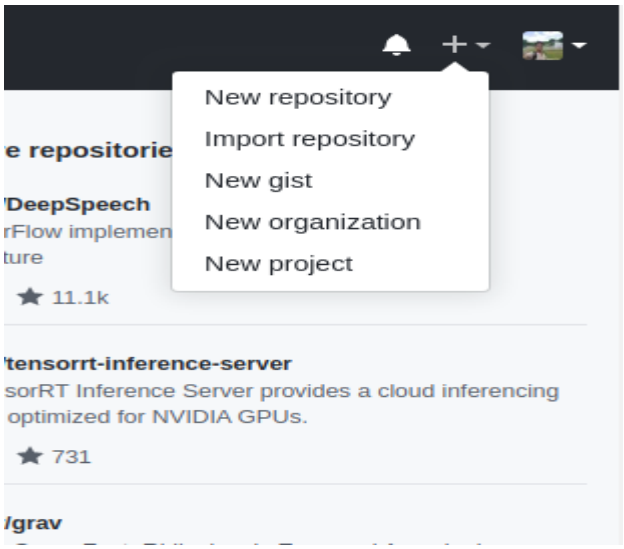
Step 2: Git installation

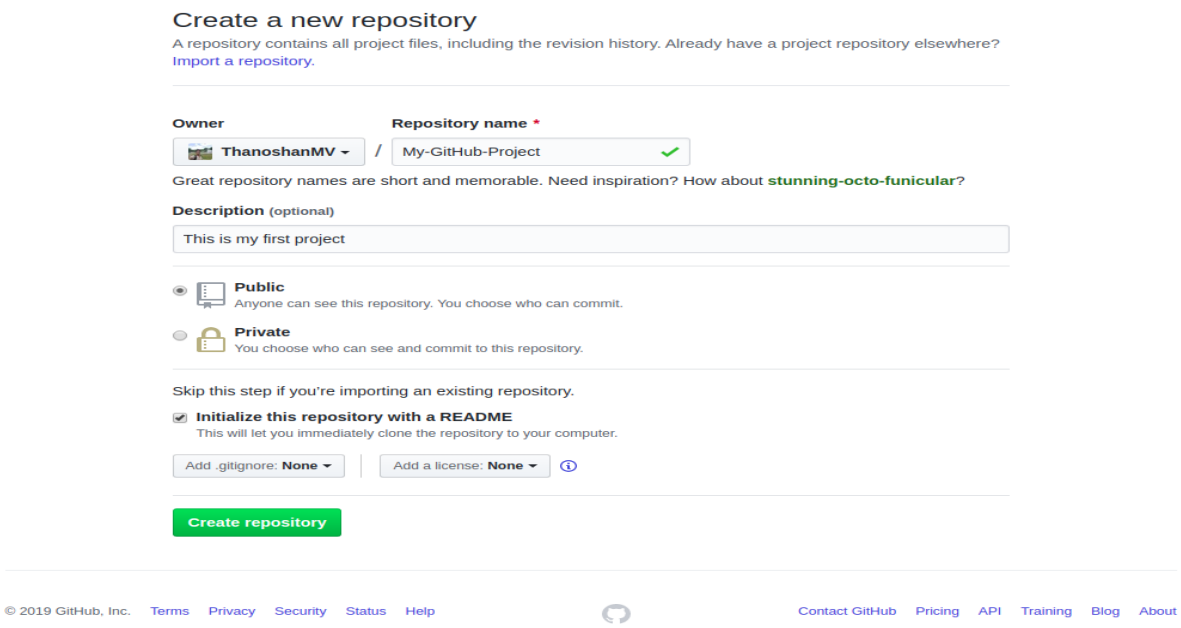
- **For Ubuntu:** `sudo apt-get install git`
- **For Windows :** download then install git using **https://git-scm.com/download/win**

PROCEDURE

TASK 1 : Create the repository, clone it to your PC, and work on it.

