COMP20050 - Software Engineering Project II

Version Control

Ravi Reddy Manumachu
ravi.manumachu@ucd.ie

UCD School of Computer Science.

Scoil na Ríomheolaíochta UCD.

COMP20050 - Weeks 1 & 2

Week1

Module Introduction

Software Project Specification

Week2

Version Control

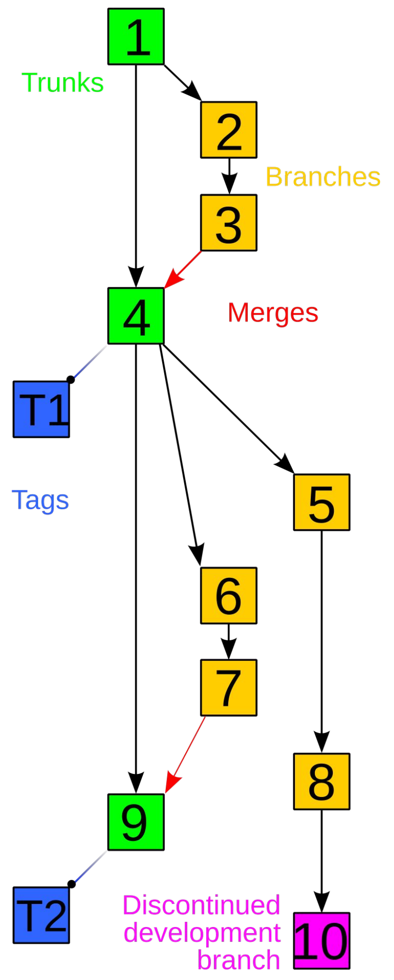
Software Architectural Design

Outline (Learning Objectives)

- Understand what is Version Control.
- Compare and contrast the types of Version Control Systems.
- Understand the basics of Git version control.
- Become familiar with GitHub.

What is Version Control?

- **Version control** is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- Files that are version controlled can nearly be any type of file on a computer (not just source code).



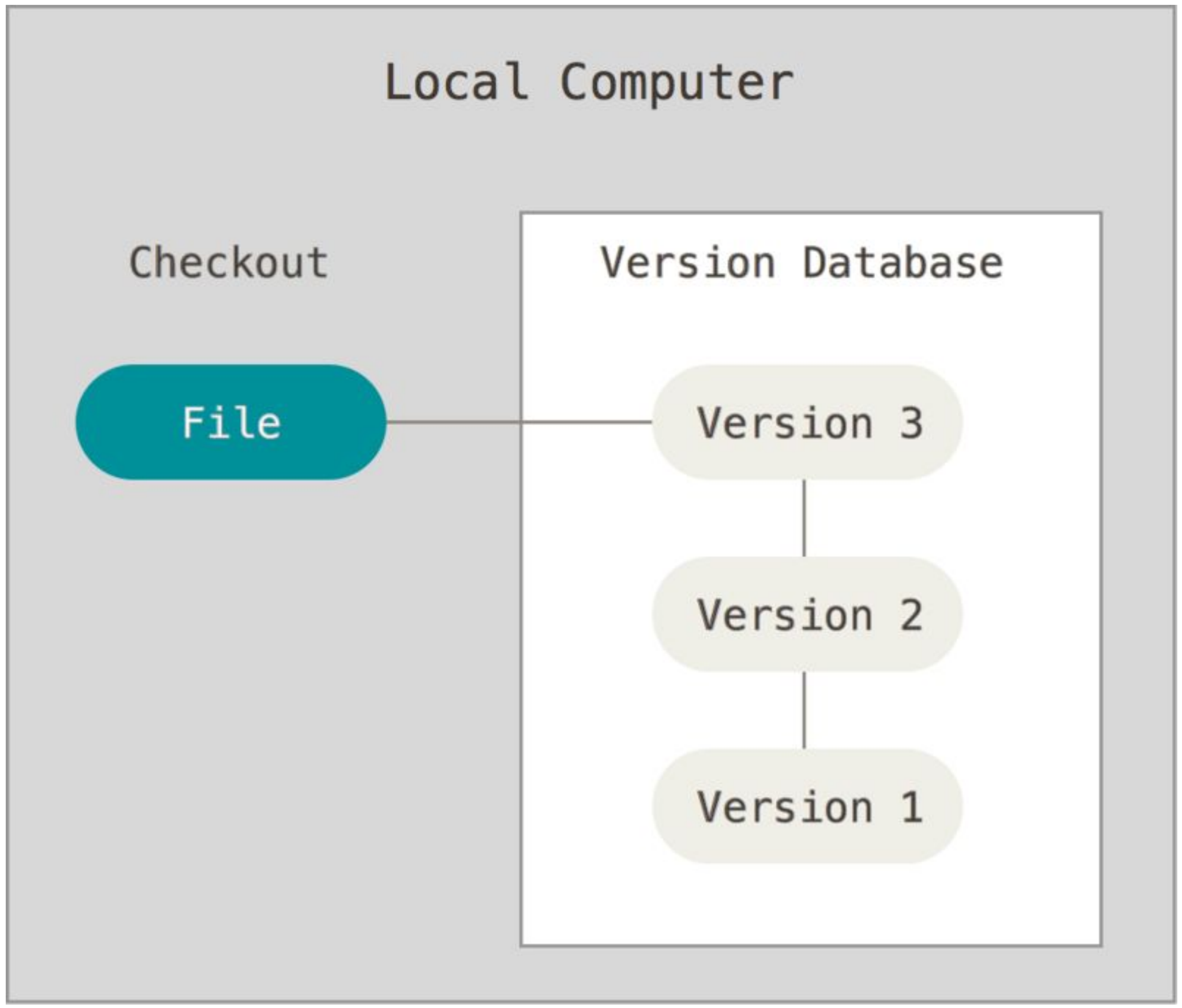
Version Control System

- A **Version Control System (VCS)** allows you to
    - Revert selected files back to a previous state.
    - Revert the entire project back to a previous state.
    - Compare changes over time.
    - See who last modified something that might be causing a problem.
    - Who introduced an issue and when.
    - And more.

.

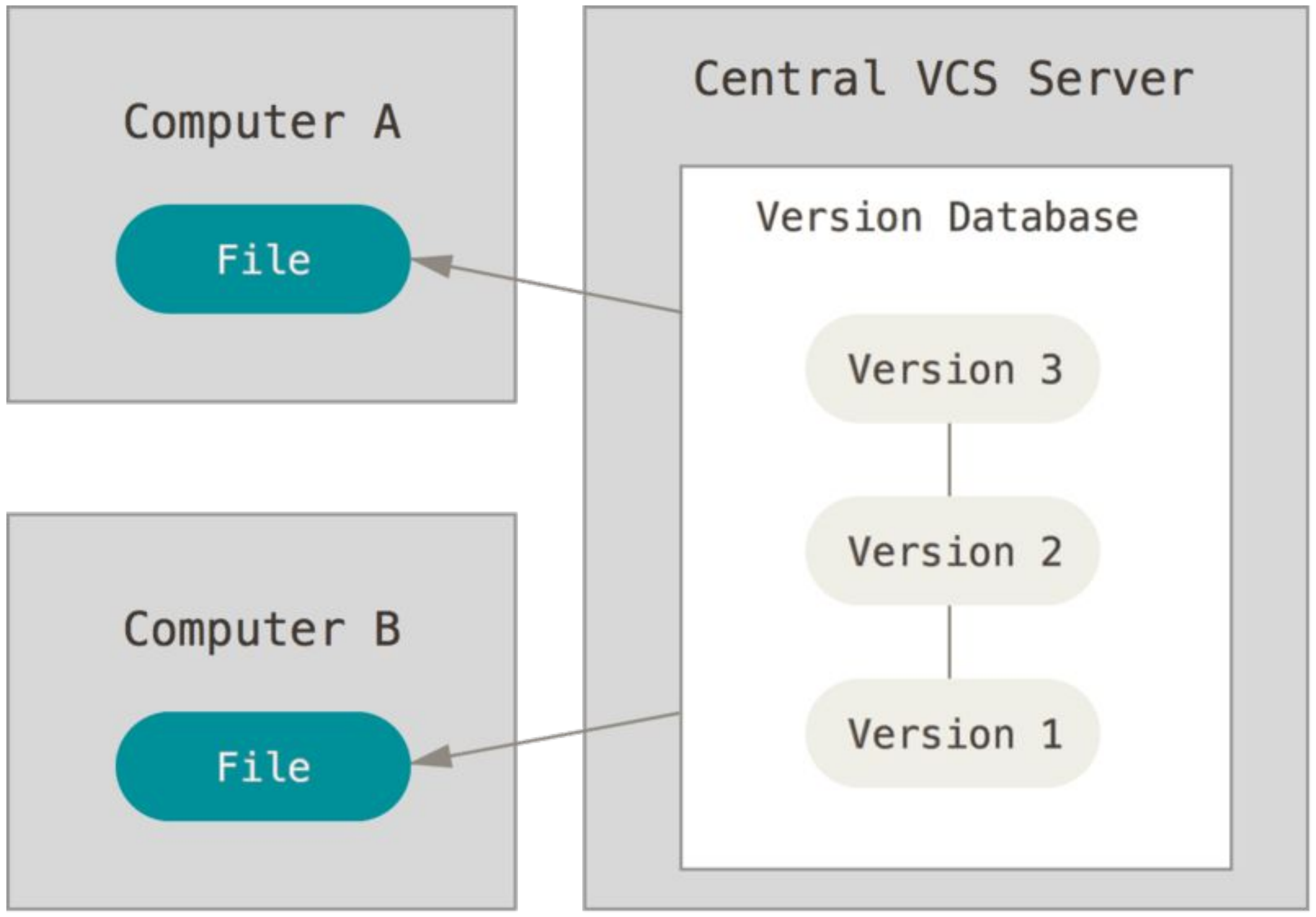
- All of the above with little overhead.

