GITHUB TUTORIAL

*Prepared by Nupur Thakur(Jun-Nov 2019)*

# Introduction

GitHub is a social code-hosting platform, i.e. It is a software highly used for version-control . In simple terms GitHub is a cloud for codes.

# Git v/s GitHub

Git is a version control system that lets you manage and keep track of your source code history. GitHub is a cloud-based hosting service that lets you manage Git repositories. If you have open-source projects that use Git, then GitHub is designed to help you better manage them.

# GitHub Terminologies

1. **Repository (or Repo) :** You can think of it as a folder or storage space where your project files can be stored.
2. **Fork:** Fork can also be termed as copy.E.g. Let’s consider you found some issues in a file of a project but you can’t directly make changes in the main files. So you’ll have to make a copy of it. This is what forking means.
3. **Commit:** Commit is like ‘saving’ an updated file to its original folder and overwrites an older version
4. **Clone:**Clone refers to coping of a repository from GitHub to the local machine.
5. **Branch:** A branch is a parallel version of a repository. I.e. Sometimes different contributors create their own branch so that changes made by other developers don’t reflect in their project.
6. **Fetch:**Fetching refers to getting the latest changes from an online repository without merging them in your local branch. Once these changes are fetched you can compare them to your local branches (the code residing on your local machine).
7. **Issue:**Issues are suggested improvements, tasks or questions related to the repository.Issues can be created by anyone (for public repositories), and are moderated by repository collaborators. Each issue contains its own discussion forum, can be labeled and assigned to a user.
8. **Merge:** Merging takes the changes from one branch (in the same repository or from a fork), and applies them into another.A merge can be done automatically via a Pull Request via the GitHub.com web interface if there are no conflicting changes, or can always be done via the command line.
9. **Pull Request:** In simplest form, pull requests are a way by which developers notify their team members that they have made some changes and asking your permission to add your changes to the main file.
10. **Version Control:** Basically, When you have a Microsoft Word file, you either overwrite every saved file with a new save, or you save multiple versions. With Git, you don’t have to. It keeps “snapshots” of every point in time in the project’s history, so you can never lose or overwrite it.

# Git-specific Commands :

Since Git was designed with a big project like Linux in mind, there are a lot of Git commands. However, to use the basics of Git, you’ll only need to know a few terms. They all begin the same way, with the word “git.”

**git init**: Initializes a new Git repository. Until you run this command inside a repository or directory, it’s just a regular folder. Only after you input this does it accept further Git commands.

**git config**: Short for “configure,” this is most useful when you’re setting up Git for the first time.

**git help:** Forgot a command? Type this into the command line to bring up the 21 most common git commands. You can also be more specific and type “git help init” or another term to figure out how to use and configure a specific git command.

**git status:** Check the status of your repository. See which files are inside it, which changes still need to be committed, and which branch of the repository you’re currently working on.

**git add:** This does not add new files to your repository. Instead, it brings new files to Git’s attention. After you add files, they’re included in Git’s “snapshots” of the repository.

**git commit:** Git’s most important command. After you make any sort of change, you input this in order to take a “snapshot” of the repository. Usually it goes git commit -m “Message here.” The -m indicates that the following section of the command should be read as a message.

**git branch:** Working with multiple collaborators and want to make changes on your own? This command will let you build a new branch, or timeline of commits, of changes and file additions that are completely your own. Your title goes after the command. If you wanted a new branch called “cats,” you’d type git branch cats.

**git checkout:** Literally allows you to “check out” a repository that you are not currently inside. This is a navigational command that lets you move to the repository you want to check. You can use this command as git checkout master to look at the master branch, or git checkout cats to look at another branch.

**git merge**: When you’re done working on a branch, you can merge your changes back to the master branch, which is visible to all collaborators. git merge cats would take all the changes you made to the “cats” branch and add them to the master.

**git push**: If you’re working on your local computer, and want your commits to be visible online on GitHub as well, you “push” the changes up to GitHub with this command.

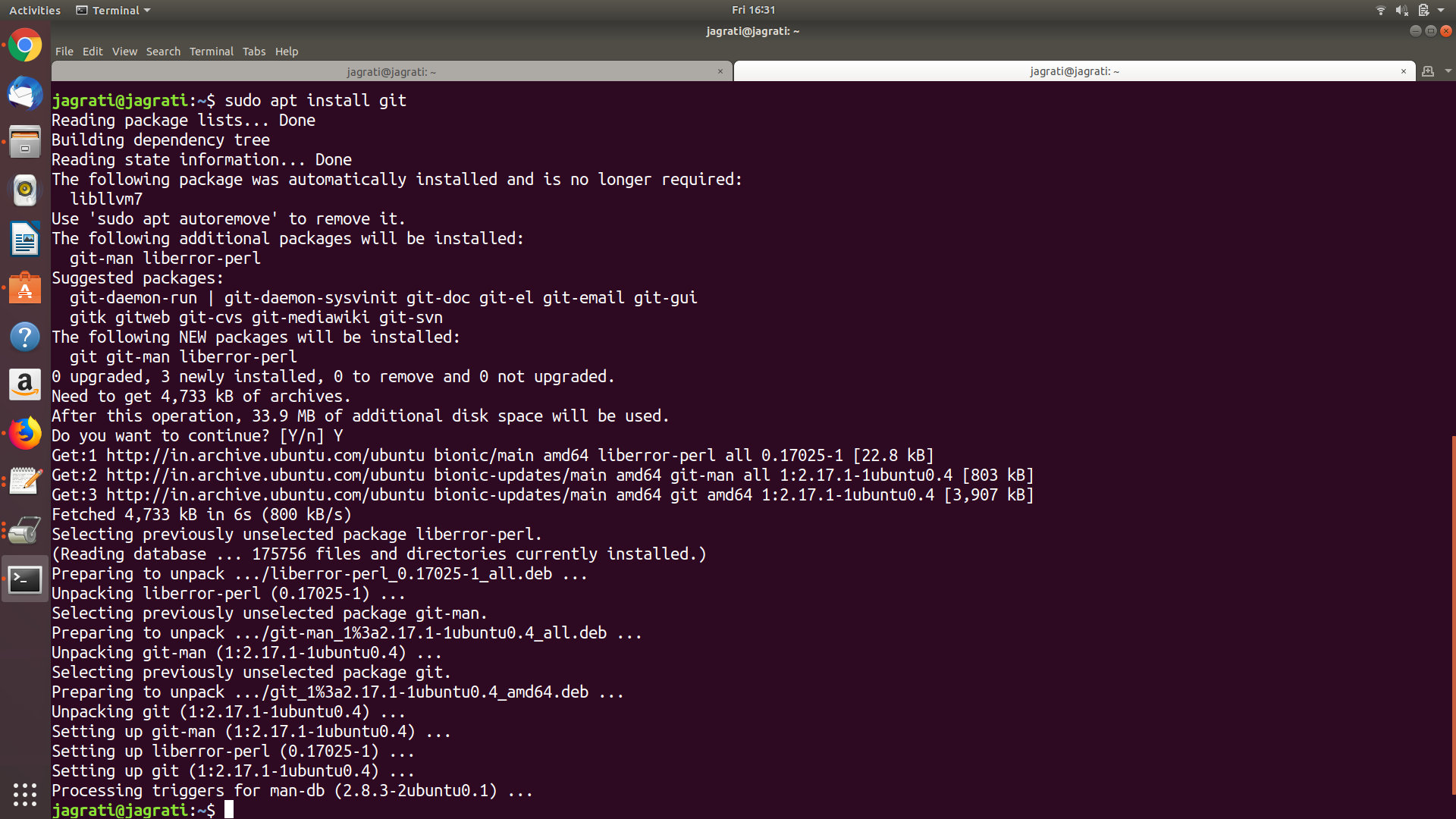
**git pull**: If you’re working on your local computer and want the most up-to-date version of your repository to work with, you “pull” the changes down from GitHub with this command.