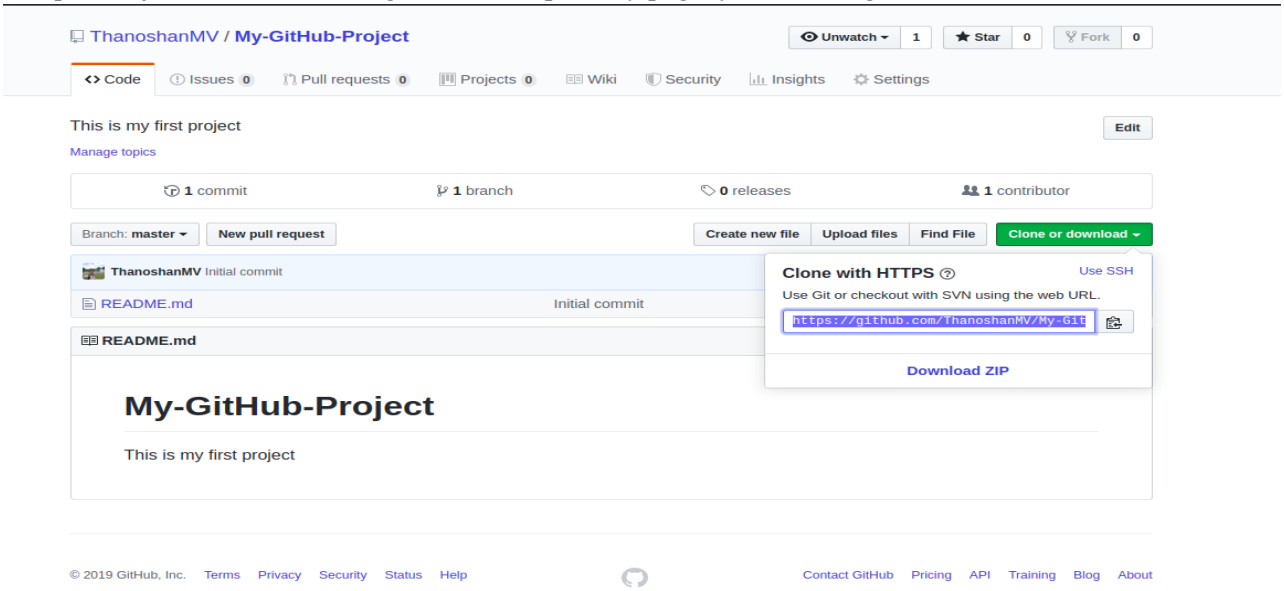
Step 1: Create Github the repository





Step 2: Clone Github repository

To clone a repository means that you're taking a repository that's on the server and cloning it to your computer – just like downloading it. On the repository page, you need to get the "HTTPS" address.



- **`git clone [HTTPS ADDRESS]`**

```
Cloning into 'My-GitHub-Project'...
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.
```

- **`cd [NAME OF REPOSITORY]`**

```
$ cd My-GitHub-Project
/My-GitHub-Project$
```

- **`git status`**

```
thanos18@lifecompanion:~/My-GitHub-Project$ 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.html

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

Step 3: Manage repository

- **`git add [FILENAME] [FILENAME] [...]`**
- **`git add sample.html`**