Types of VCS: Local VCS



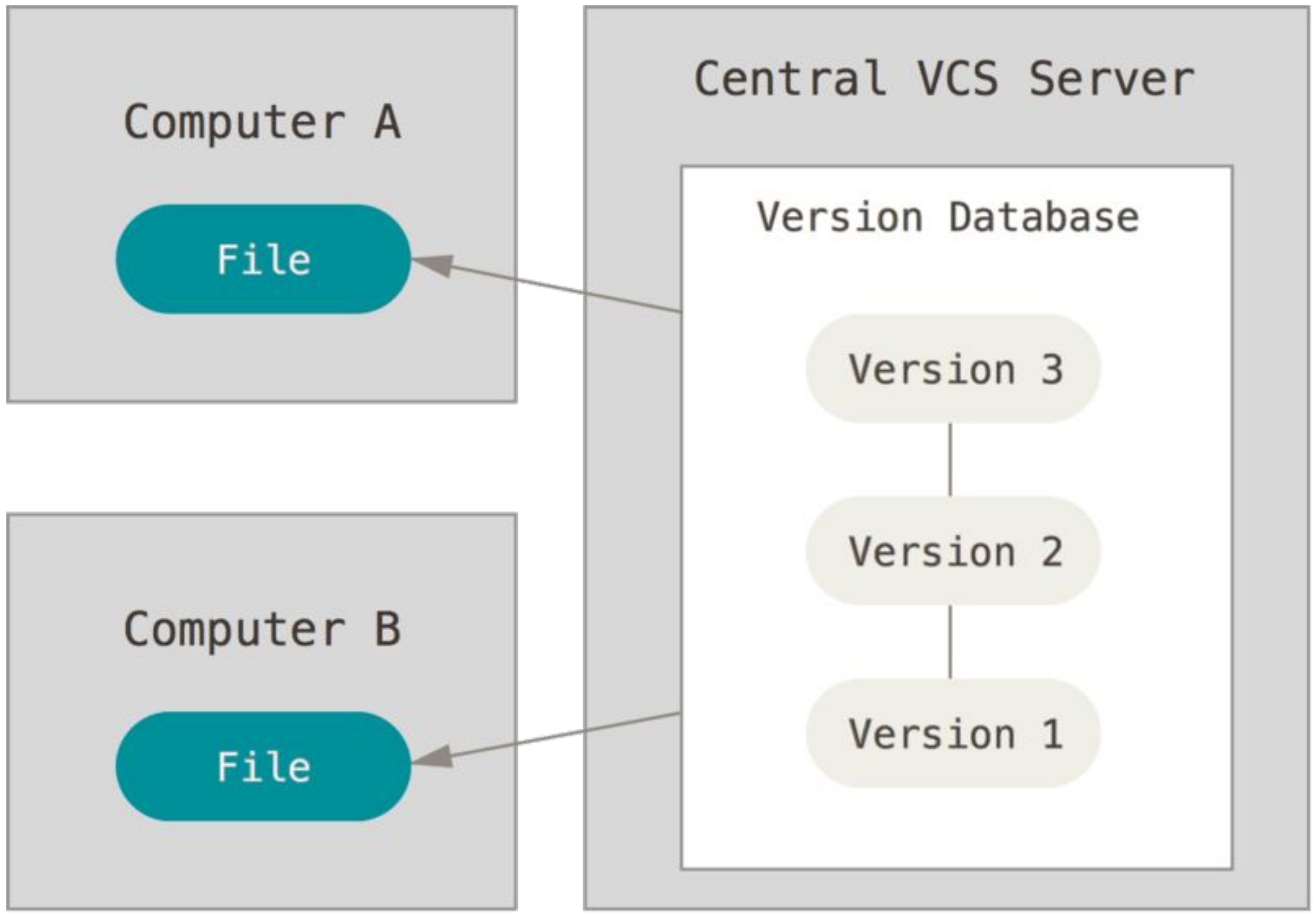
- A **local VCS** has a simple database locally that keeps all the changes to files under revision control.
- A popular local VCS is GNU Revision Control System (RCS). https://www.gnu.org/software/rcs/
- RCS works by keeping **patch sets** (the differences between files) in a special format on disk.
- It can recreate what any file looked like at any point in time by adding up all the patches.

Centralized VCS (1/2)



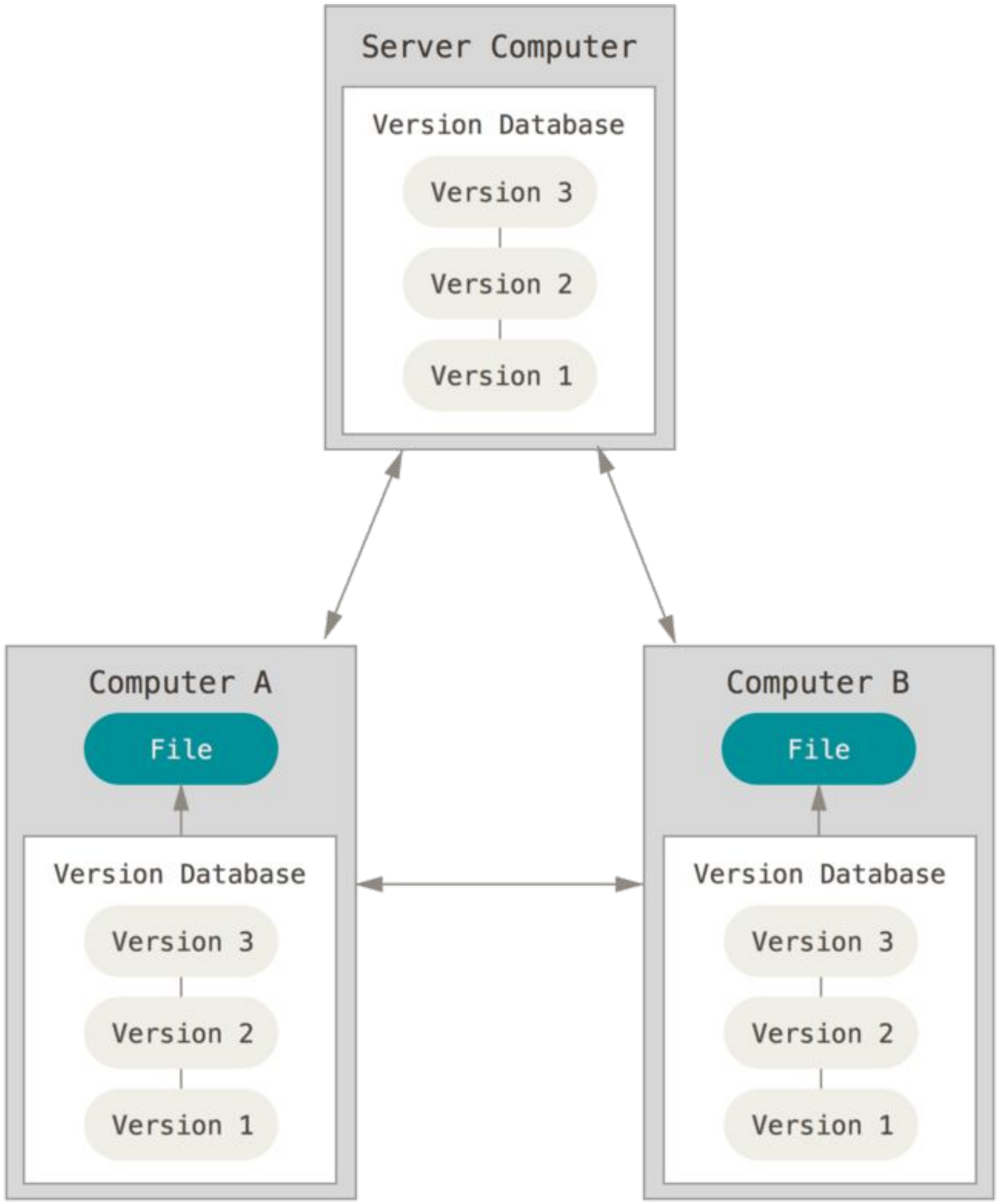
- Problem with **local VCSs** is they have no facility for developers to collaborate.
- Centralized VCSs have a single server that contains all the versioned files.
  - **CVS**
  - **Subversion**
  - **Perforce**
- Clients check out files from the server.

Centralized VCS (2/2)



- The most obvious downside is that the **centralized server represents single point of failure**.
- Server failure halts software development.
- All work is lost if server's database has no backups.

Distributed VCS



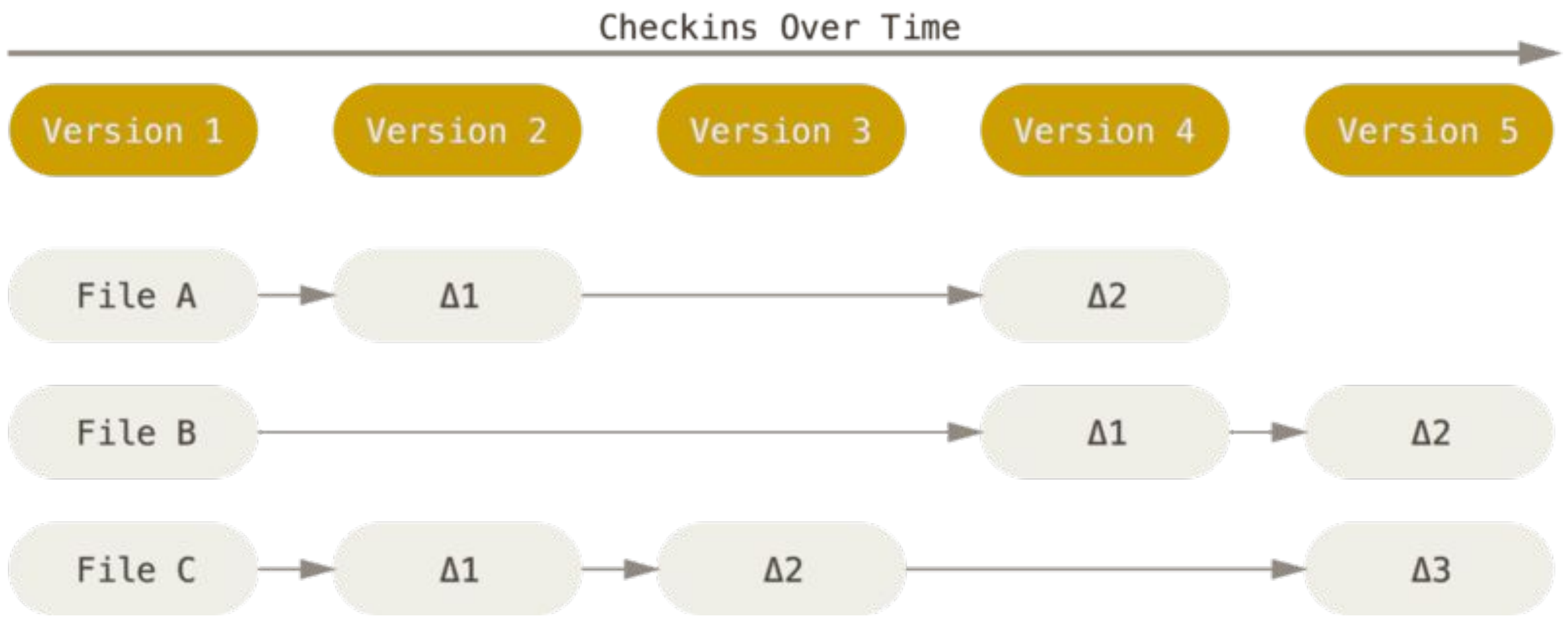
- In **distributed VCS,** clients fully mirror the repository, including its full history, unlike a snapshot in centralized VCS

- Therefore, every clone is a full backup of the server's data. Server can be restored from any of the clones.