# Getting started with Git

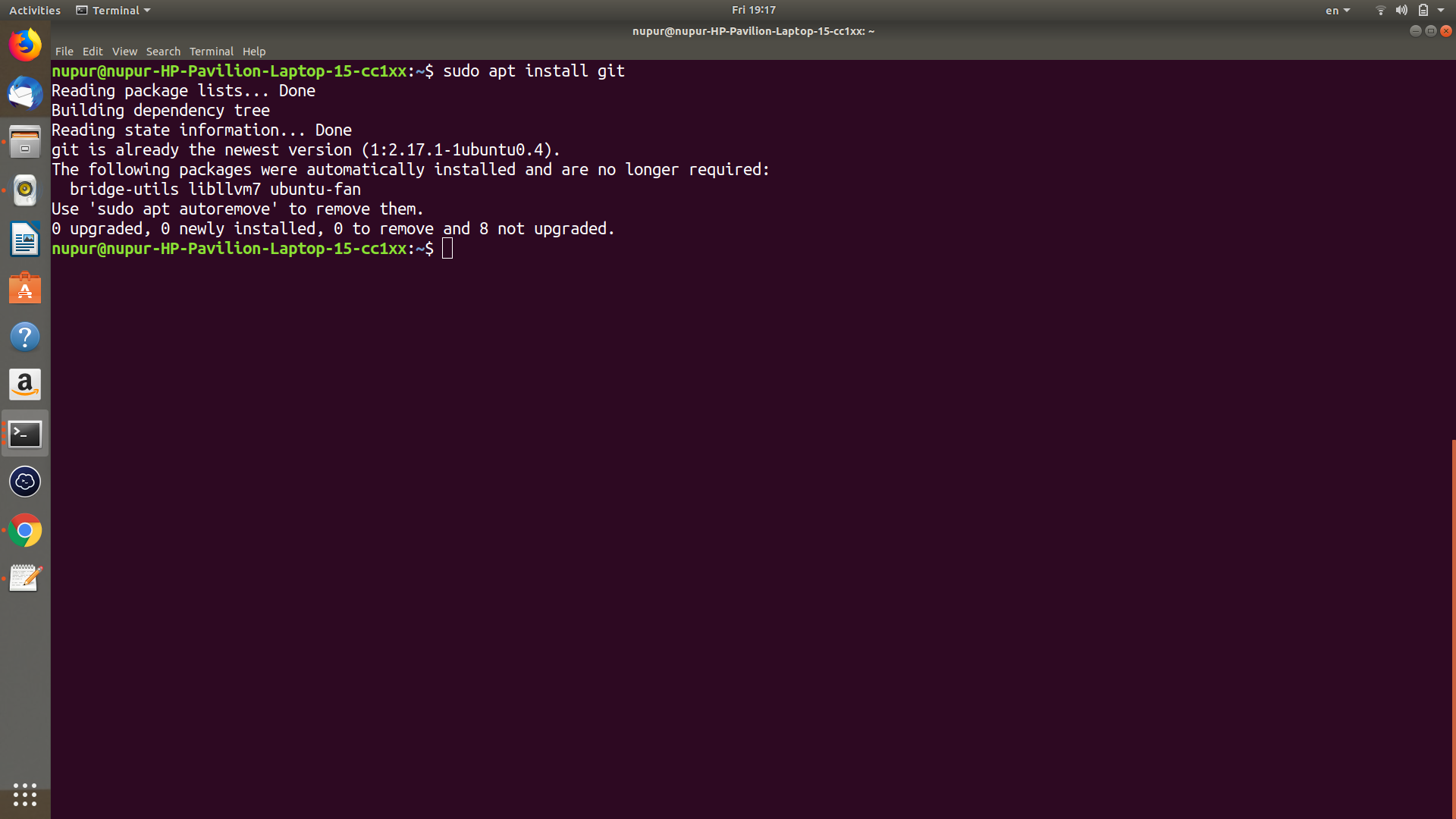
1. **Setup:**

IN LINUX : (ubuntu) sudo apt install git

**(installing git for the first time)**

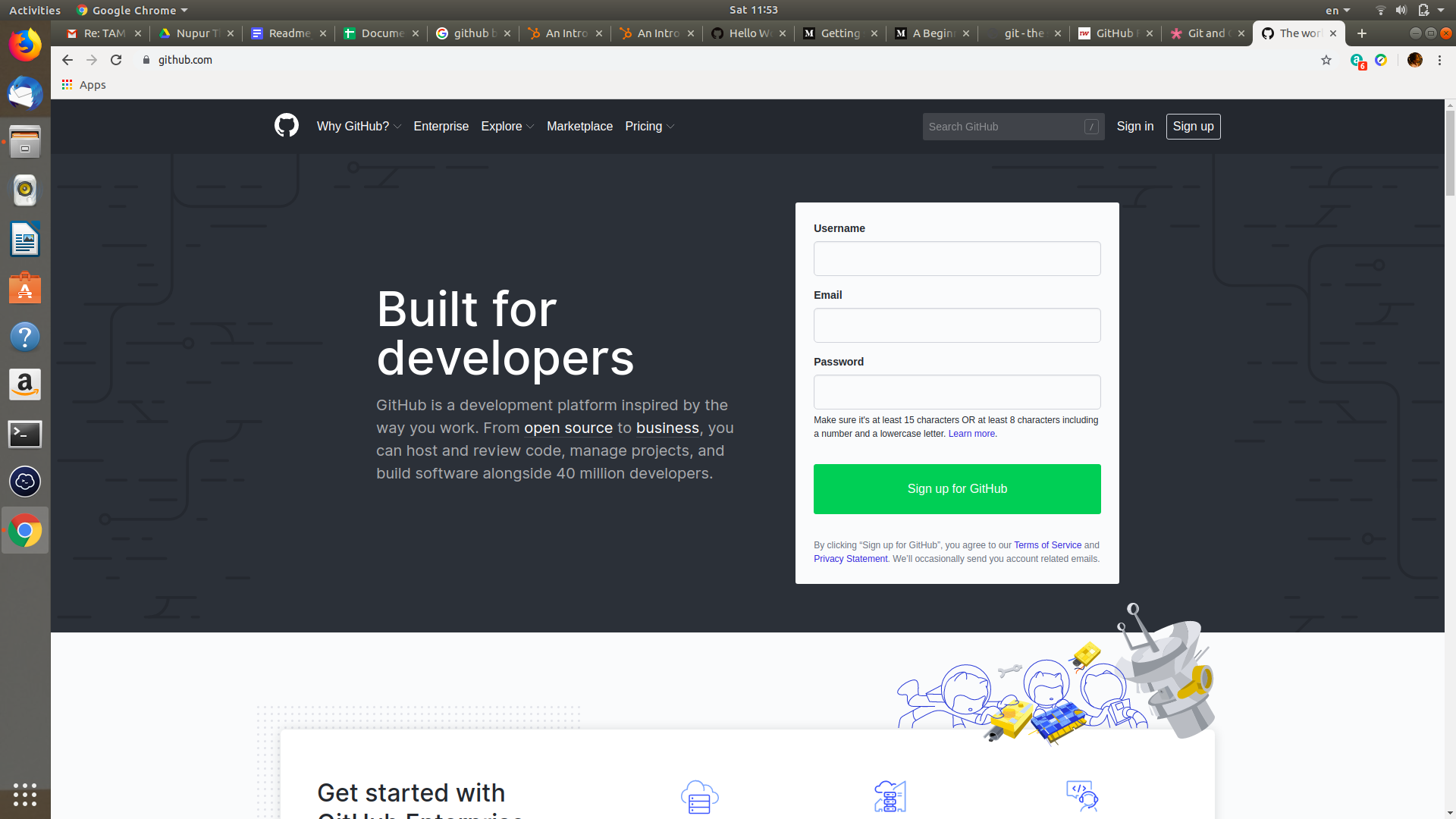


**(if already installed)**

****

1. **Using Git**

Register for your new Github account on github.com (or login if you have one already, and I’ll be with you in a second).