```
thanos18@lifecompanion:~/My-GitHub-Project$ git add sample.html
thanos18@lifecompanion:~/My-GitHub-Project$
```

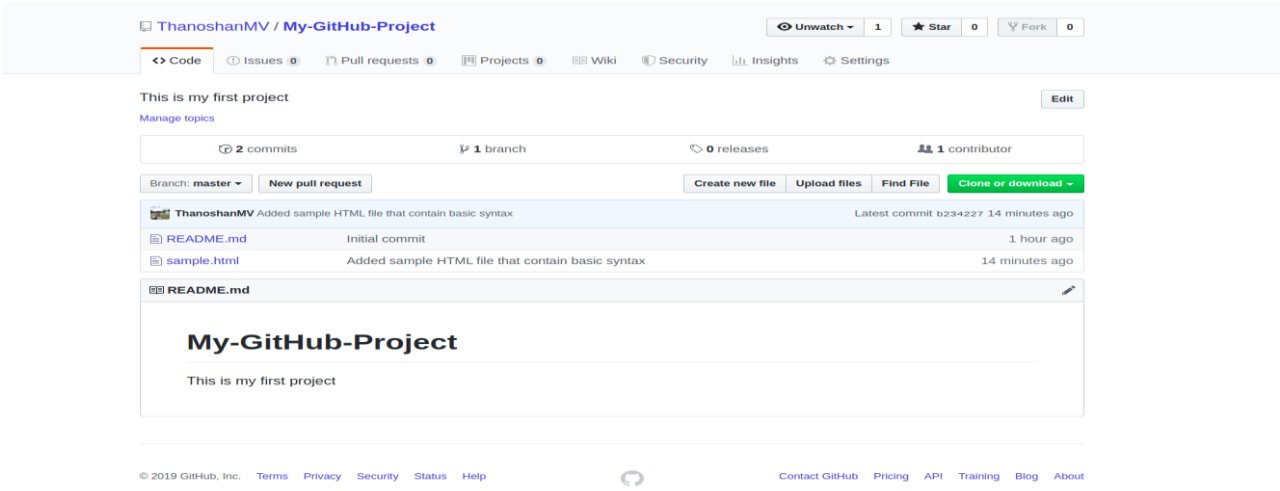
- **`git commit -m "Added sample HTML file that contain basic syntax"`**

```
thanos18@lifecompanion:~/My-GitHub-Project$  git commit -m "Added sample HTML fi
le that contain basic syntax"
[master b234227] Added sample HTML file that contain basic syntax
 1 file changed, 12 insertions(+)
 create mode 100644 sample.html
```

- **`git remote`**

```
thanos18@lifecompanion:~/My-GitHub-Project$ git remote
origin
```

- **`git push origin master`**



TASK 2: Work on your project locally then create the repository on GitHub and push it to remote.

Step 1: Create Your project on your local computer

```
thanos18@lifecompanion:~/Code-School/WebDev/Survey Form Project$
```

Step 2: Initiate your project using git

```
thanos18@lifecompanion:~/Code-School/WebDev/Survey Form Project$ git init
Initialized empty Git repository in /home/thanos18/Code-School/WebDev/Survey For
m Project/.git/
```

Step3: Manage your local repository

- **git status**

```
thanos18@lifecompanion:~/Code-School/WebDev/Survey Form Project$ git status
On branch master

No commits yet

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

        index.html
        style.css

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

- **git add [FILENAME] [FILENAME] [...]**
- **git add.**

```
thanos18@lifecompanion:~/Code-School/WebDev/Survey Form Project$ git add .
thanos18@lifecompanion:~/Code-School/WebDev/Survey Form Project$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   index.html
        new file:   style.css
```

- **git commit -m "Adding web Survey form"**

```
thanos18@lifecompanion:~/Code-School/WebDev/Survey Form Project$  git commit -m
"Adding web Survey form"
[master (root-commit) aa0a70a] Adding web Survey form
 2 files changed, 306 insertions(+)
 create mode 100644 index.html
 create mode 100644 style.css
```

Step4: Create Github Repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.

Owner ThanoshanMV / **Repository name** * SOCS-Survey-Form

Great repository names are short and memorable. Need inspiration? How about **miniature-umbrella**?

Description (optional)

A survey form that is created to my University's Society of Computer Science(SOCS).

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

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

Skip this step if you're importing an existing repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer.