- Examples:
  - **Git**
  - **Mercurial**
  - **Darcs**

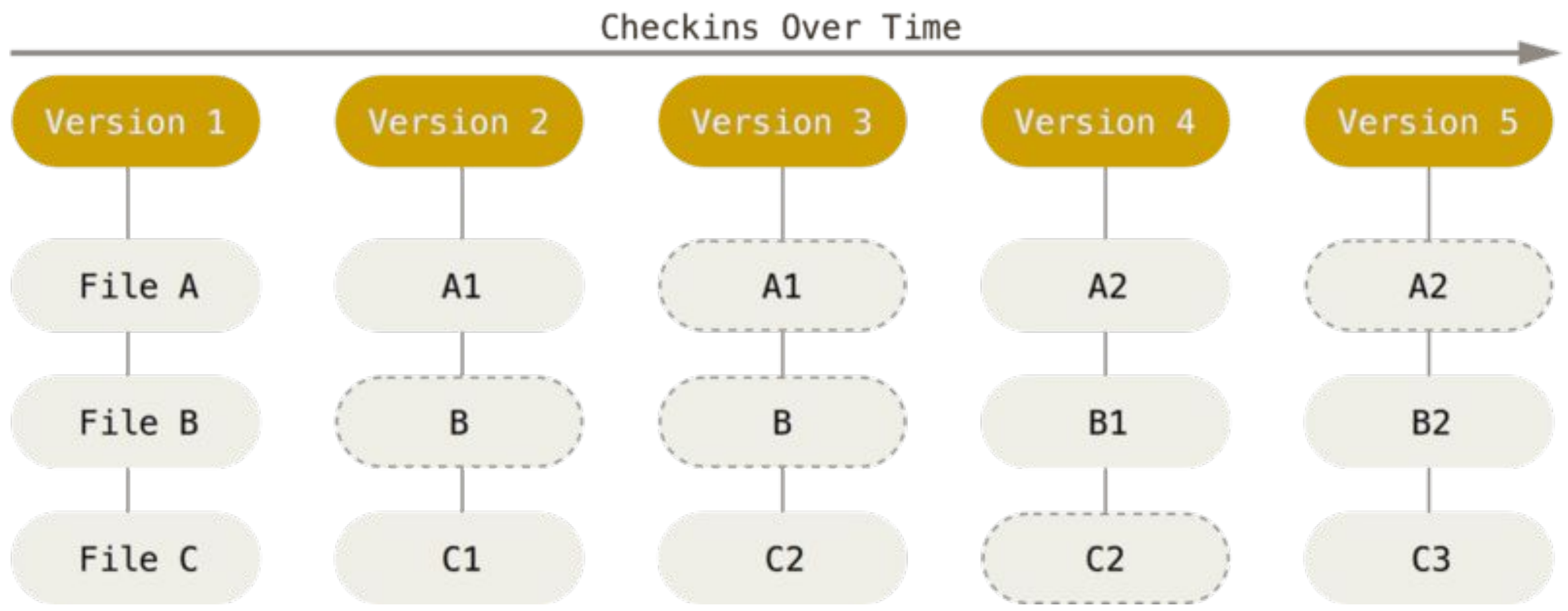
Git Basics

What is Git?



- The major difference between Git and any other VCS is that most other systems store information as a list of file-based changes (called **delta-based version control)**.

Git Snapshots



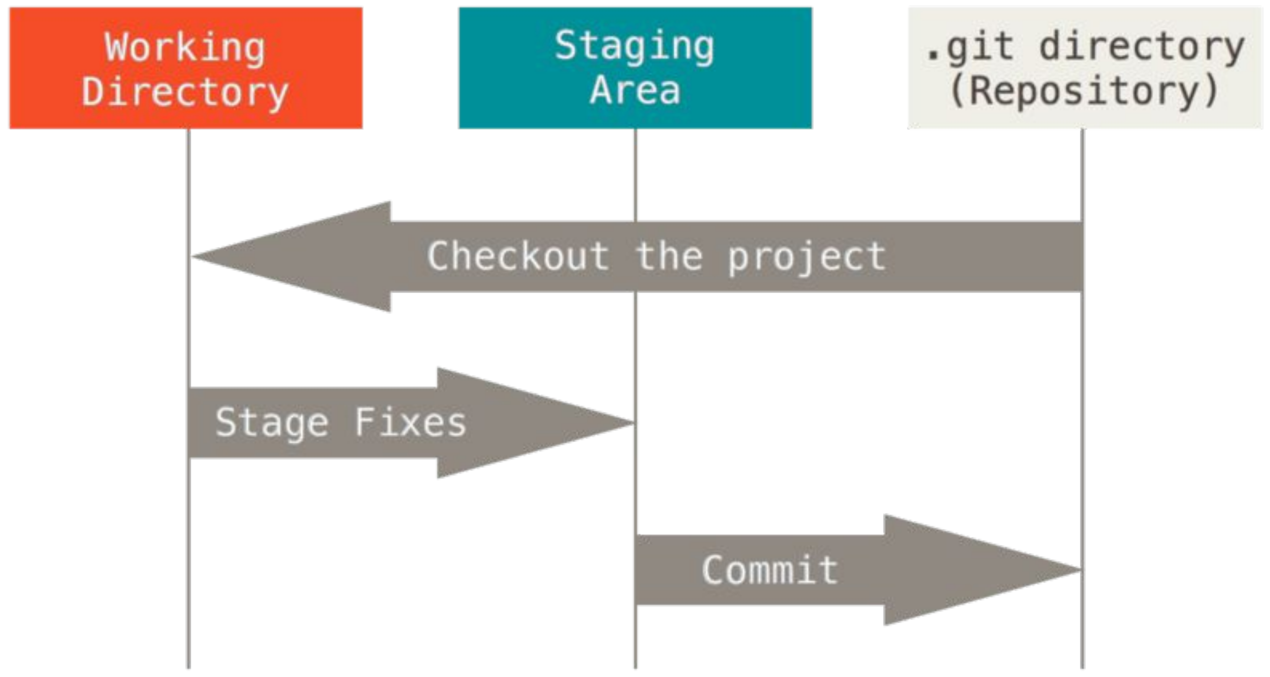
- **Git** thinks of its data more like a series of snapshots of a miniature filesystem.
- Every time you save the state of your project, **Git** basically stores a reference to that snapshot.

Nearly Every Git Operation is Local

- Most **Git** operations seem almost instantaneous because you have the entire history of the project on your local disk.

- If you're offline or on a airplane with no network connection, you can continue to do work (in your local copy) until you get to a network connection to upload.

The Three States



- Files in **Git** are in three main states: **modified**, **staged**, and **committed**.
- **Modified** means that you have changed the file but have not committed it to your database yet.
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
- **Committed** means that the data is safely stored in your local database.

Working tree, Staging area, and Git directory

- The **working tree** is a single checkout of one version of the project.

- The **staging area** is a file, generally contained in your **Git directory**, that stores information about what will go into your next commit.

- The **Git directory** (**.git/**) is where Git stores the metadata and object database for your project.

Basic Git Workflow

- You modify files in your **working tree**.
- You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the **staging area**.
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your **Git directory**.
- **However, these are just changes saved to your local repository.**

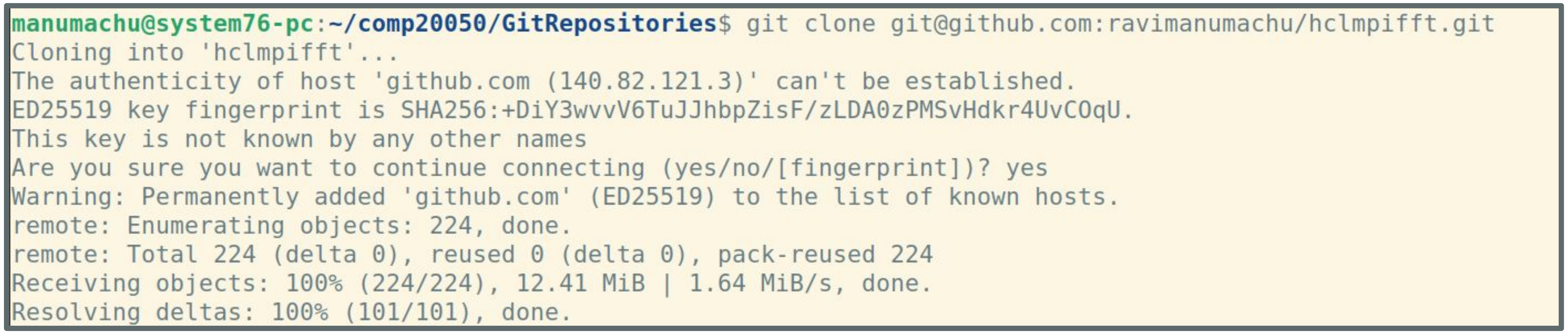
Getting a Git Repository

- Take a local directory that is currently not under version control, and turn it into a Git repository.

OR

- **Clone an existing Git repository.**
- **You will clone from GitHub in the current module.**

Cloning from GitHub



```
manumachu@system76-pc:~/comp20050/GitRepositories$ git clone git@github.com:ravimanumachu/hclmpifft.git
Cloning into 'hclmpifft'...
The authenticity of host 'github.com (140.82.121.3)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
remote: Enumerating objects: 224, done.
remote: Total 224 (delta 0), reused 0 (delta 0), pack-reused 224
Receiving objects: 100% (224/224), 12.41 MiB | 1.64 MiB/s, done.
Resolving deltas: 100% (101/101), done.
```

- Clone a repository using the command:

git clone <url>

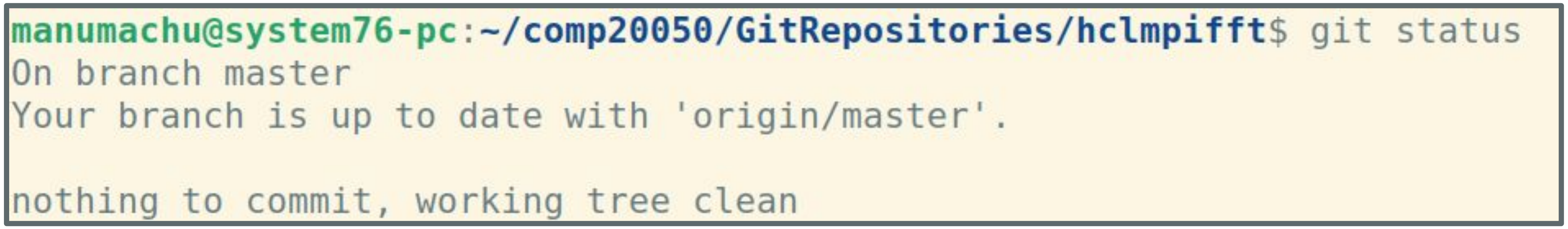
- For example, I cloned my **hclmpifft** repo from GitHub as follows:

git clone git@github.com:ravimanumachu/hclmpifft.git

- To give a different name for your local repository,

git clone git@github.com:ravimanumachu/hclmpifft.git mympifft

Recording Changes to the Git Repository



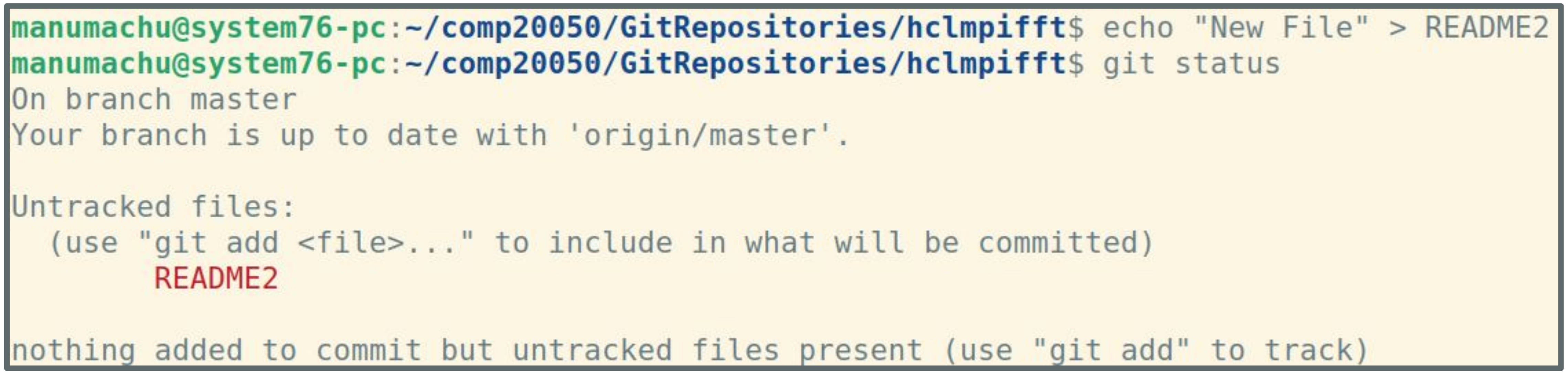
```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

- The main tool to determine the state of the files:

git status

- This tool guides you at every stage of the version control process.
- Output above means none of your tracked files are modified.

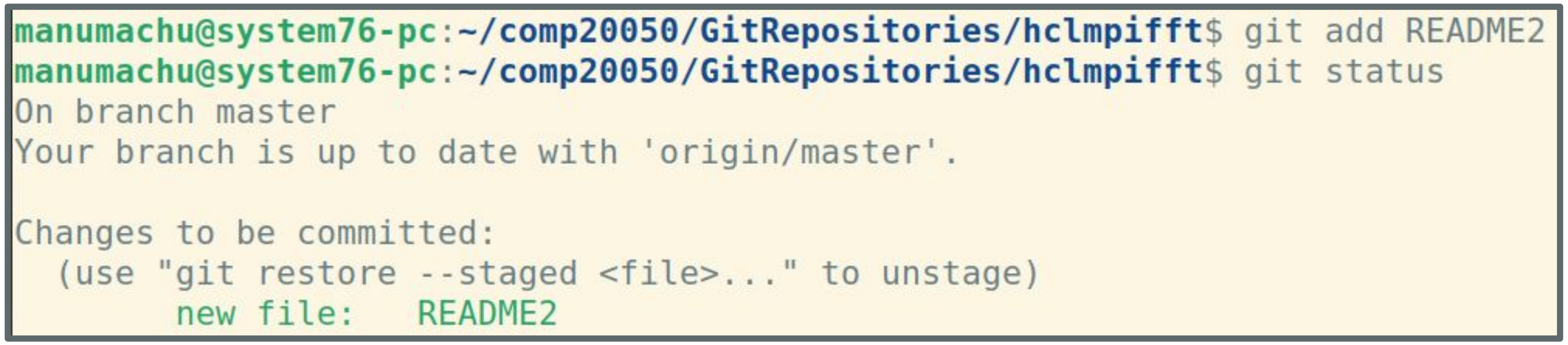
Tracking New Files (1/2)



```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ echo "New File" > README2
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ 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)
        README2

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

- I have added a new file to the project, **README2**.
- **git status** shows the untracked file.
- **Git won't include README2 in your commit snapshots until you explicitly tell it to do so.**

Tracking New Files (2/2)



```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git add README2
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   README2
```

- To stage README2, run the following command.

git add

- **git add** is a multipurpose command used to track new files, stage files, and marking merge-conflicted files as resolved.

Committing Your Changes

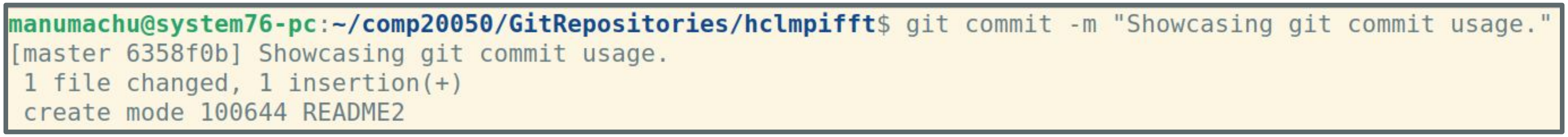
- You can commit your changes using one of the following commands:

```
git commit
git commit -m "<commit message>"
```

- Any files you have created or modified after **git add** will not go into this commit.

- You must execute **git add** again followed by **git commit**.

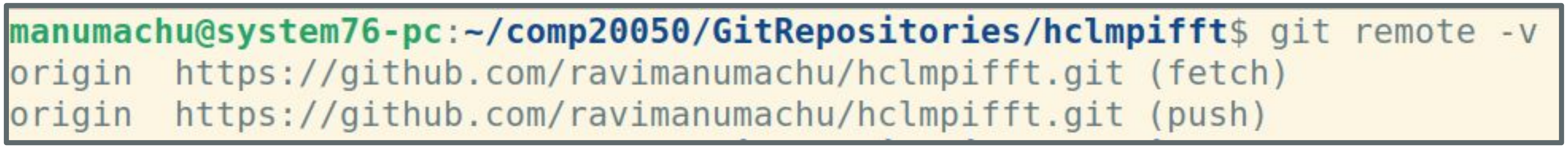
Committing Your Changes



```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git commit -m "Showcasing git commit usage."
[master 6358f0b] Showcasing git commit usage.
 1 file changed, 1 insertion(+)
 create mode 100644 README2
```

- git commit gives following output:
  - Which branch you committed to (**master**).
  - What SHA-1 checksum the commit has (**6358f0b**), which becomes the commit identifier.
  - How many files were changed, and
  - Statistics about lines added and removed in the commit.

Working with Remotes



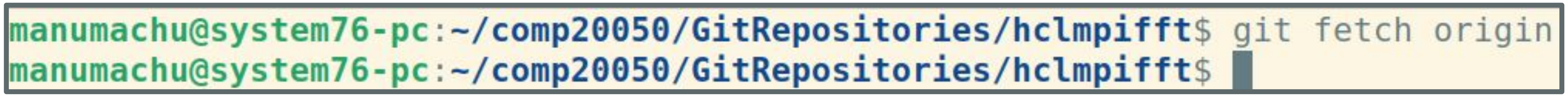
```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git remote -v
origin  https://github.com/ravimanumachu/hclmpifft.git (fetch)
origin  https://github.com/ravimanumachu/hclmpifft.git (push)
```

- Collaborating with your group members involves managing the GitHub remote repository.

- To see which remote servers you have configured, you can run the **git remote** command.

- **origin** is the default name Git gives to the server you cloned from.

Fetching and Pulling from Your Remotes



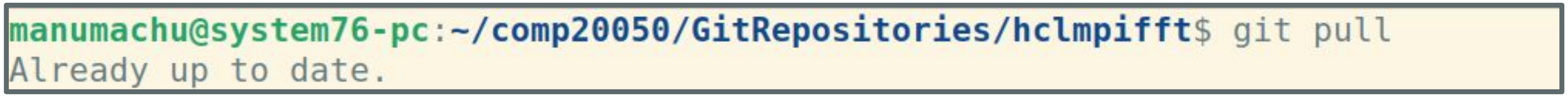
```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git fetch origin
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ 
```

- To get data from your remote projects, you can run:

git fetch <remote>

- **git fetch origin** fetches any new work that has been pushed to that server since you cloned (or last fetched from) it.
- The **git fetch** command only downloads the data to your local repository .
- It doesn't automatically merge it with any of your work or modify what you're currently working on.

Merging from Your Remotes

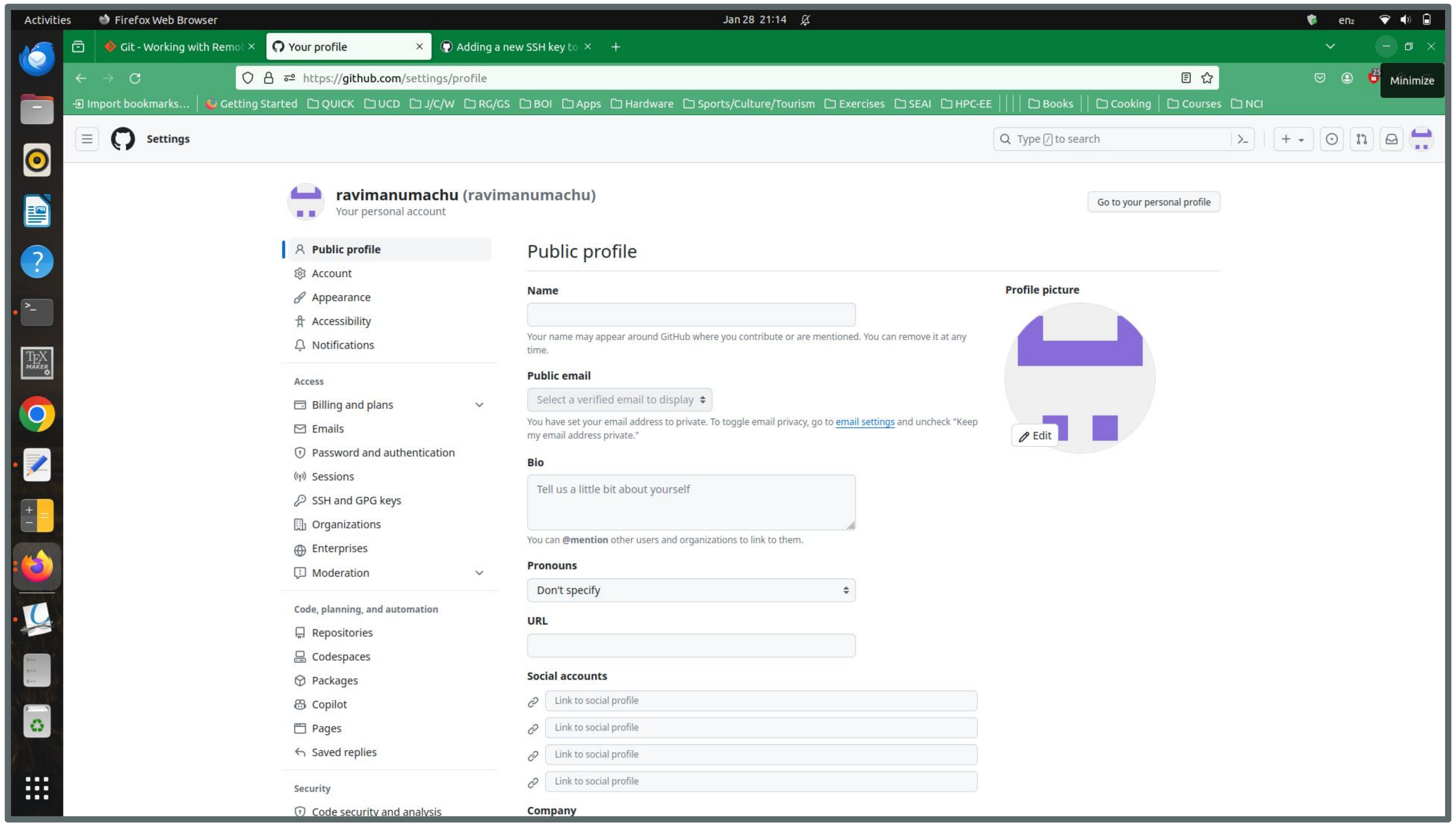


```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git pull
Already up to date.
```

- **git pull** generally fetches data from the server you originally cloned from.
- And automatically tries to merge it into the code you're currently working on.
- **Already up to date** means that all the changes from the remote branch have already been merged in your local branch.