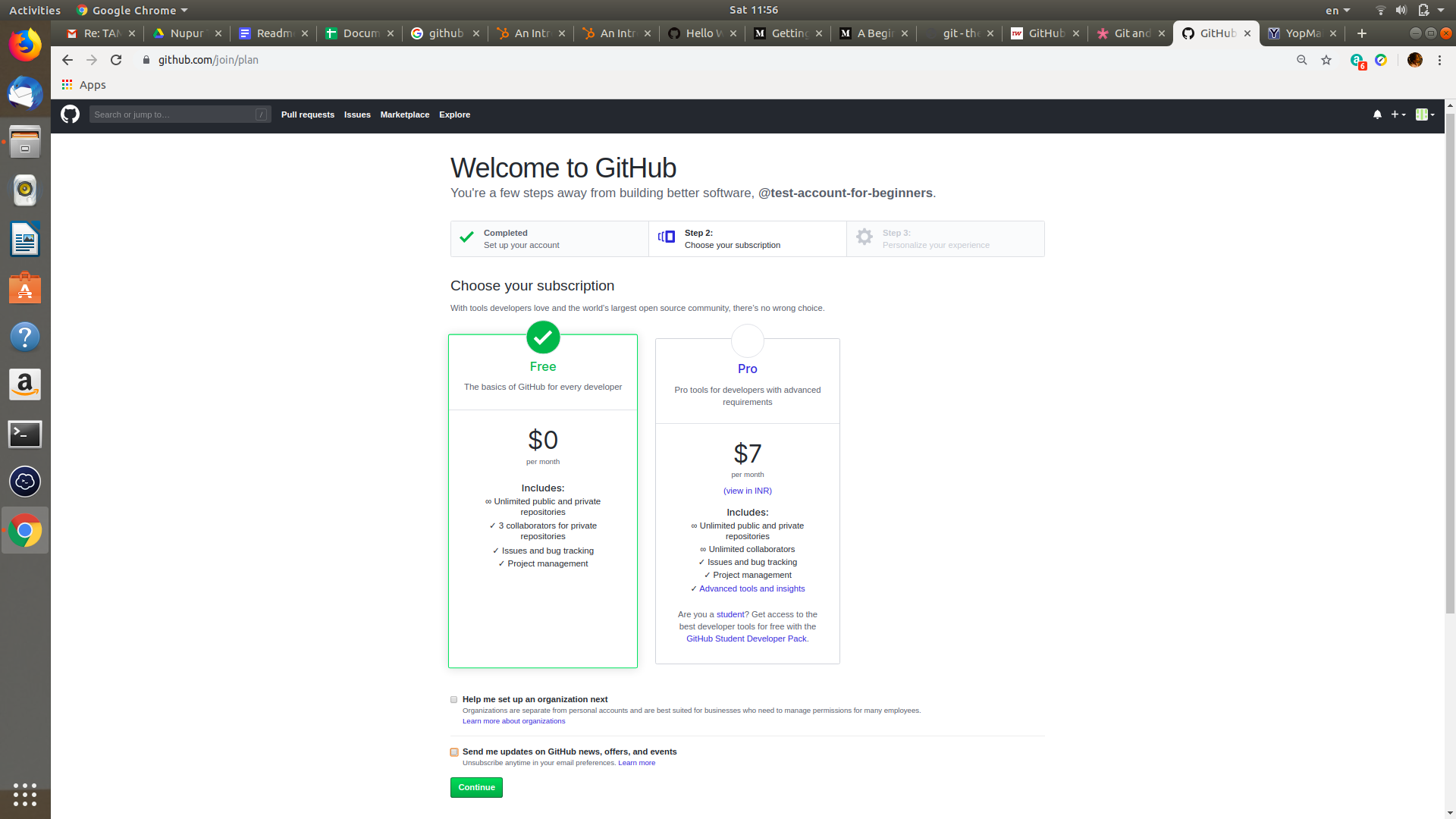


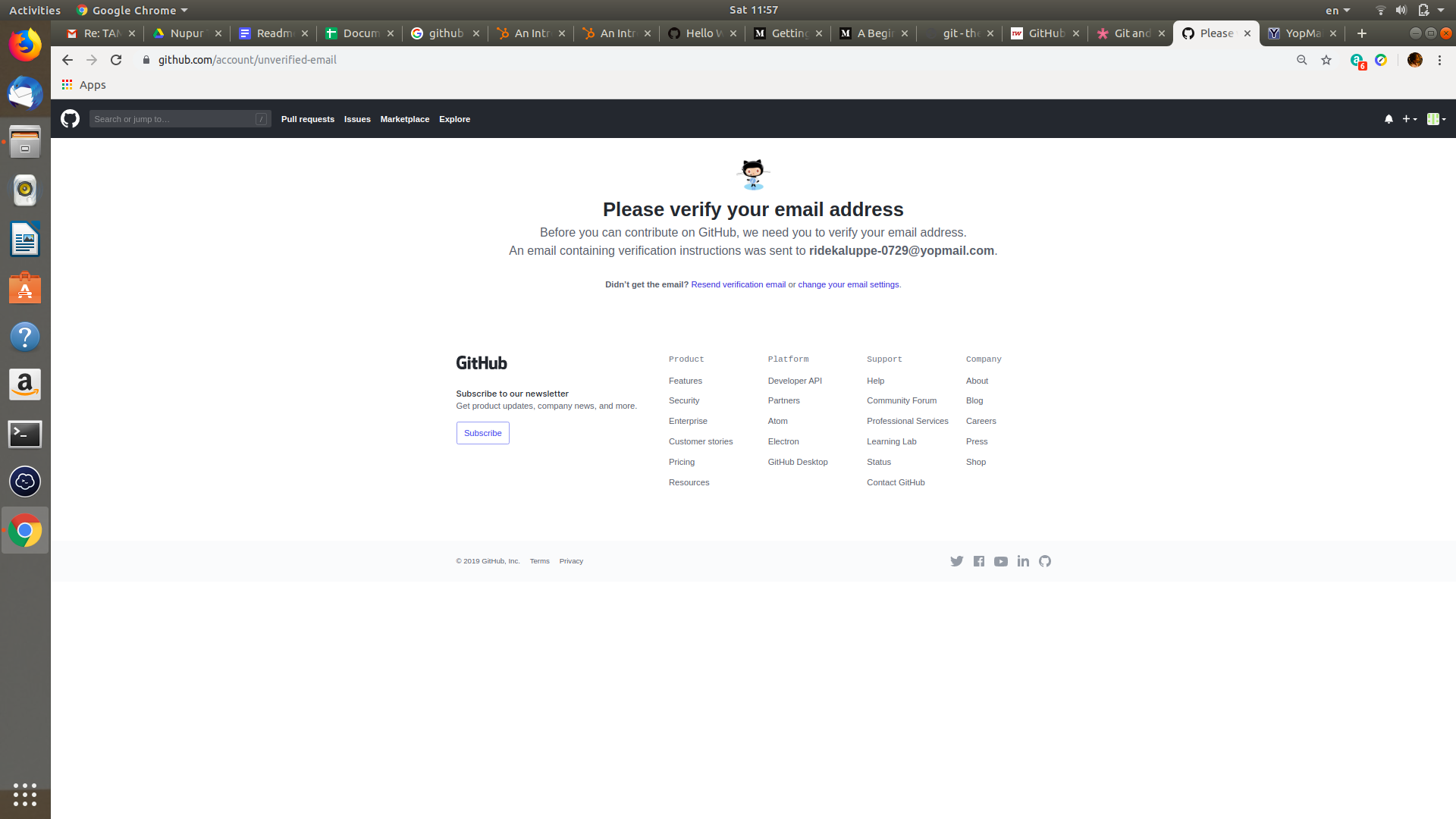
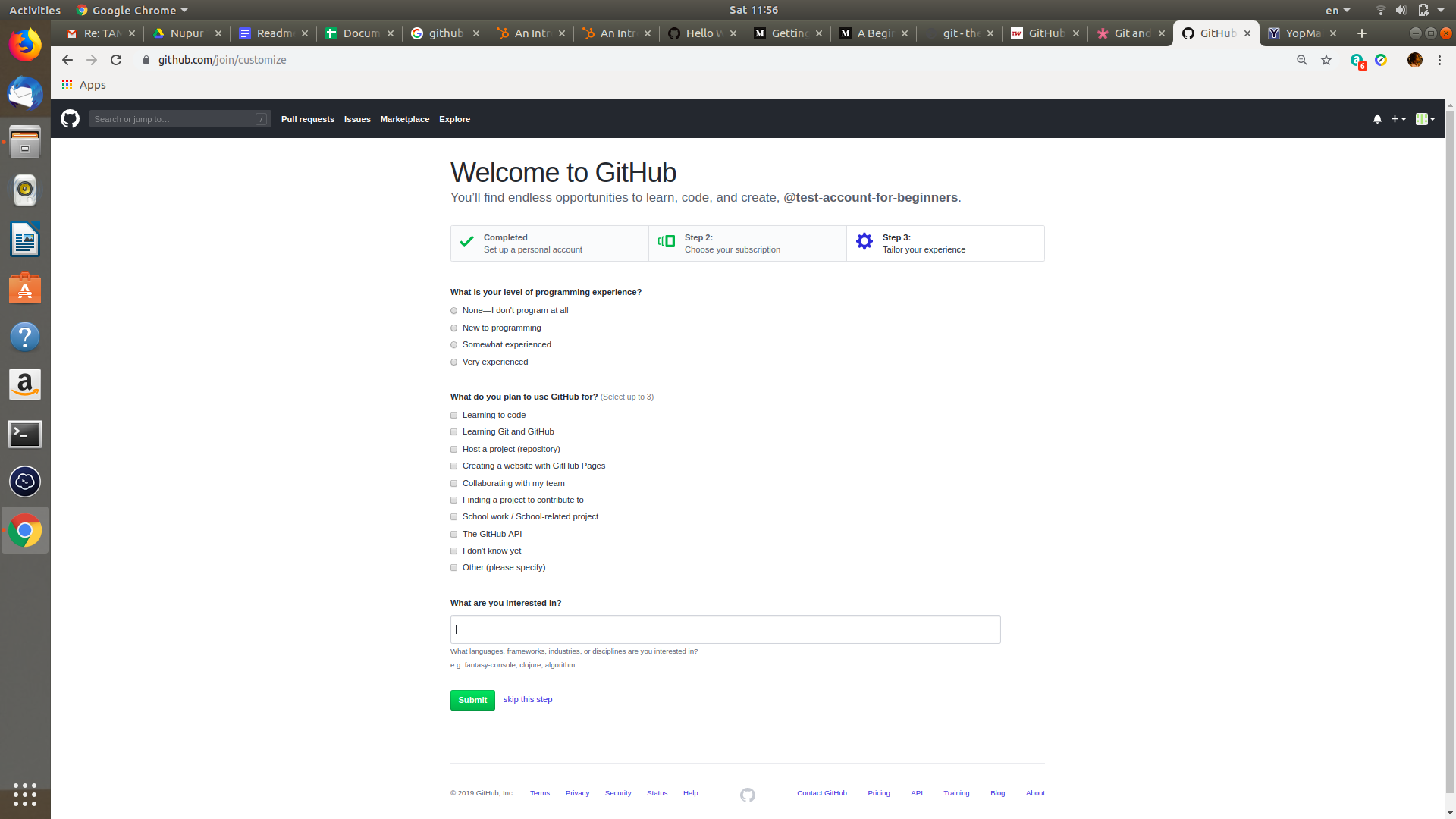
Enter your details :

Username,email address and password.

(if you already have an account, click on Sign in)



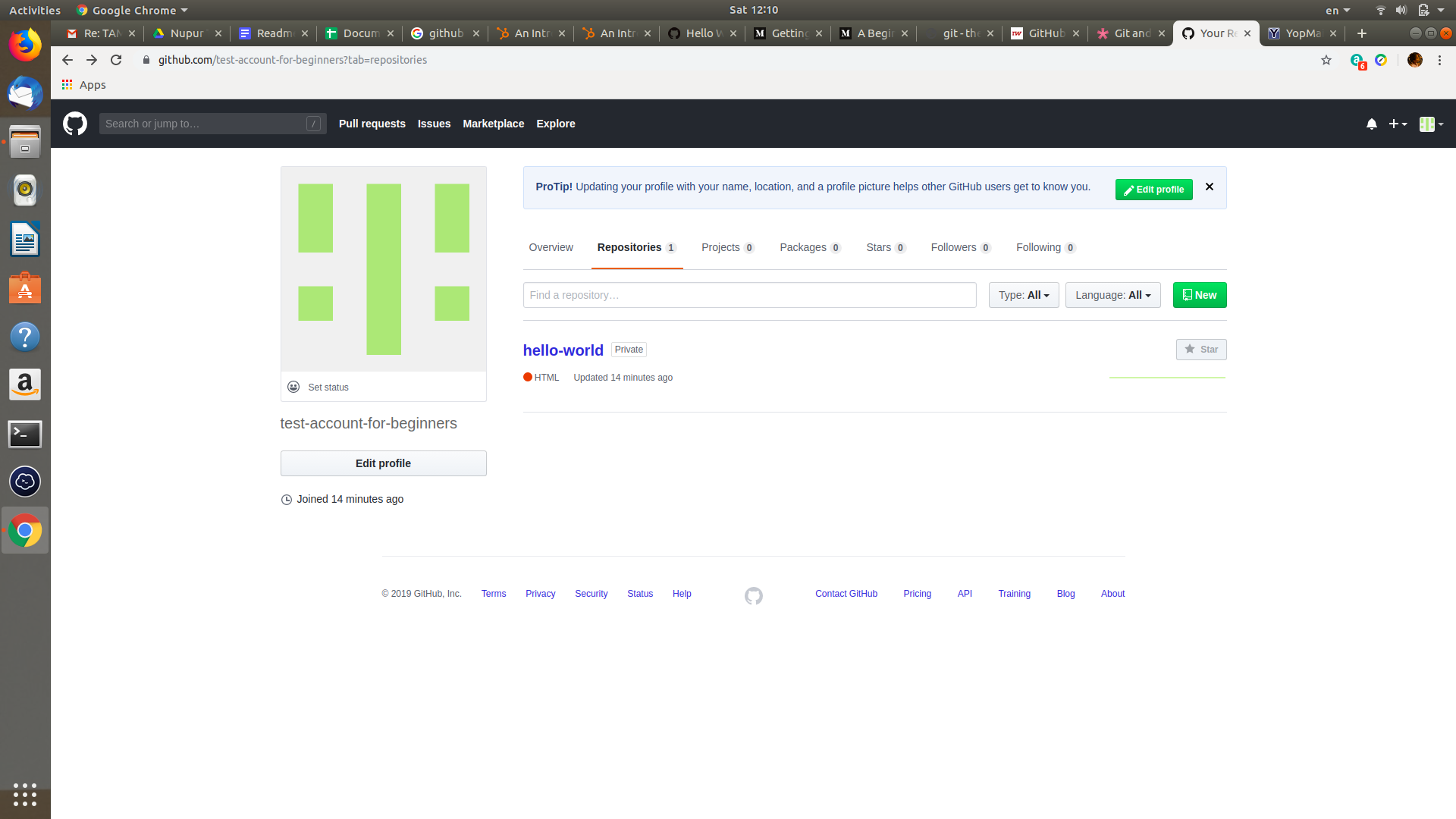




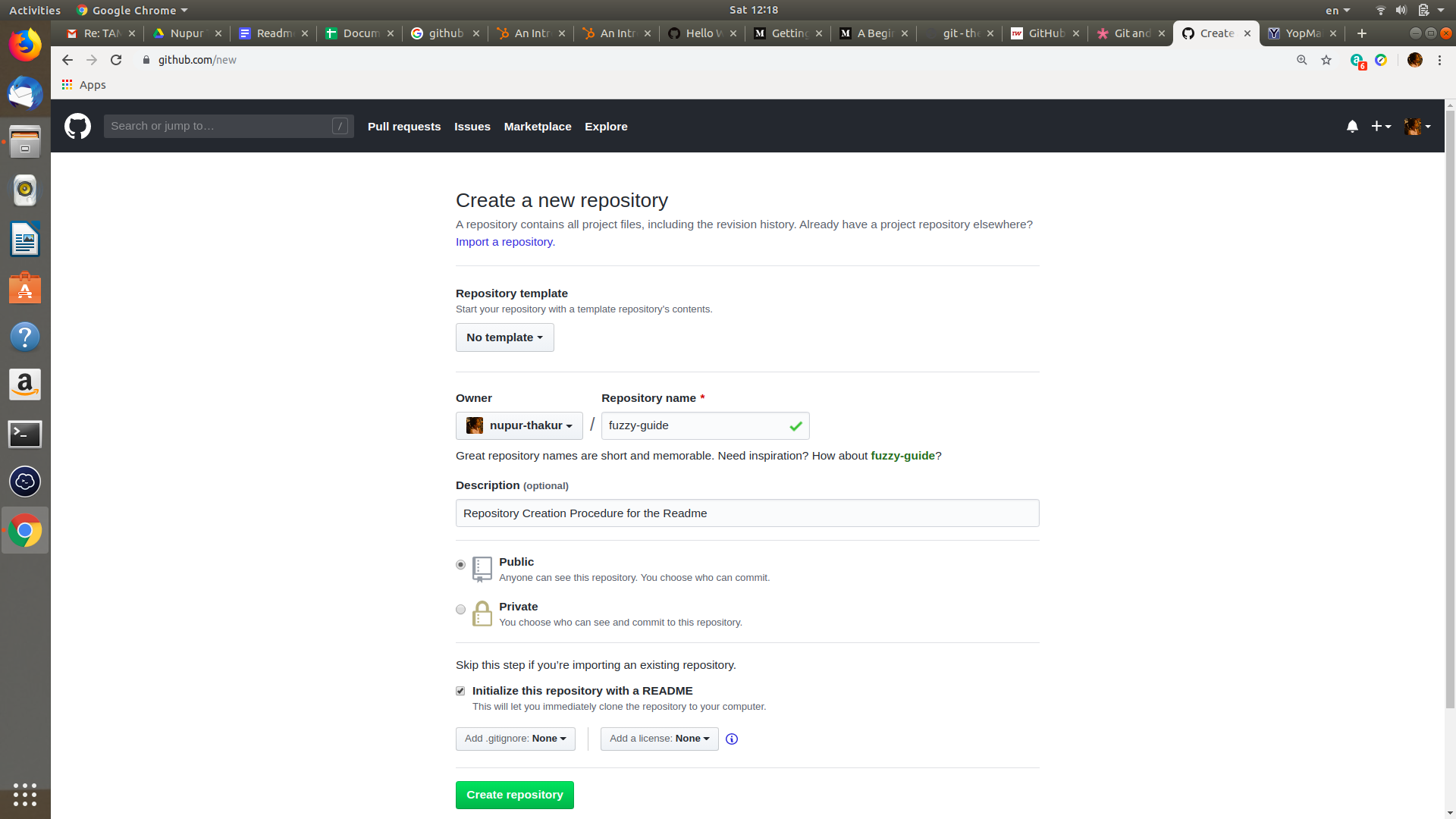
(after verifying your email, you are all set to use your GitHub account)

1. **Creating a repository**

When you create a new GitHub account, a repository named Hello-World is already created.