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

Create repository

© 2019 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Training Blog About

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

or HTTPS SSH https://github.com/ThanoshanMV/SOCS-Survey-Form.git

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

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

```
echo "# SOCS-Survey-Form" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/ThanoshanMV/SOCS-Survey-Form.git
git push -u origin master
```

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

```
git remote add origin https://github.com/ThanoshanMV/SOCS-Survey-Form.git
git push -u origin master
```

...or import code from another repository

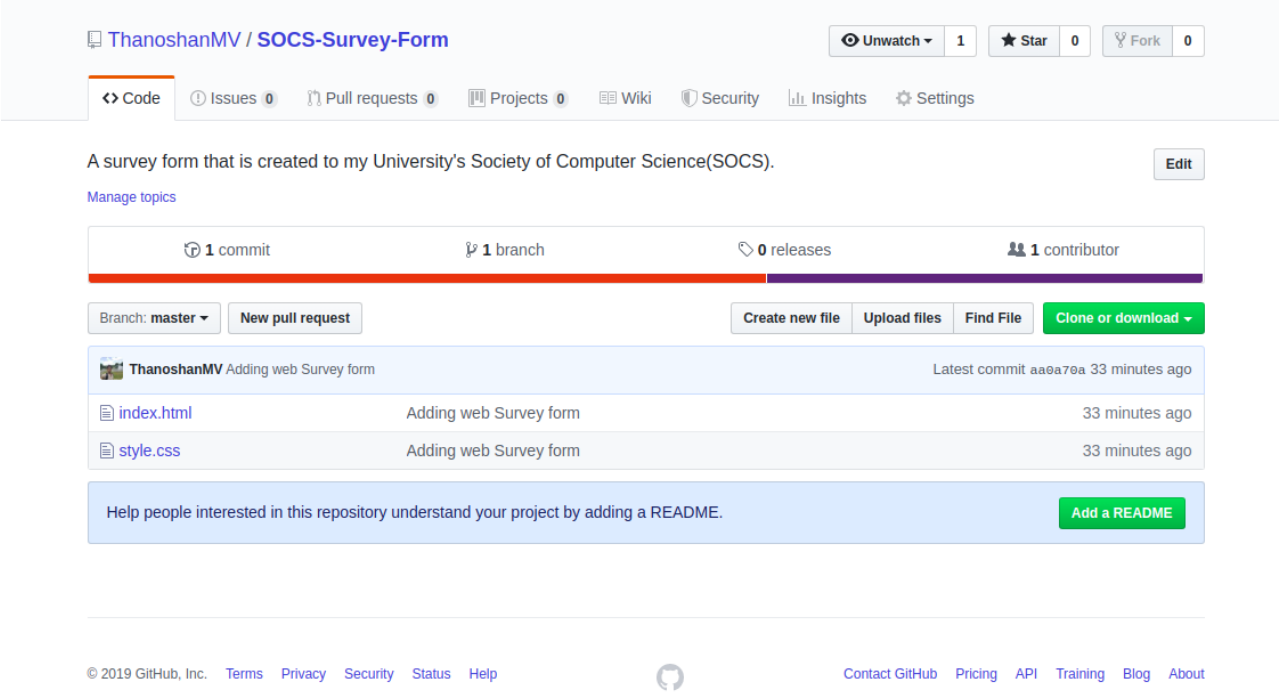
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

ProTip! Use the URL for this page when adding GitHub as a remote.