Pushing to Your Remotes: Add SSH Key

- Before you push your changes to the remote repository, you must set up your SSH key in GitHub.
- Go to: https://github.com/settings/profile



Public profile

Name

Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Public email

Select a verified email to display

You have set your email address to private. To toggle email privacy, go to email settings and uncheck "Keep my email address private."

Bio

Tell us a little bit about yourself

You can @mention other users and organizations to link to them.

Pronouns

Don't specify

URL

Social accounts

Link to social profile

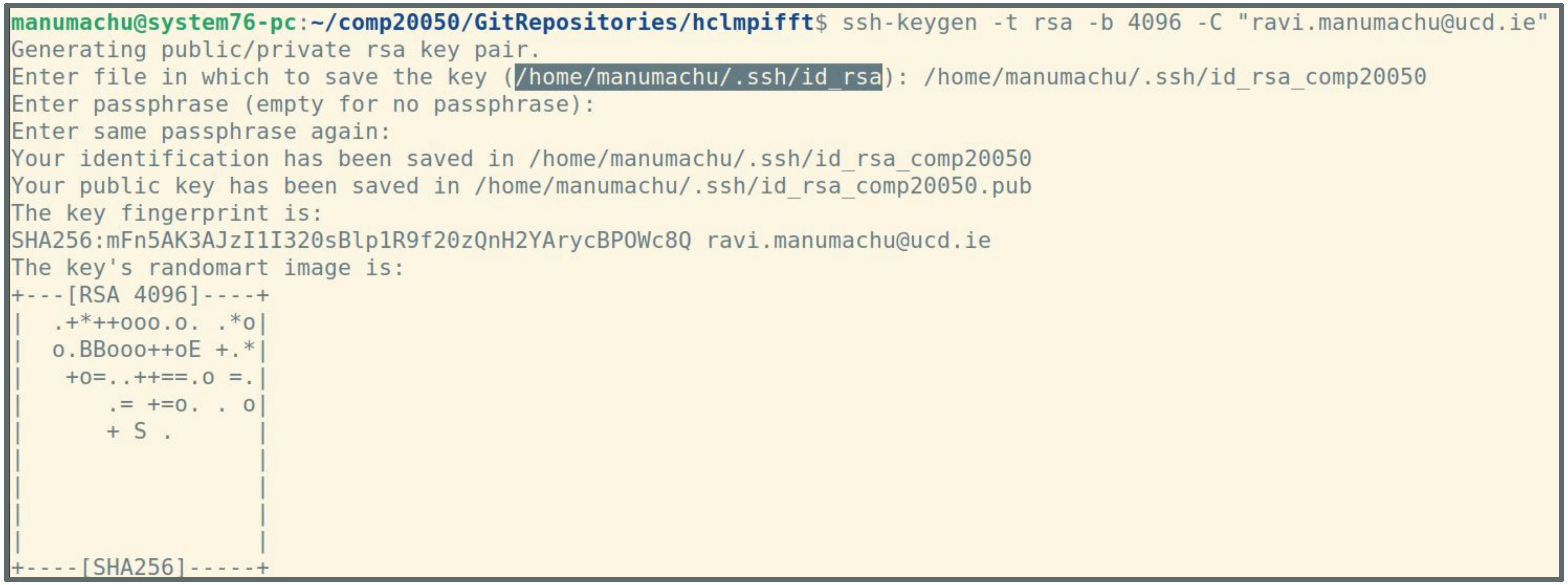
Link to social profile

Link to social profile

Link to social profile

Company

Generate SSH Key

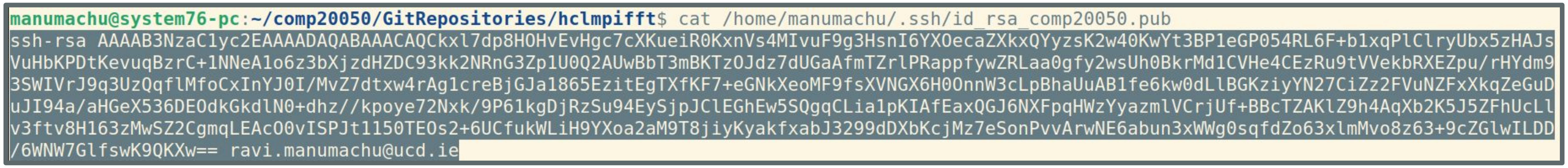


```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ ssh-keygen -t rsa -b 4096 -C "ravi.manumachu@ucd.ie"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/manumachu/.ssh/id_rsa): /home/manumachu/.ssh/id_rsa_comp20050
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/manumachu/.ssh/id_rsa_comp20050
Your public key has been saved in /home/manumachu/.ssh/id_rsa_comp20050.pub
The key fingerprint is:
SHA256:mFn5AK3AJzI1I320sBlp1R9f20zQnH2YArycBPOWc8Q ravi.manumachu@ucd.ie
The key's randomart image is:
+---[RSA 4096]----+
|   .+*++ooo.o.  .*o|
|   o.BBooo++oE +.*|
|    +o=..++==.o =.|
|       .= +=o.  . o|
|       + S  .     |
|                  |
|                  |
|                  |
|                  |
+----[SHA256]-----+
```

- Generate SSH key in a terminal using the command below (use your personal email address):

ssh-keygen -t rsa -b 4096 -C "ravi.manumachu@ucd.ie"

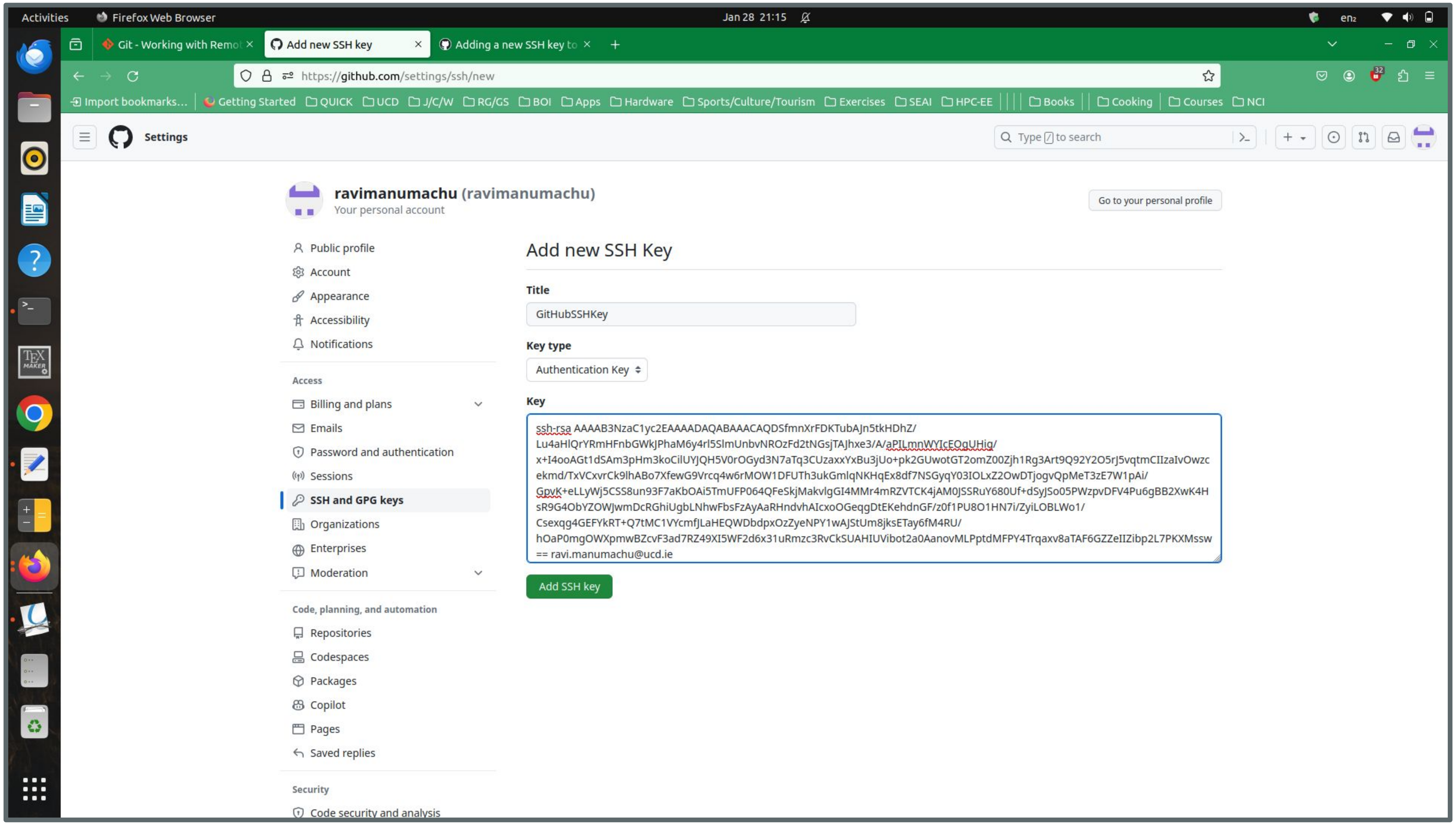
Copy SSH Key



```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ cat /home/manumachu/.ssh/id_rsa_comp20050.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCkxl7dp8HOHvEvHgc7cXKueiR0KxnVs4MIvuF9g3HsnI6YXOecaZXkxQYyzsK2w40KwYt3BP1eGP054RL6F+b1xqPlClryUbx5zHAJs
VuHbKPDtKevuqBzrC+1NNeA1o6z3bXjzdHZDC93kk2NRnG3Zp1U0Q2AUwBbT3mBKTzOJdz7dUGaAfmTZrlPRappfywZRLaa0gfy2wsUh0BkrMd1CVHe4CEzRu9tVVekbRXEZpu/rHYdm9
3SWIVrJ9q3UzQqflMfoCxInYJ0I/MvZ7dtxw4rAg1creBjGJa1865EzitEgTXfKF7+eGNkXeoMF9fsXVNGX6H0OnnW3cLpBhaUuAB1fe6kw0dLlBGKziyYN27CiZz2FVuNZFxXkqZeGuD
uJI94a/aHGeX536DEOdkGkdlN0+dhz//kpoye72Nxk/9P61kgDjRzSu94EySjpJClEGhEw5SQgqCLia1pKIAfEaxQGJ6NXFpqHWzYyazmlVCrjUf+BBcTZAKlZ9h4AqXb2K5J5ZFhUcLl
v3ftv8H163zMwSZ2CgmqLEAcO0vISPJt1150TEOs2+6UCfukWLiH9YXoa2aM9T8jiyKyakfxabJ3299dDXbKcjMz7eSonPvvArwNE6abun3xWWg0sqfdZo63xlmMvo8z63+9cZGlwILDD
/6WNW7GlfswK9QKXw== ravi.manumachu@ucd.ie
```

- Copy the SSH key and add it in GitHub.