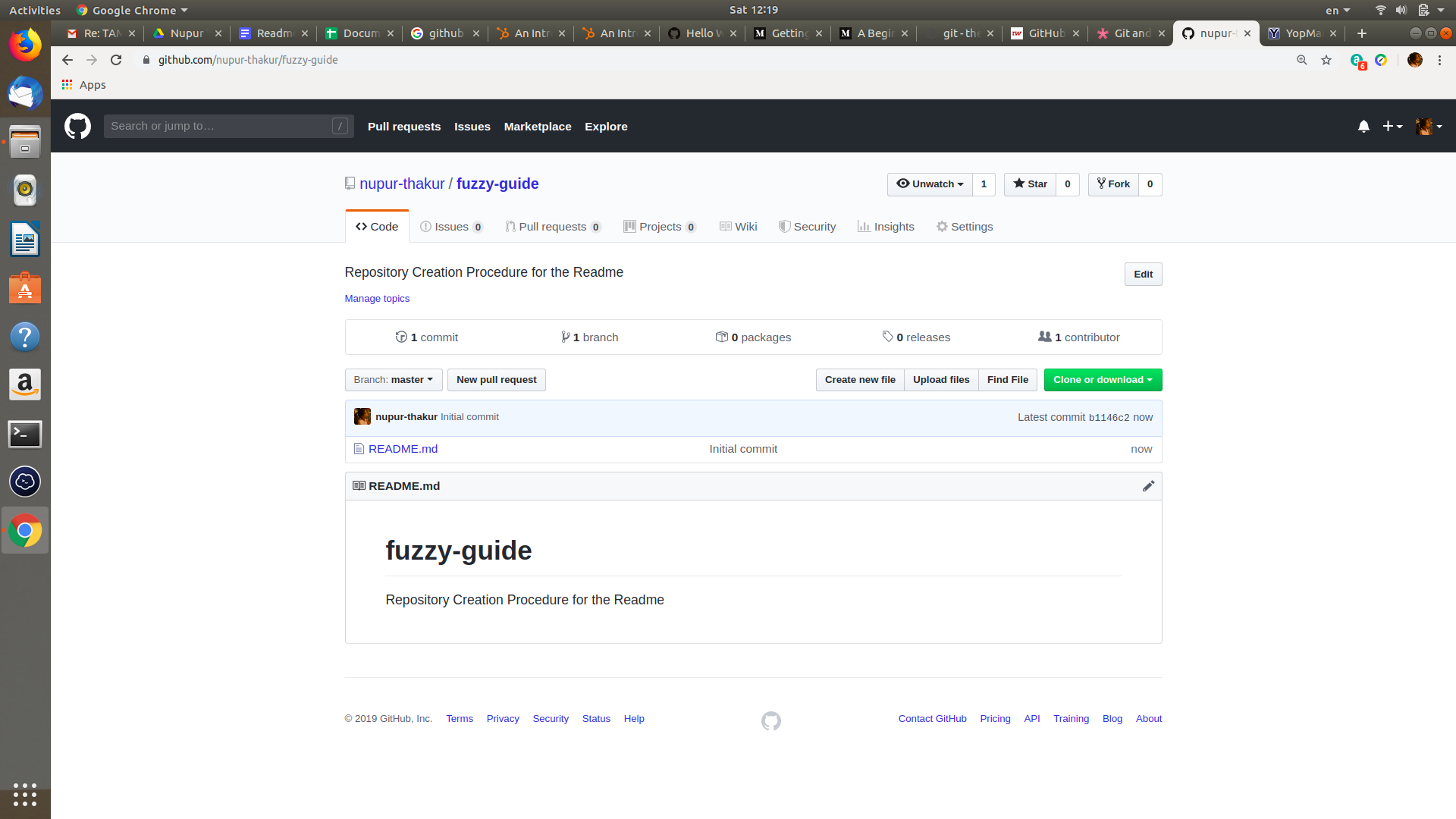


To create a new repository, click on the “new” button.



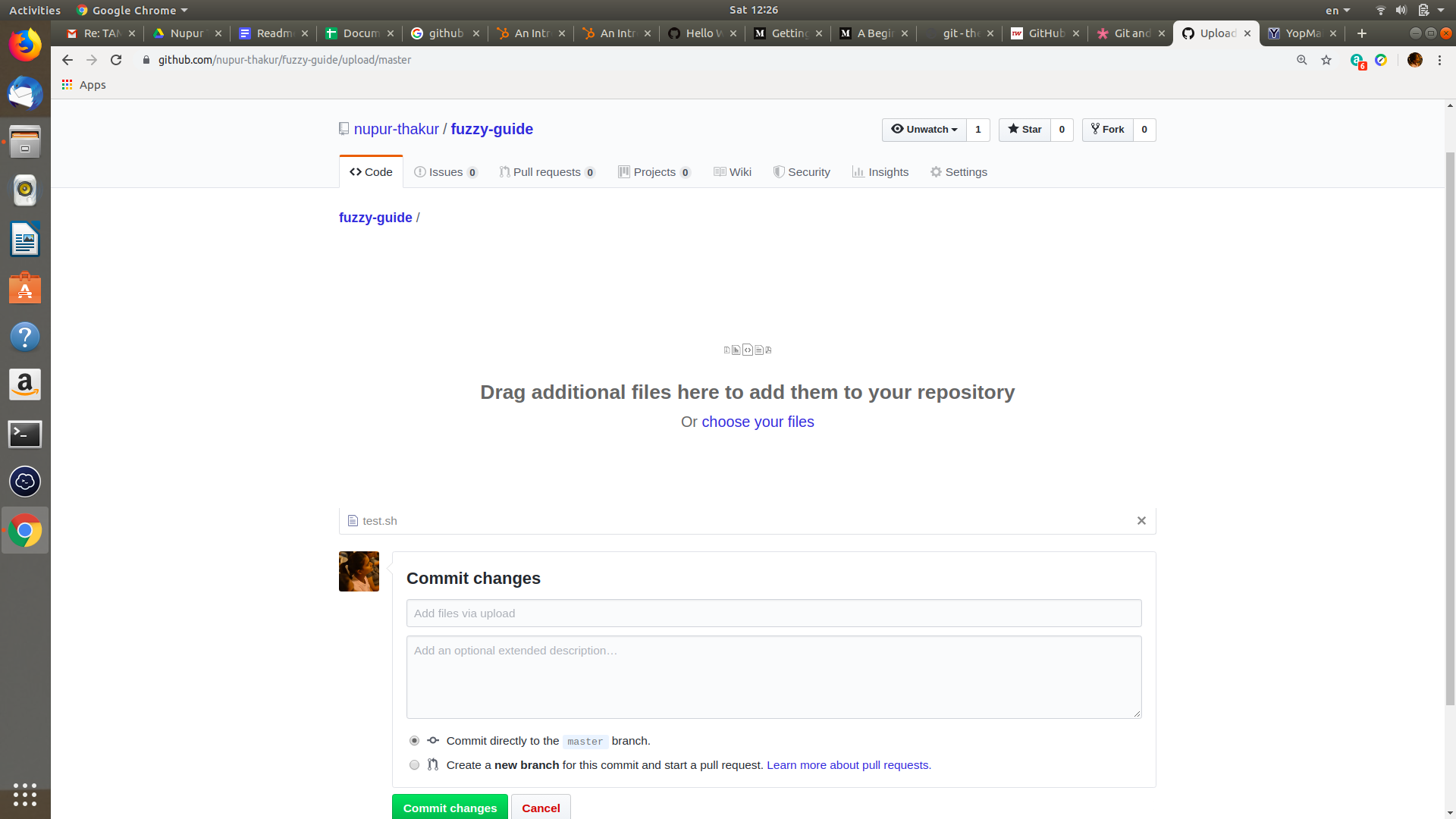
After clicking on “New” Button you will be redirected to this page, where you have to enter a valid repository name, write a description about the repo if you want to, Select whether you want your repo to be public or private and if you want to initialize your repo with a readme click on the check provided for it.

Finally click on “Create Repository” Button.

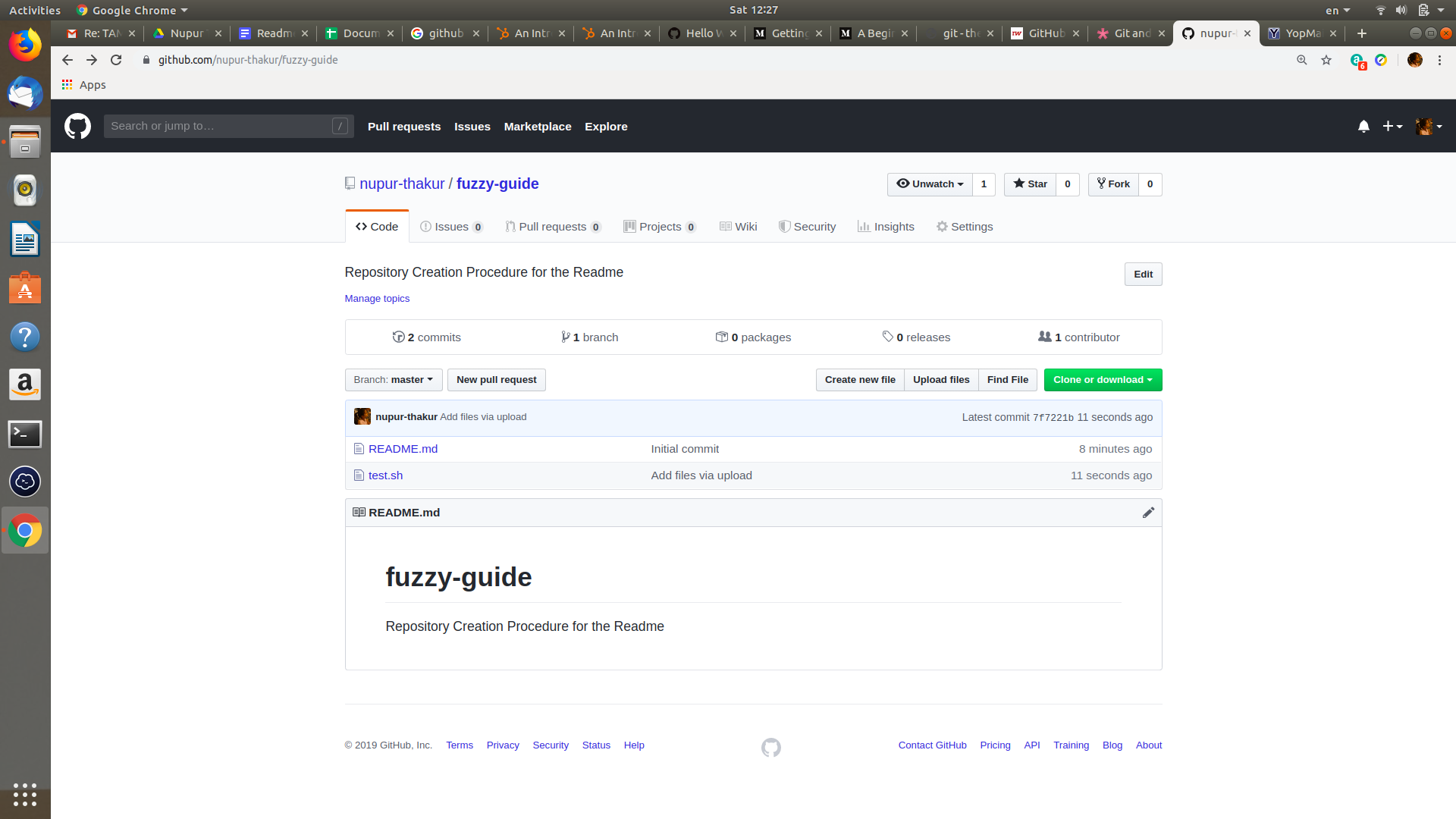


Now you can create a new file from here only or you can upload new file from your local machine. Adding a new file can be done graphically and from command line as well.