EXPERIMENT 1 GitHub Source Control

- `git remote add origin [HTTPS ADDRESS]`
- `git push origin master`

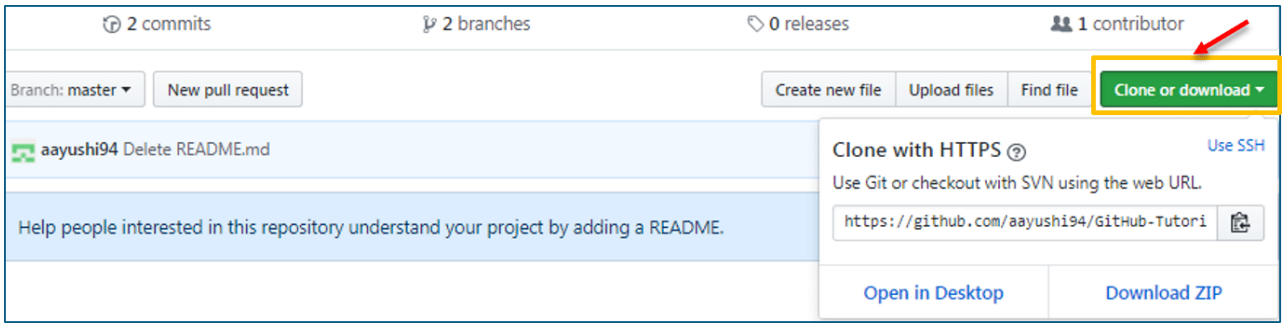


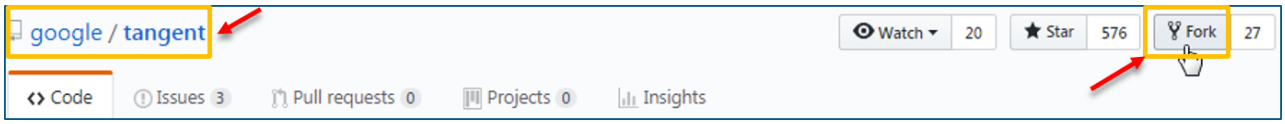
EXPERIMENT 1 GitHub Source Control

TASK 3: Contribute to someone's repository

Step 1: Clone repository

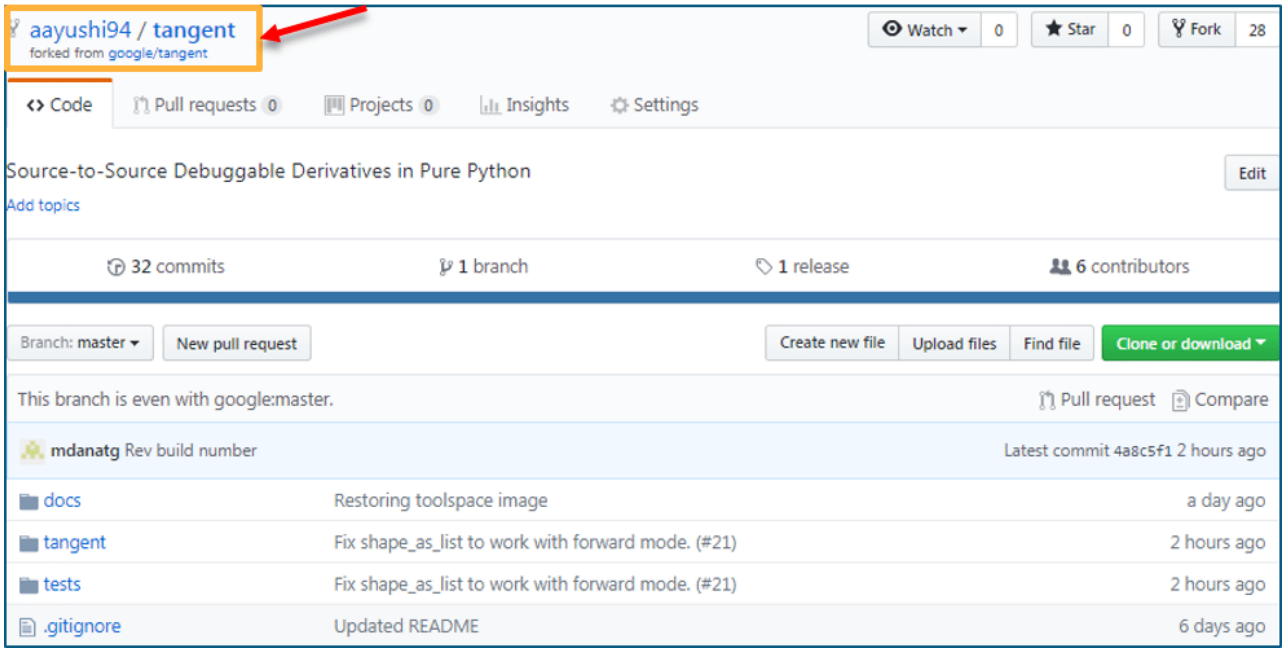
- Go to the repository on github. (Say it's by `myfriend`, and is called `the_repo`, then you'll find it at `https://github.com/myfriend/the_repo`.)
- Click the "Fork" button at the top right.
- You'll now have your own copy of that repository in your github account.