Add SSH Key to GitHub


ravimanumachu (ravimanumachu)
Your personal account

Add new SSH Key

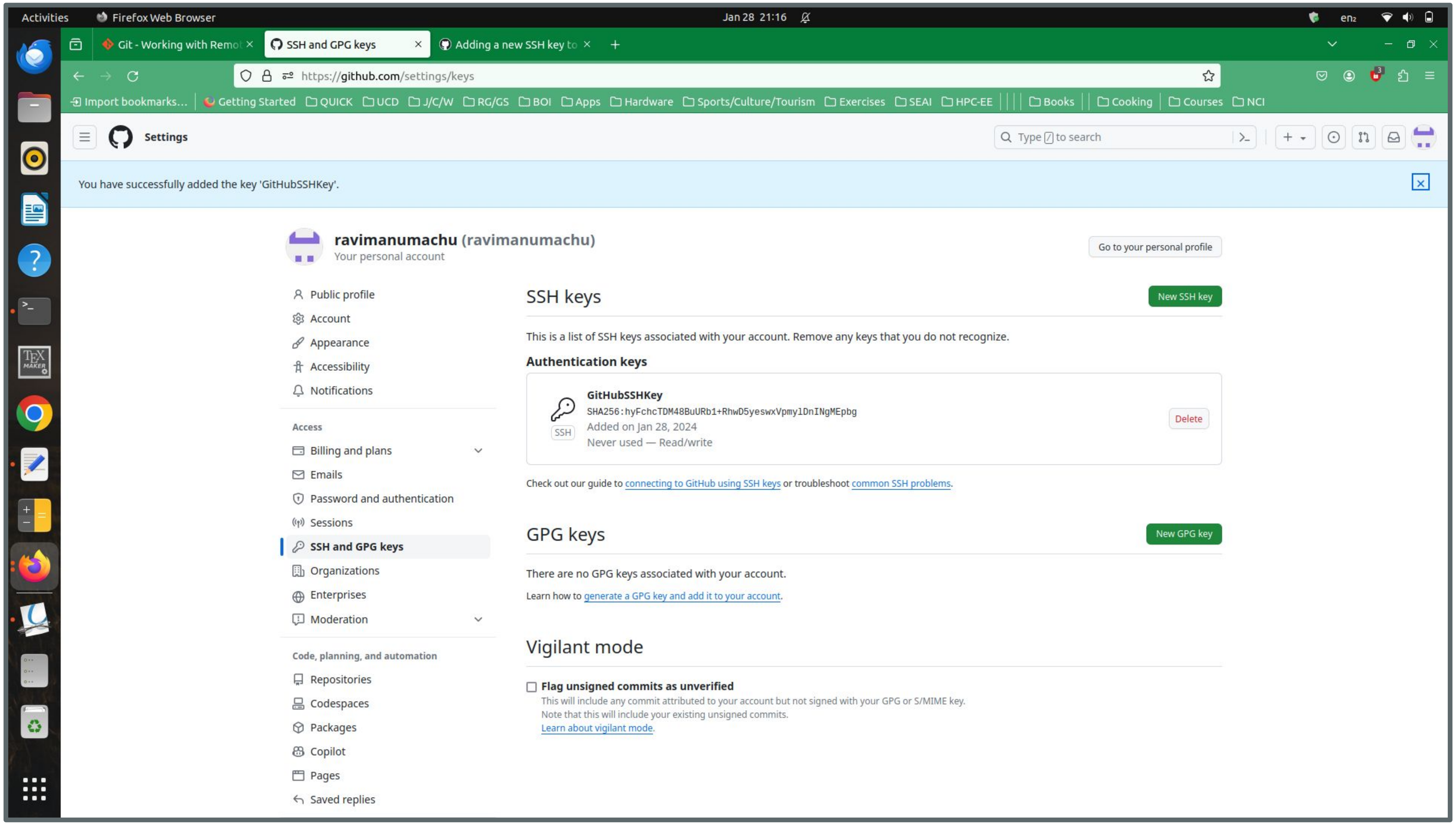
Title: GitHubSSHKey

Key type: Authentication Key

Key:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDSfmnXrFDKTubAJn5tkHDhZ/
Lu4aHlQrYRmHFnbGWkJPhaM6y4rl5SlmUnbvNROzFd2tNGsjTAJhxe3/A/aPILmnWYIcEOgUHig/
x+I4ooAGt1dSAm3pHm3koCilUYJQH5V0rOGyd3N7aTq3CUzaxxYxBu3jUo+pk2GUwotGT2omZ00Zjh1Rg3Art9Q92Y2O5rJ5vqtmCIIzaIvOwzc
ekmd/TxVCxvrCk9lhABo7XfewG9Vrcq4w6rMOW1DFUTh3ukGmlqNKHqEx8df7NSGyqY03IOLxZ2OwDTjogvQpMeT3zE7W1pAi/
GpvK+eLLyWj5CSS8un93F7aKbOAi5TmUFP064QFeSkjMakvlgGI4MMr4mRZVTCK4jAM0JSSRuY680Uf+dSyJSo05PWzpvDFV4Pu6gBB2XwK4H
sR9G4ObYZOWJwmDcRGhiUgbLNhwFbsFzAyAaRHndvhAIcxoOGeqgDtEKehdnGF/z0f1PU8O1HN7i/ZyiLOBLWo1/
Csexqg4GEFYkRT+Q7tMC1VYcmfJLaHEQWDbdpxOzZyeNPY1wAJStUm8jksETay6fM4RU/
hOaP0mgOWXpmwBZcvF3ad7RZ49XI5WF2d6x31uRmzc3RvCkSUAHIUVibot2a0AanovMLPptdMFPY4Trqaxv8aTAF6GZZeIIZibp2L7PKXMssw
== ravi.manumachu@ucd.ie
```

SSH Key Added



You have successfully added the key 'GitHubSSHKey'.

ravimanumachu (ravimanumachu)
Your personal account

SSH keys

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

Authentication keys

GitHubSSHKey
SSH
`SHA256:hyFchcTDM48BuURb1+RhwD5yeswxVpmylDnINgMEpbg`
Added on Jan 28, 2024
Never used — Read/write

Check out our guide to connecting to GitHub using SSH keys or troubleshoot common SSH problems.

GPG keys

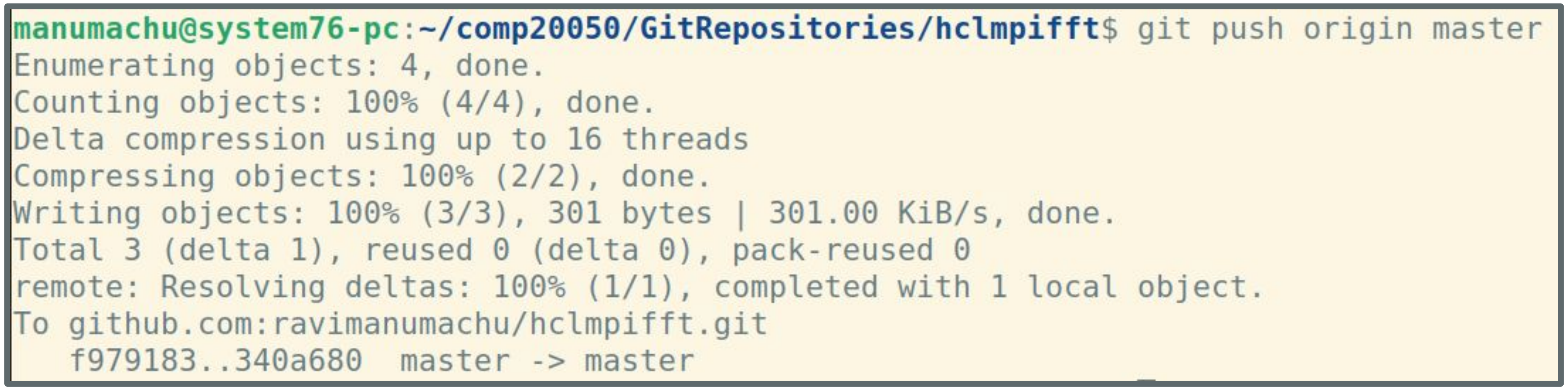
There are no GPG keys associated with your account.

Learn how to generate a GPG key and add it to your account.

Vigilant mode

- [ ] **Flag unsigned commits as unverified**
  This will include any commit attributed to your account but not signed with your GPG or S/MIME key.
  Note that this will include your existing unsigned commits.
  Learn about vigilant mode.

Pushing to Your Remotes