Uploading a new file from the GitHub website itself(will tell how to upload from command line in future).

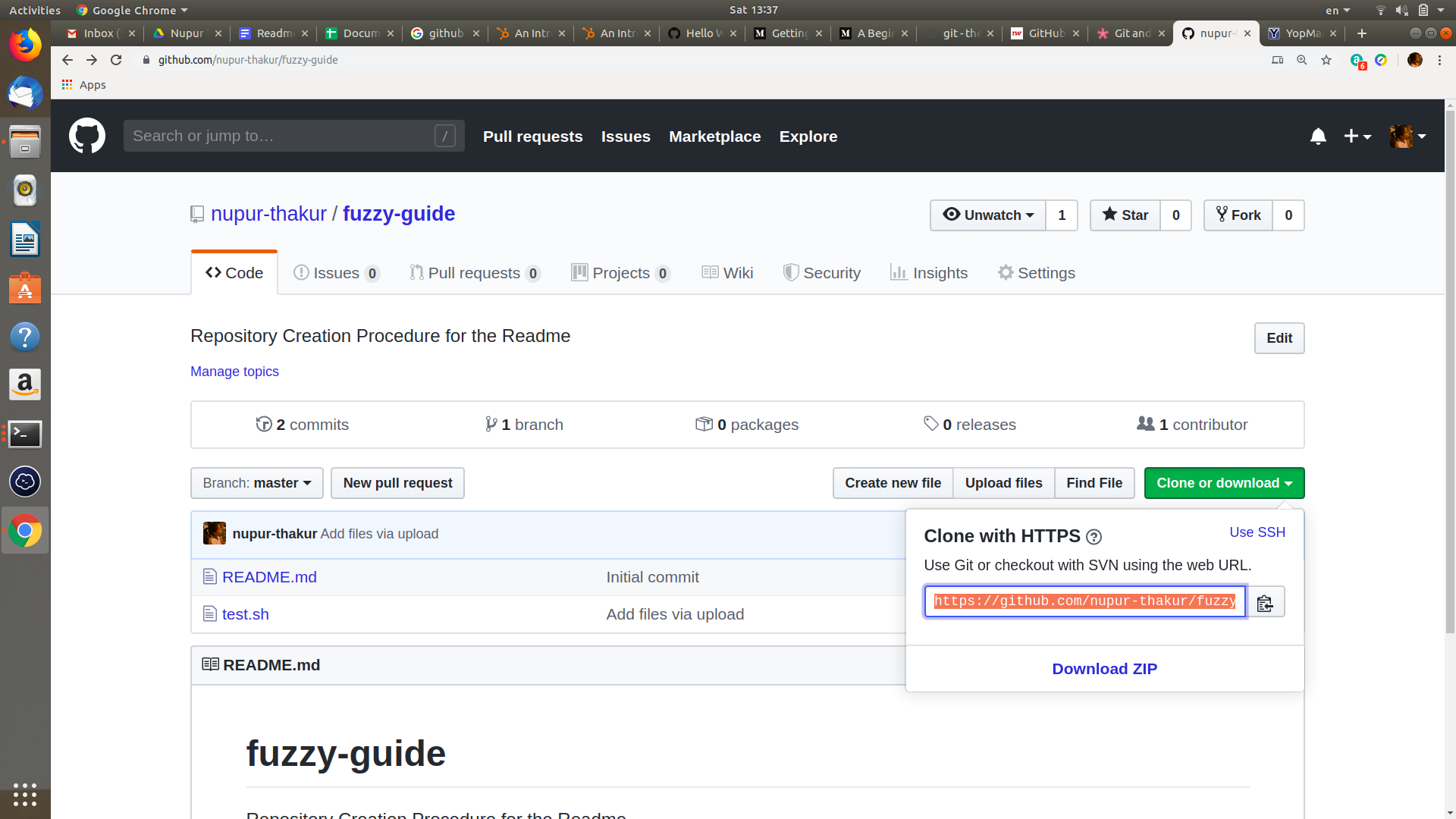


First choose the file from your local machine(e.g. Test.sh, in my case), Commit changes with a message either directly to master branch(If it’s your personal repository if you are contributing to some other repository then click on the option of create a new branch for this commit so that your changes will not conflict with the master branch)



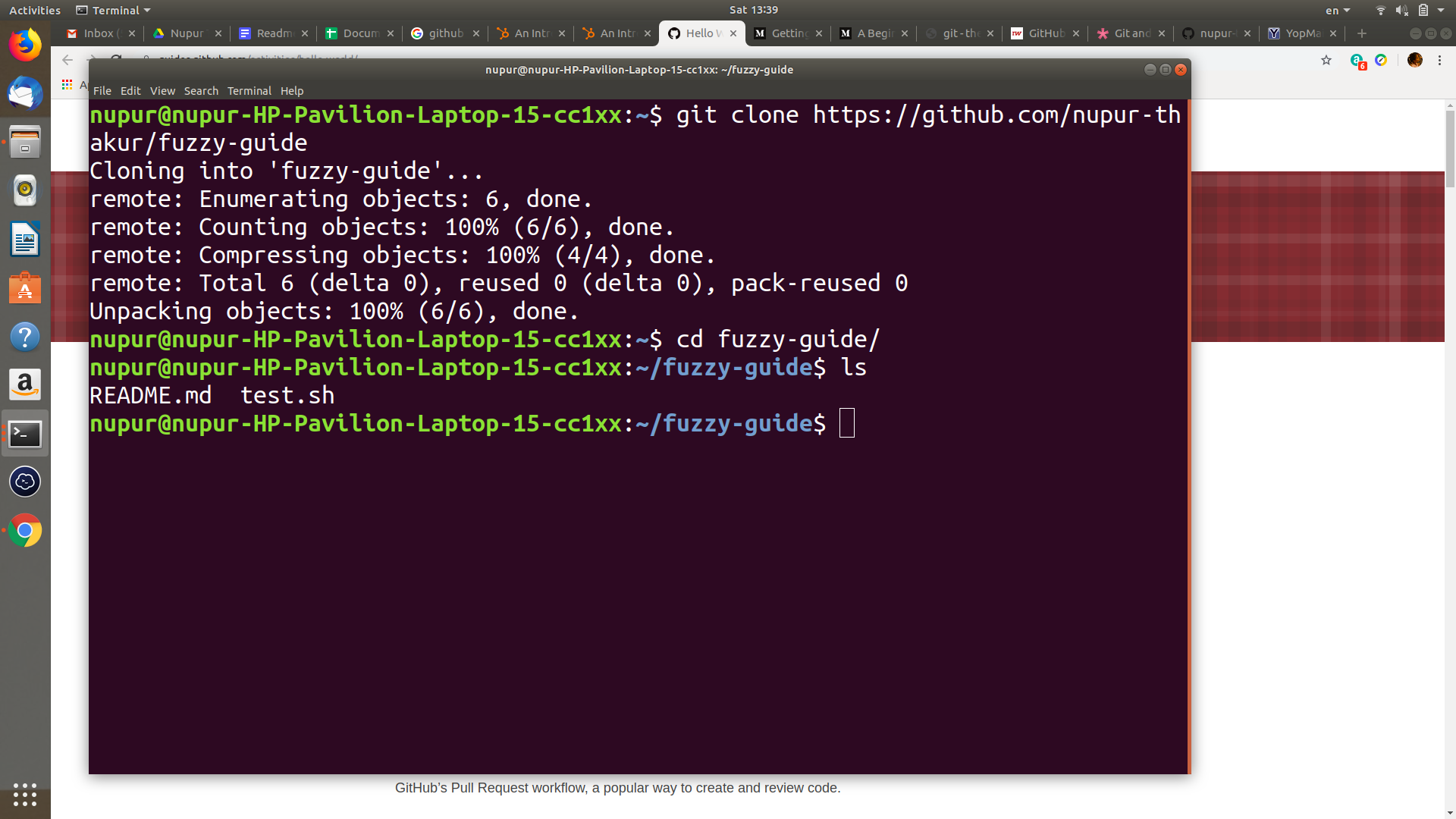
1. **Cloning a repository**

To clone a repository you can go to that repo on GitHub and click on Clone or Download.



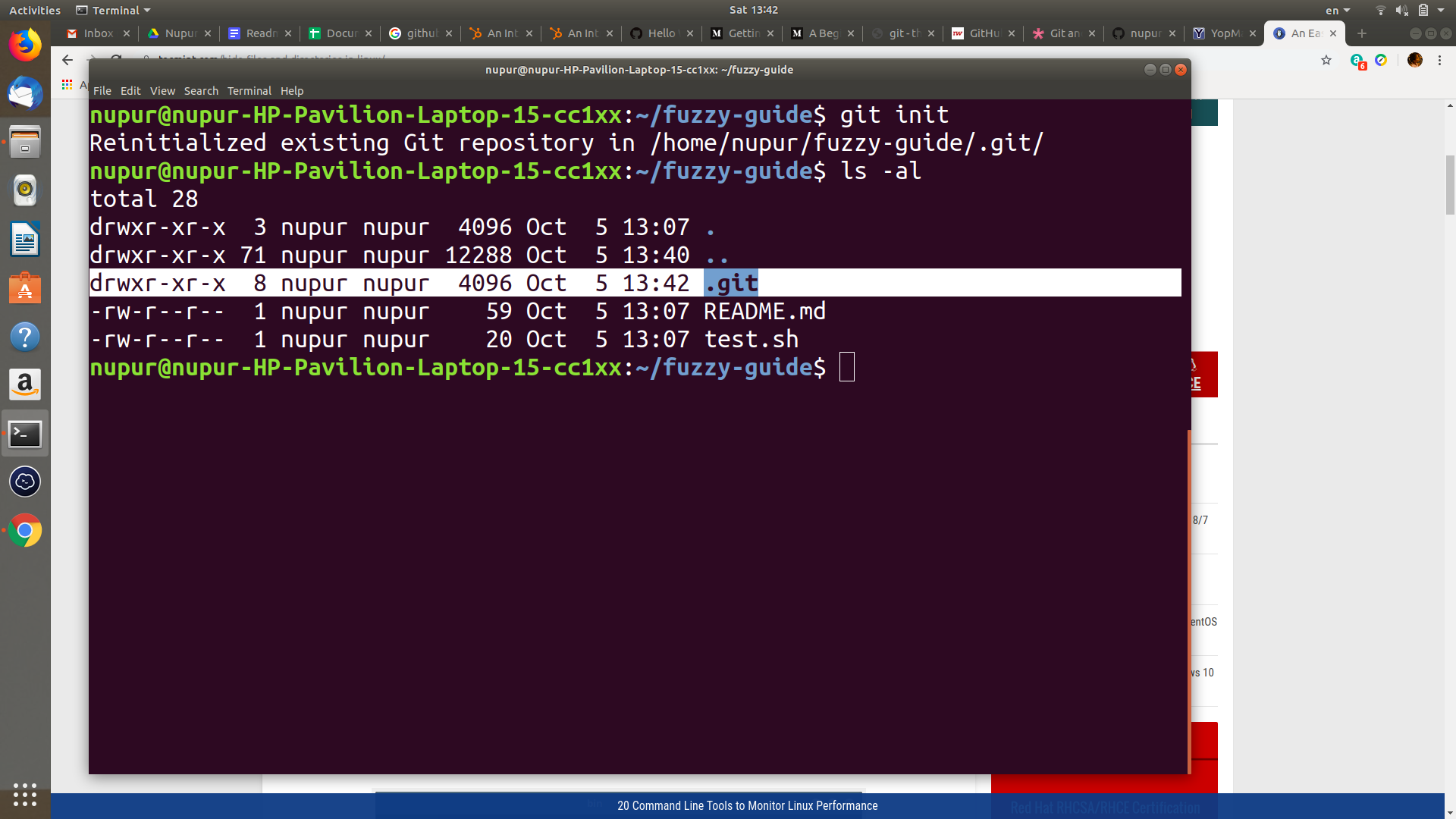
From command line, you can use the command :