- Open a terminal/shell.
- Type

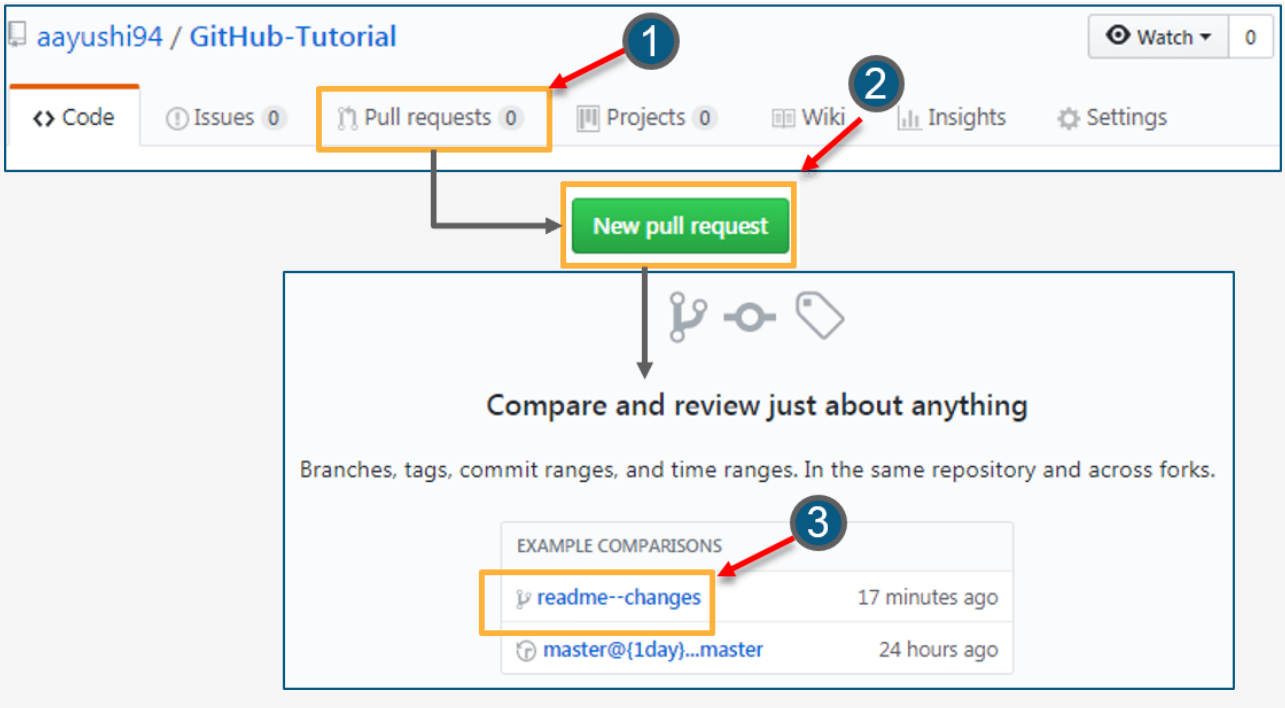
```
-   git clone git@github.com:username/the_repo

-   cd the_repo

-   git remote add myfriend git://github.com/myfriend/the_repo

-   git remote add repo_nickname git://github.com/myfriend/the_repo
```



Step 2: Pulling others' changes

Before you make further changes to the repository, you should check that your version is up to date relative to your friend's version. This will pull down and merge all of the changes that your friend has made.

```
- git pull myfriend master
```



Now push them back to your github repository.

```
- git push
```

Step3: Handling pull requests

```
- git remote add myfriend git://github.com/myfriend/the_repo

- git pull myfriend master

- git push
```

Step 4: Handling merge conflicts

One of the best features of git is its ability to easily merge multiple changes by different people.

Say you and a friend have both made changes to the same file at the same time. When you pull your friend's changes, git will often be able to combine them without any problem.

Sometimes, though, after you do

- `git pull myfriend master`You'll get a message like

```
Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

Automatic merge failed; fix conflicts and then commit the result.
```