```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 301.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:ravimanumachu/hclmpifft.git
   f979183..340a680  master -> master
```

- Now push your changes to the remote repository using the command below:

git push <remote> <branch>

- If you want to push your **master branch** to your **origin server**, then you can run the following command to push any commits:

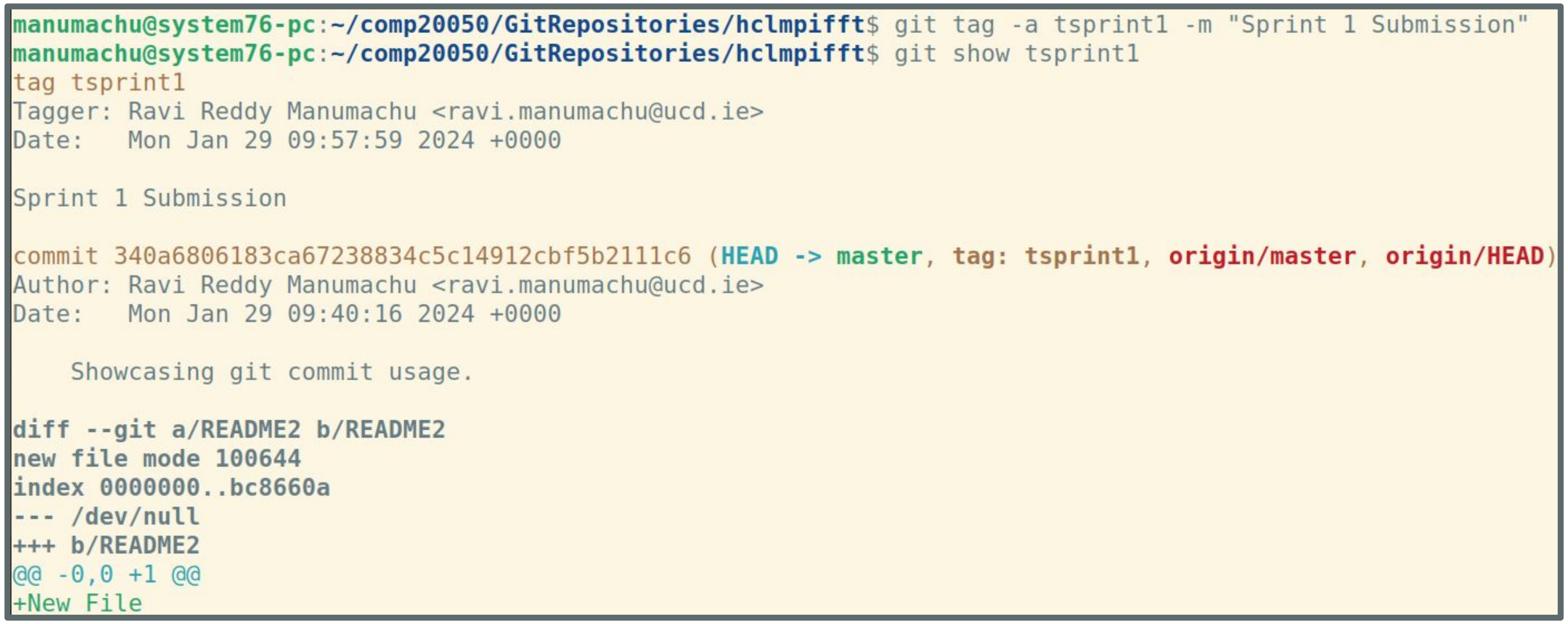
git push origin master

Git Tagging

Tagging

- Git allows **tagging** specific points in a repository's history as being important.
- Developers use this functionality to mark release points.
- **You will use this option to tag sprint releases and the final project release.**
    - **5 tags (4 sprint submissions, 1 final submission).**

Create Tags



```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git tag -a tsprint1 -m "Sprint 1 Submission"
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git show tsprint1
tag tsprint1
Tagger: Ravi Reddy Manumachu <ravi.manumachu@ucd.ie>
Date:   Mon Jan 29 09:57:59 2024 +0000

Sprint 1 Submission

commit 340a6806183ca67238834c5c14912cbf5b2111c6 (HEAD -> master, tag: tsprint1, origin/master, origin/HEAD)
Author: Ravi Reddy Manumachu <ravi.manumachu@ucd.ie>
Date:   Mon Jan 29 09:40:16 2024 +0000

    Showcasing git commit usage.

diff --git a/README2 b/README2
new file mode 100644
index 0000000..bc8660a
--- /dev/null
+++ b/README2
@@ -0,0 +1 @@
+New File
```

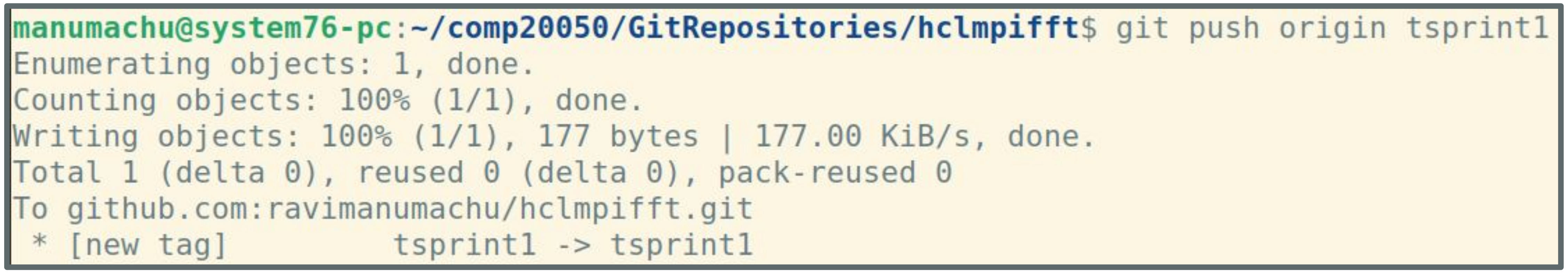
- Create an annotated tag using the command **git tag** as follows:

git tag -a tsprint1 -m "Sprint 1 Submission"

- You can see the tag data using the **git show** command.

git show tsprint1

Sharing Tags