**git clone <url-to-the-repo>**

****

1. **Initializing a repository**

Before you can work with Git, you have to initialize a project repository, setting it up so that Git will manage it. Open up your terminal, and in your project directory run the command git init as shown in the screenshot below.

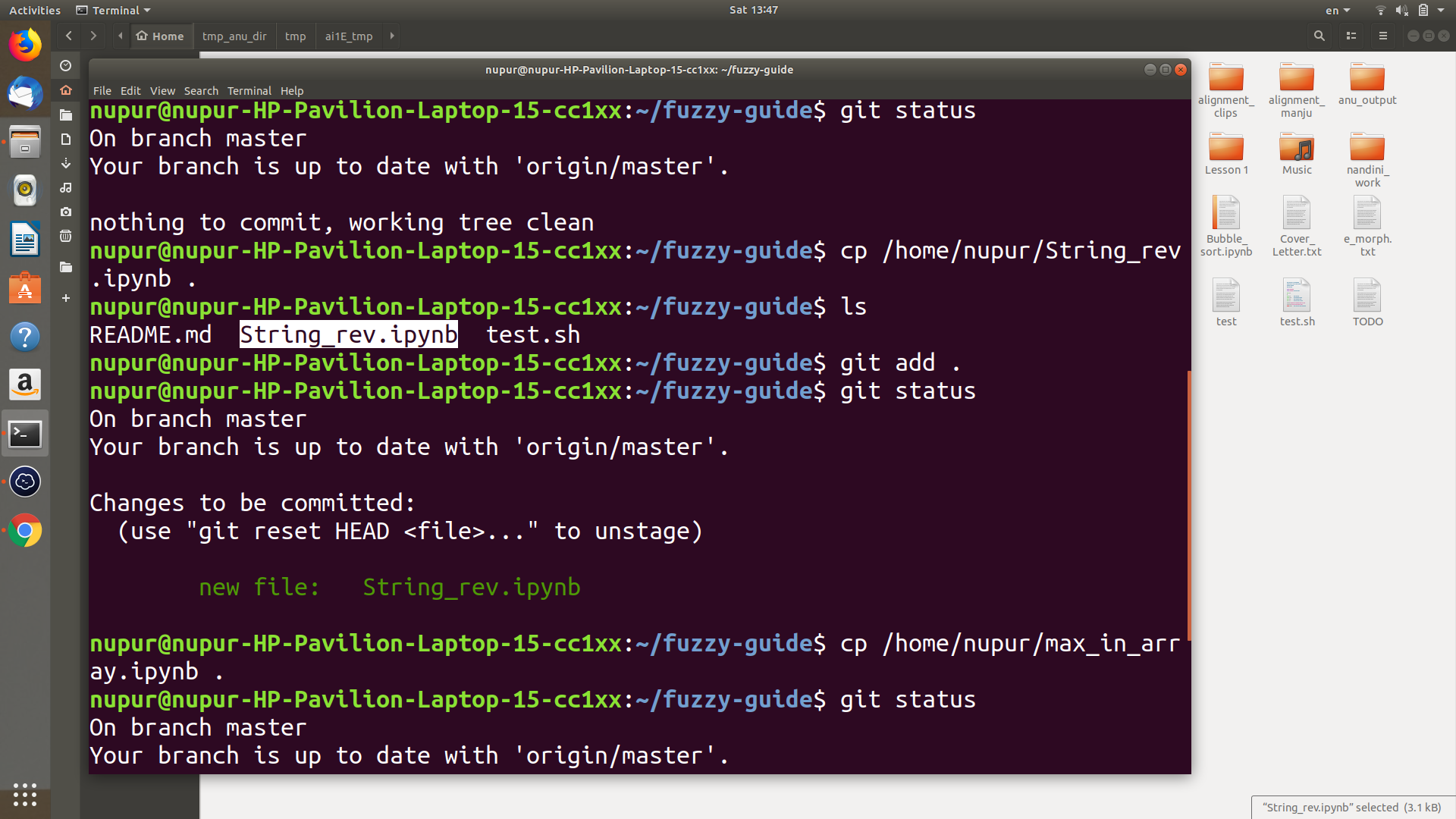


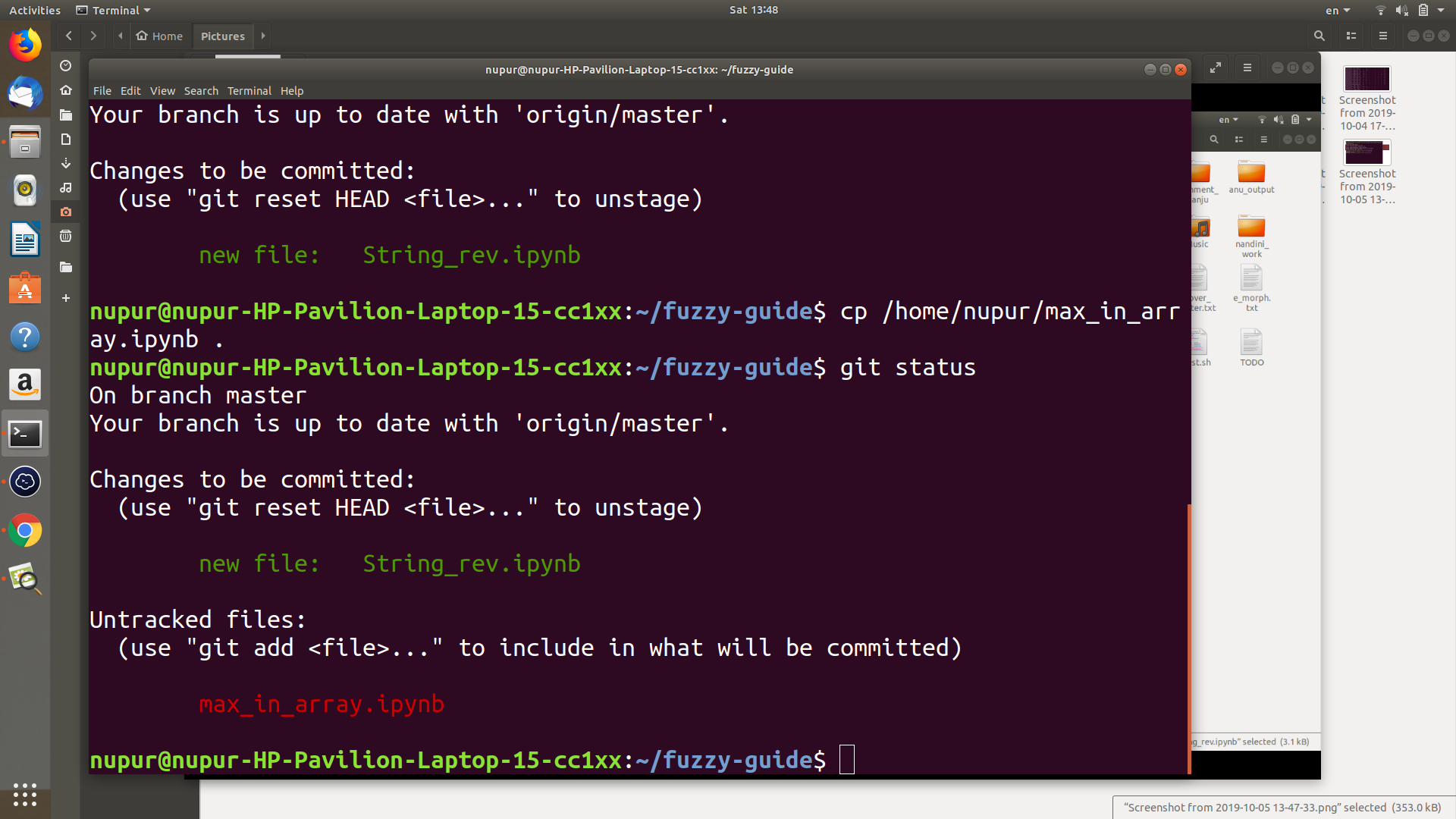
**(in the screenshot it displays *reinitialized* because I’ve already initialized it once)**

A new hidden directory called .git will now be present in your project directory. This is where Git stores its database and configuration information, so that it can track your project.

1. **Adding and Committing to a Repository**

git status – This command will show you what files have not been added to the list for the next commit. In most terminal apps, the unadded/added files will be red/green, respectively.