```
manumachu@system76-pc:~/comp20050/GitRepositories/hclmpifft$ git push origin tsprint1
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 177 bytes | 177.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:ravimanumachu/hclmpifft.git
 * [new tag]         tsprint1 -> tsprint1
```

- By default, the **git push** command doesn't transfer tags to remote servers.

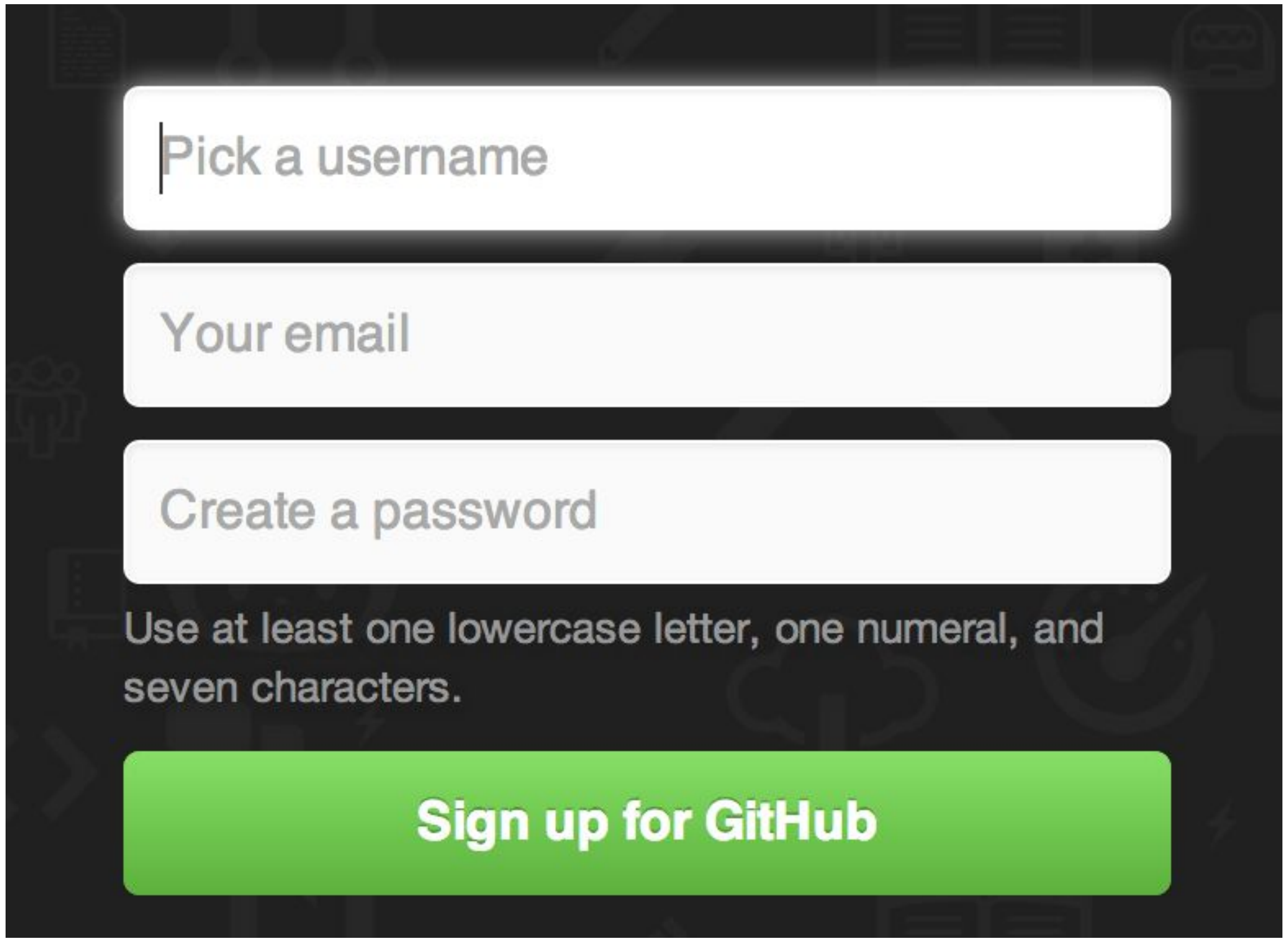
- You must explicitly push tags to GitHub repository using the command:

git push origin <tagname>

GitHub

GitHub

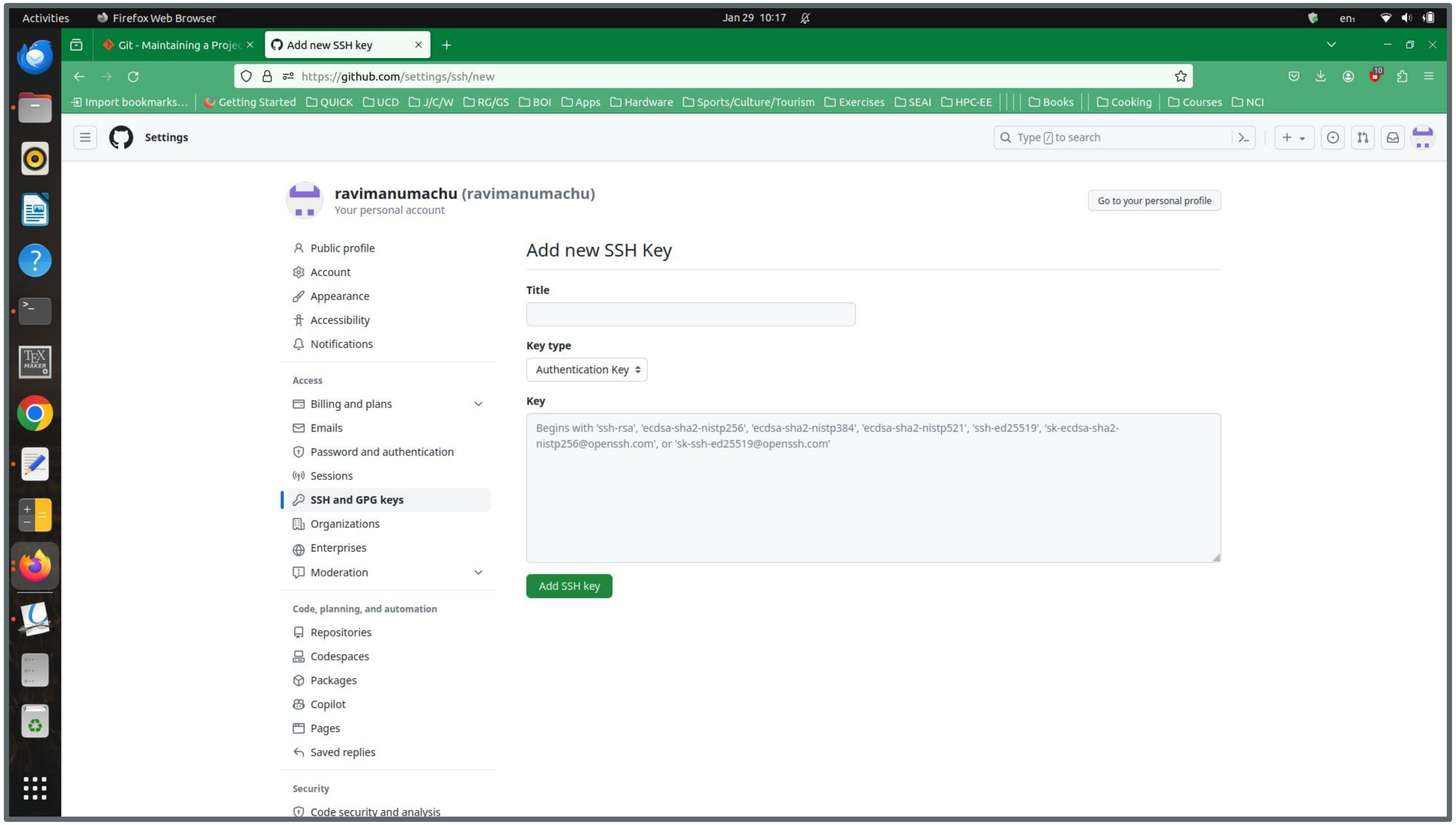
- GitHub and GitLab are hosts for thousands of Git repositories.
- We will use **GitHub** in this module.
- First, setup your account on GitHub.



Use at least one lowercase letter, one numeral, and seven characters.

SSH Access

- Add your SSH key. Covered in previous slides.



Add new SSH Key

Title

Key type

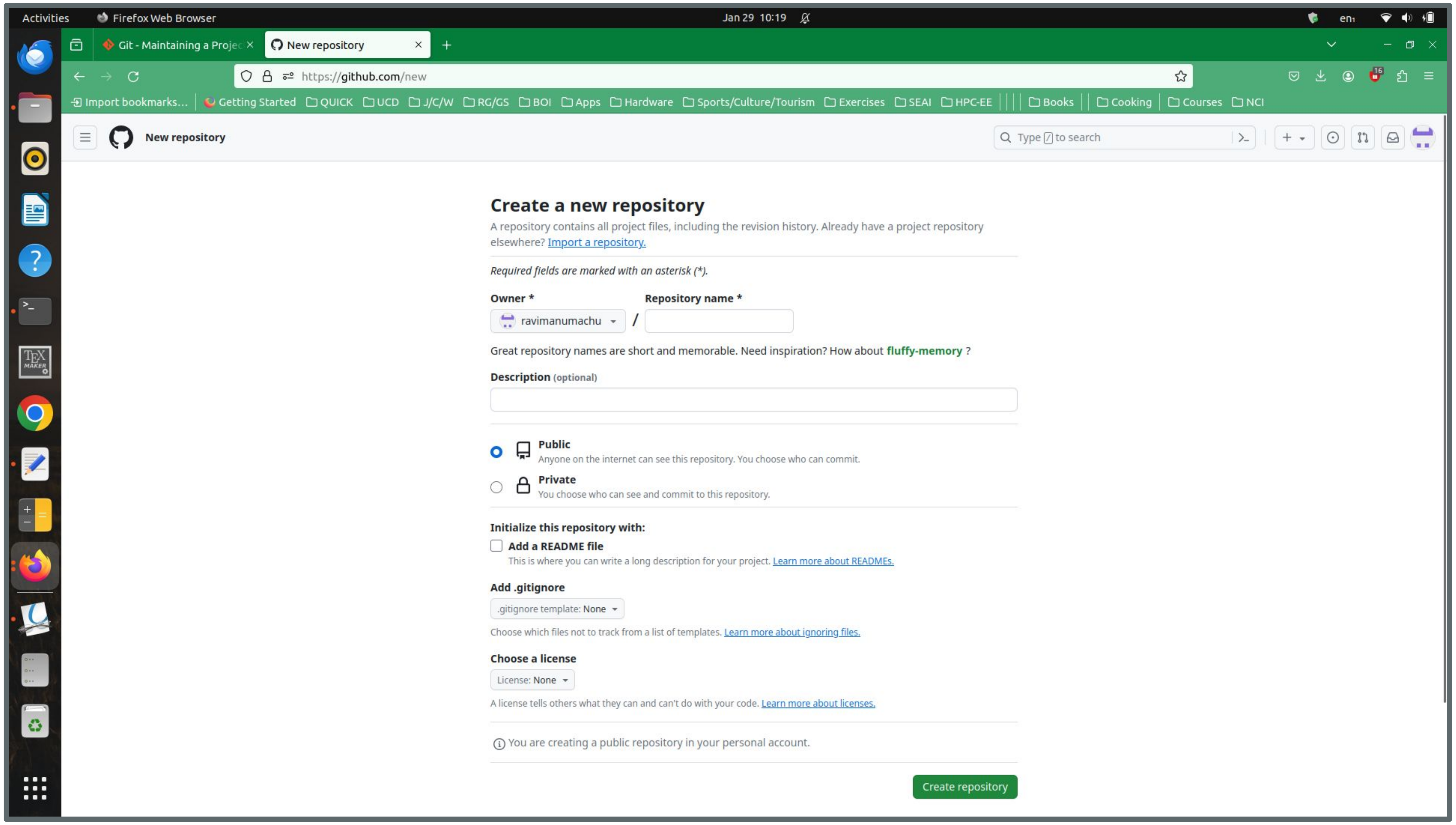
Authentication Key

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Create a New Repository

- Create a new repository to share our project code (https://github.com/new).



Create a new repository

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

Required fields are marked with an asterisk (*).

Owner * ravimanumachu / **Repository name ***

Great repository names are short and memorable. Need inspiration? How about **fluffy-memory** ?

Description (optional)

[✓ **Public**] Anyone on the internet can see this repository. You choose who can commit.

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

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. Learn more about READMEs.

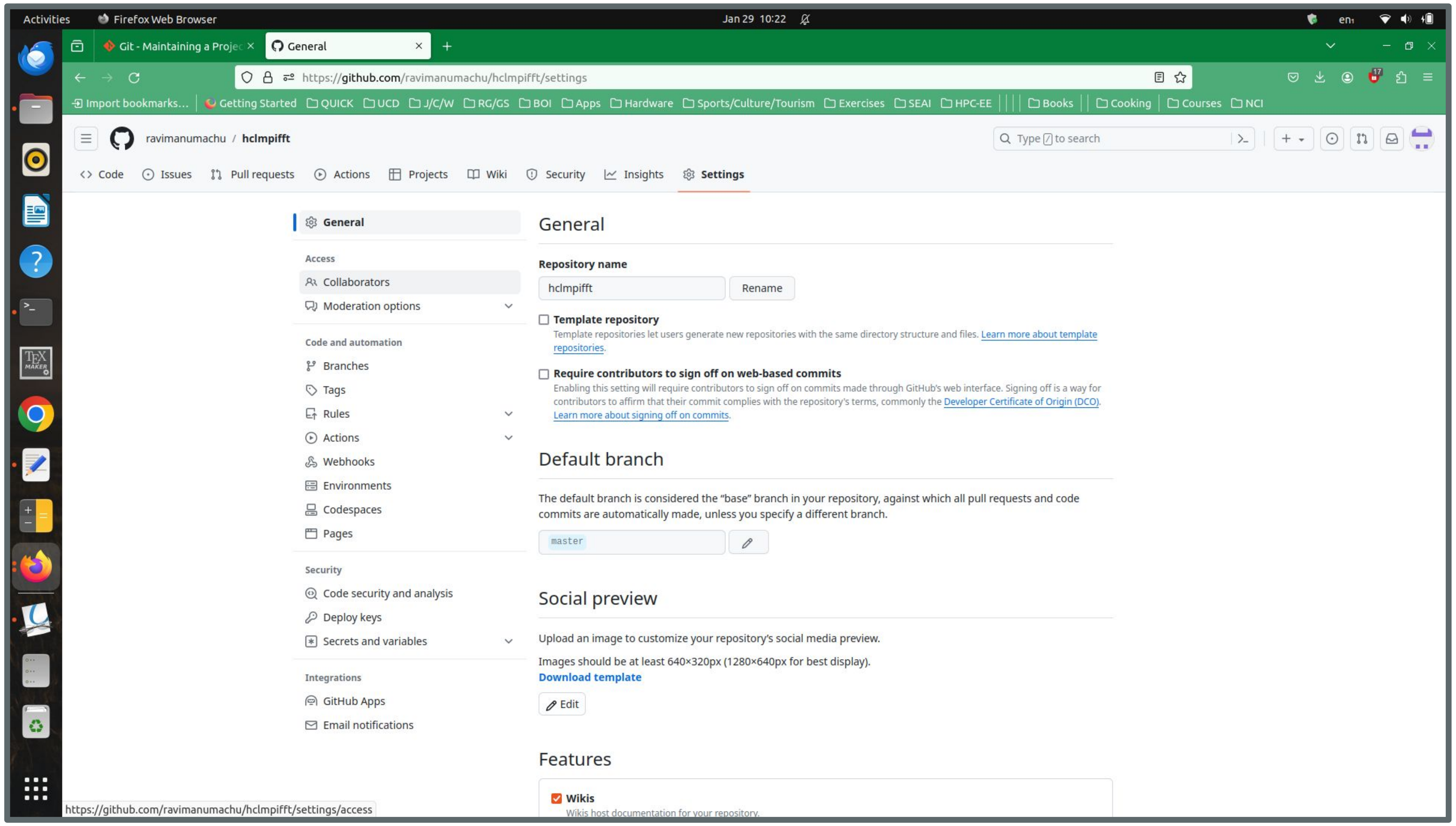
Add .gitignore
.gitignore template: None
Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license
License: None
A license tells others what they can and can't do with your code. Learn more about licenses.

ⓘ You are creating a public repository in your personal account.

Add Collaborators

- Add your group members to the repository using the **Collaborators** option on the left menu.
- **Collaborators** will have push access, which means they have both read and write access to the Git repository.



General

Repository name

hclmpifft | Rename

☐ **Template repository**
Template repositories let users generate new repositories with the same directory structure and files. Learn more about template repositories.

☐ **Require contributors to sign off on web-based commits**
Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO). Learn more about signing off on commits.

Default branch

The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

`master`

Social preview

Upload an image to customize your repository's social media preview.

Images should be at least 640×320px (1280×640px for best display).
Download template

Edit

Features

✓ **Wikis**
Wikis host documentation for your repository.

Q&A

To follow…

Software Architectural Design