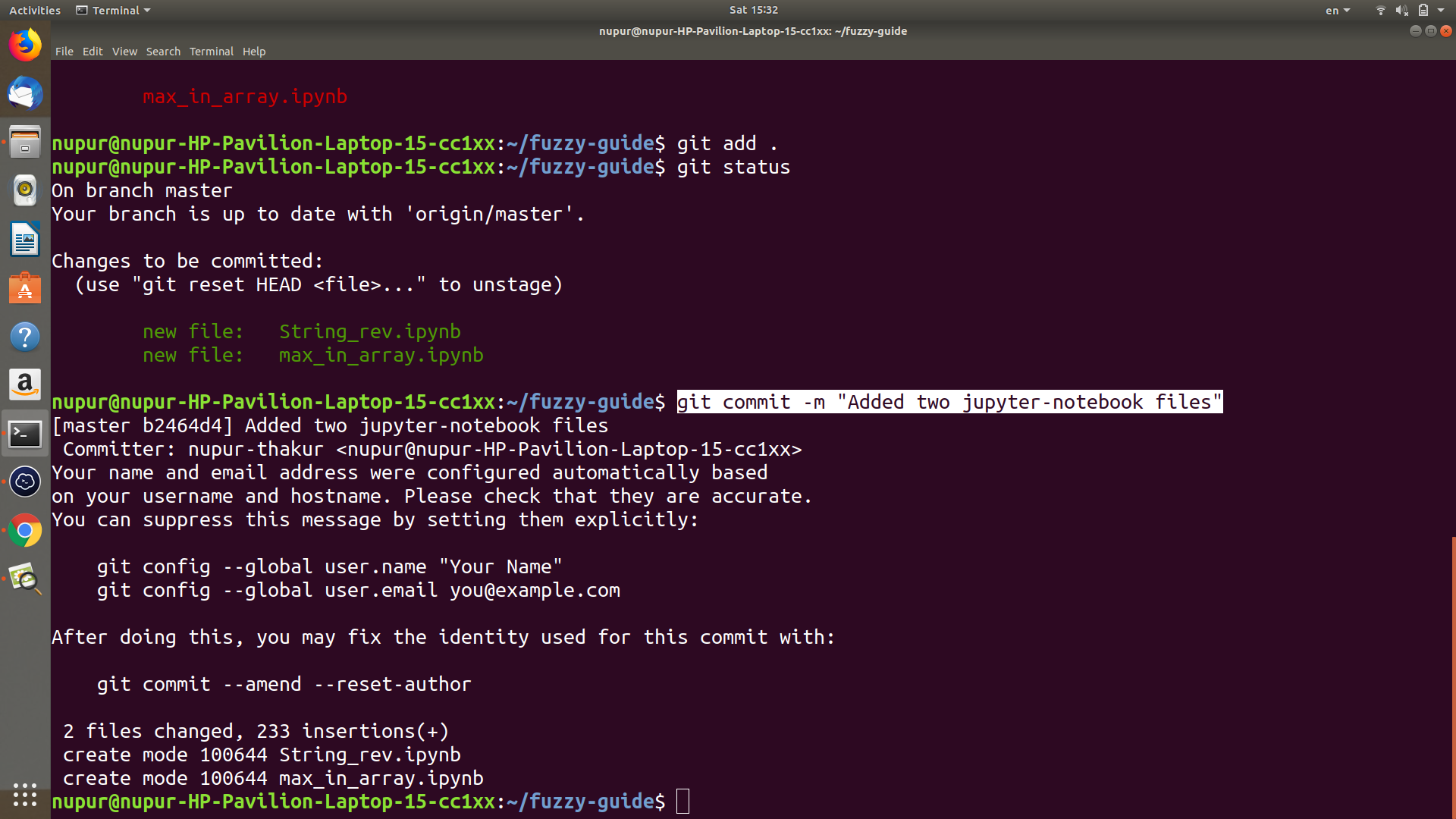




git add . – by adding a . at the end of the command, you’re telling git to include everything in the directory. If you want to add a single file, just use it’s filename. Such as git add String\_Rev.ipunb.

git commit -m “added two jupyter-notebook files” – When you commit these files, you should also leave a quick message to let other contributors (or your future self) know exactly what was in that commit. Standard git conventions say that you should be as concise and specific as possible.

To git commit with the long message. Just use the command **git commit** and press enter the text editor(nano) will open and then write your commit message in it to save the content press Ctrl+O and to exit press Ctrl+X in nano text-editor.

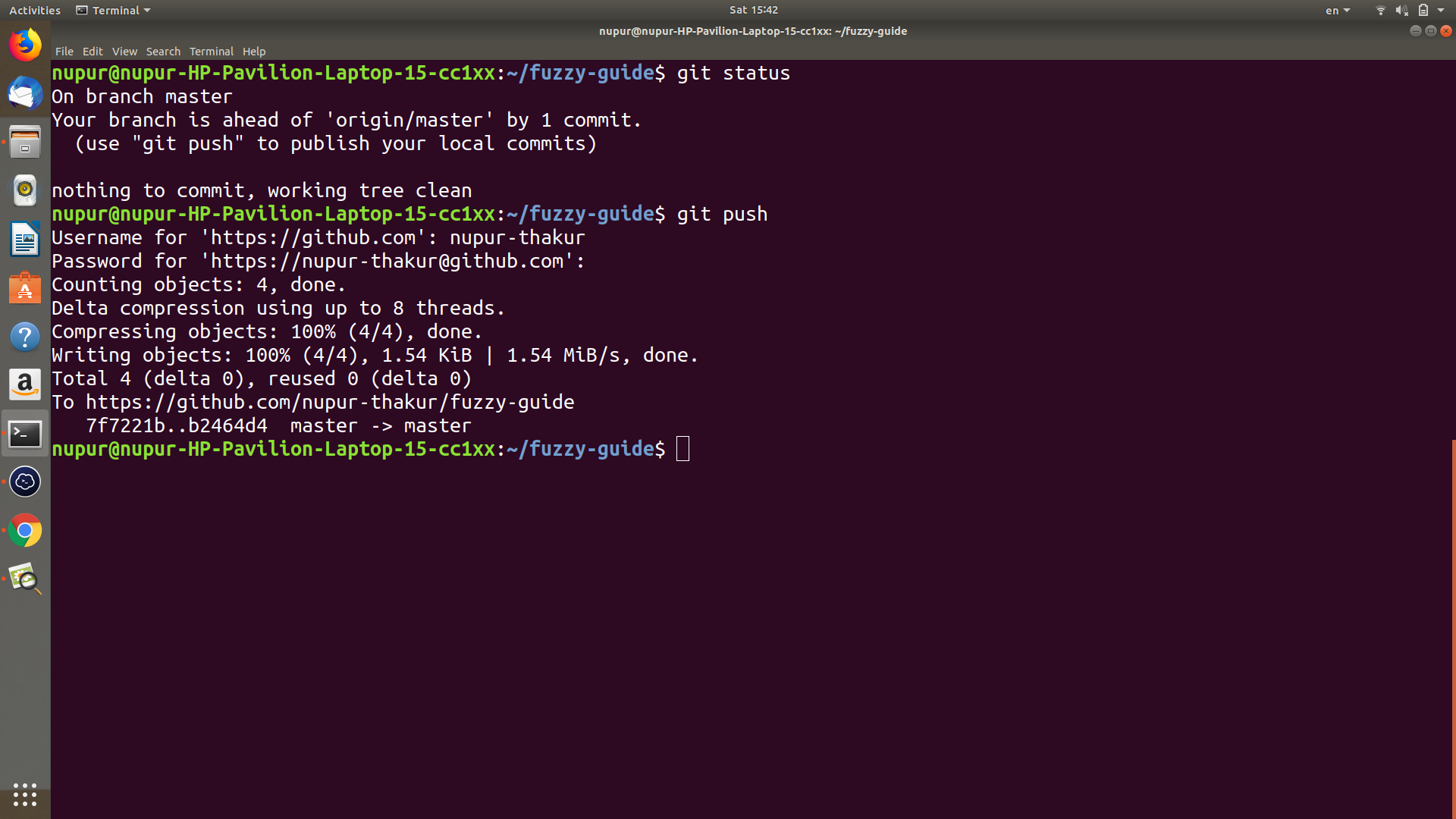


1. **Pushing to a Remote Git Repository like Github**

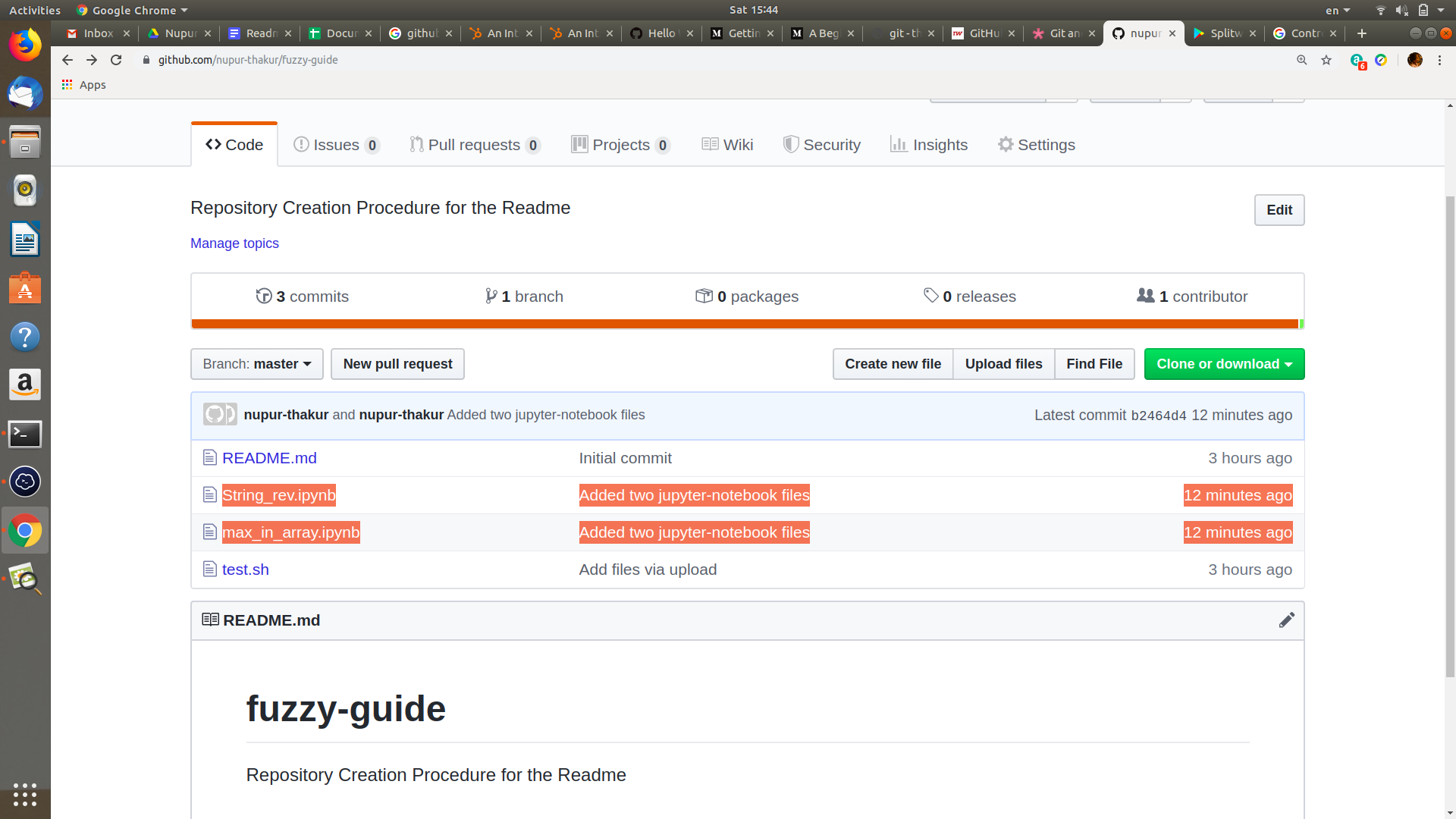
**git status** – You should always check to see how things look and to make sure you’re on the correct branch.

**git push**

**Enter your username and password for Github**( the password field will remain empty, even when you type but It’s okay. )



You can go to your Github account and check the repo, and your files will be there, commit messages and all.



1. **Branching and Merging**

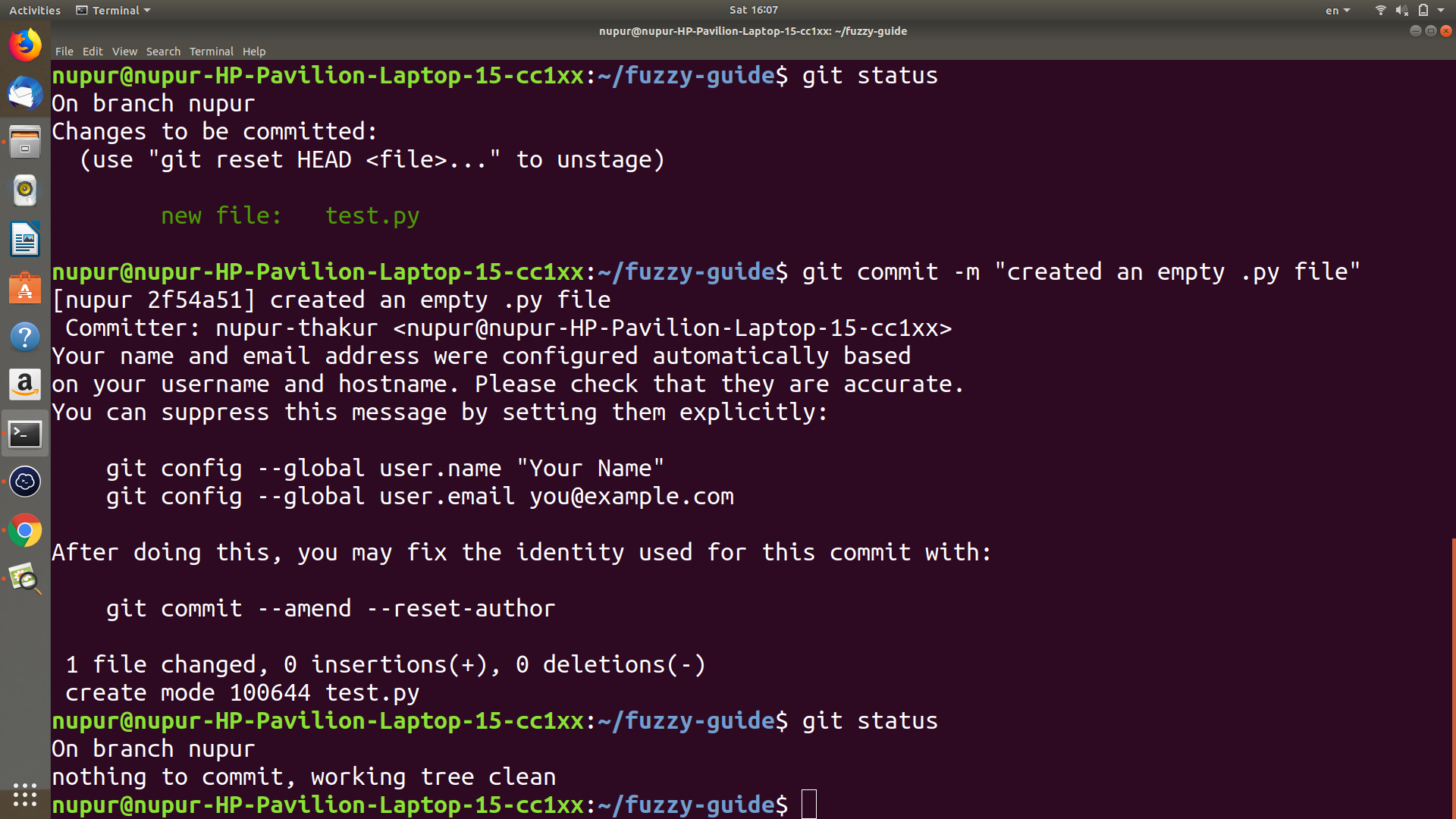
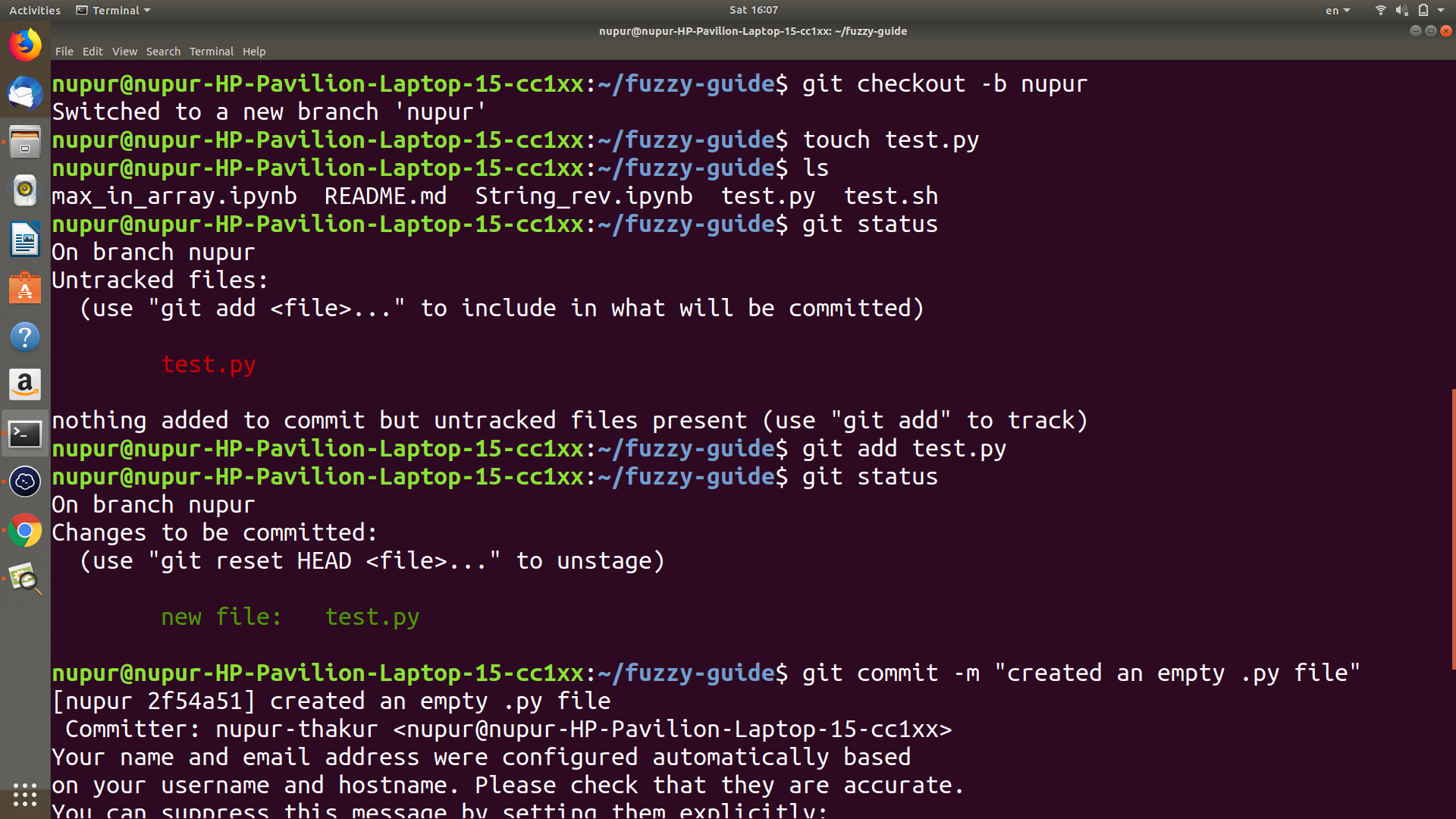
The final two commands you’ll need to be familiar with deal are branching and merging(and are very important when you are contributing to a repository who you are not the owner of).

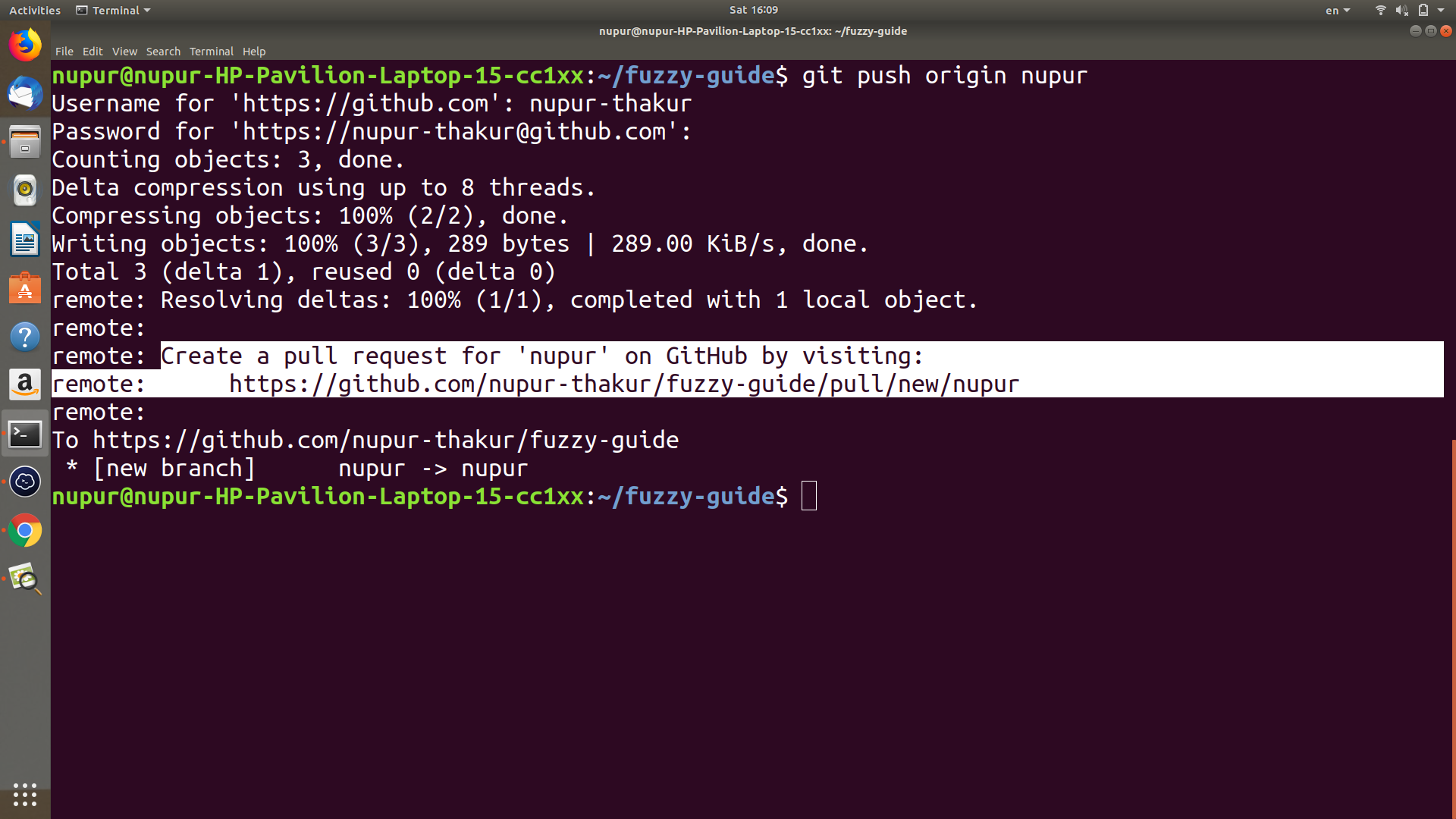
Working on a branch keeps the main code safe, and once your changes are all ready(basically when the file you want to push is all tested and ready), you can merge it all back together.

This can be very tricky at first but getting experienced with it makes it all easy.

git checkout -b <branch-name> – This command will both create a new branch that you title and swap you over to it as your working branch. It combines both the git checkout and git branch commands into one line of code.

touch test.py – this is just a linux cmd to create a new empty python file that I will merge back into the master.

And like before, you will **git status, git add test.py, git commit -m “created an empty .py file”** git status again will let you see that everything is as it should be.****

****

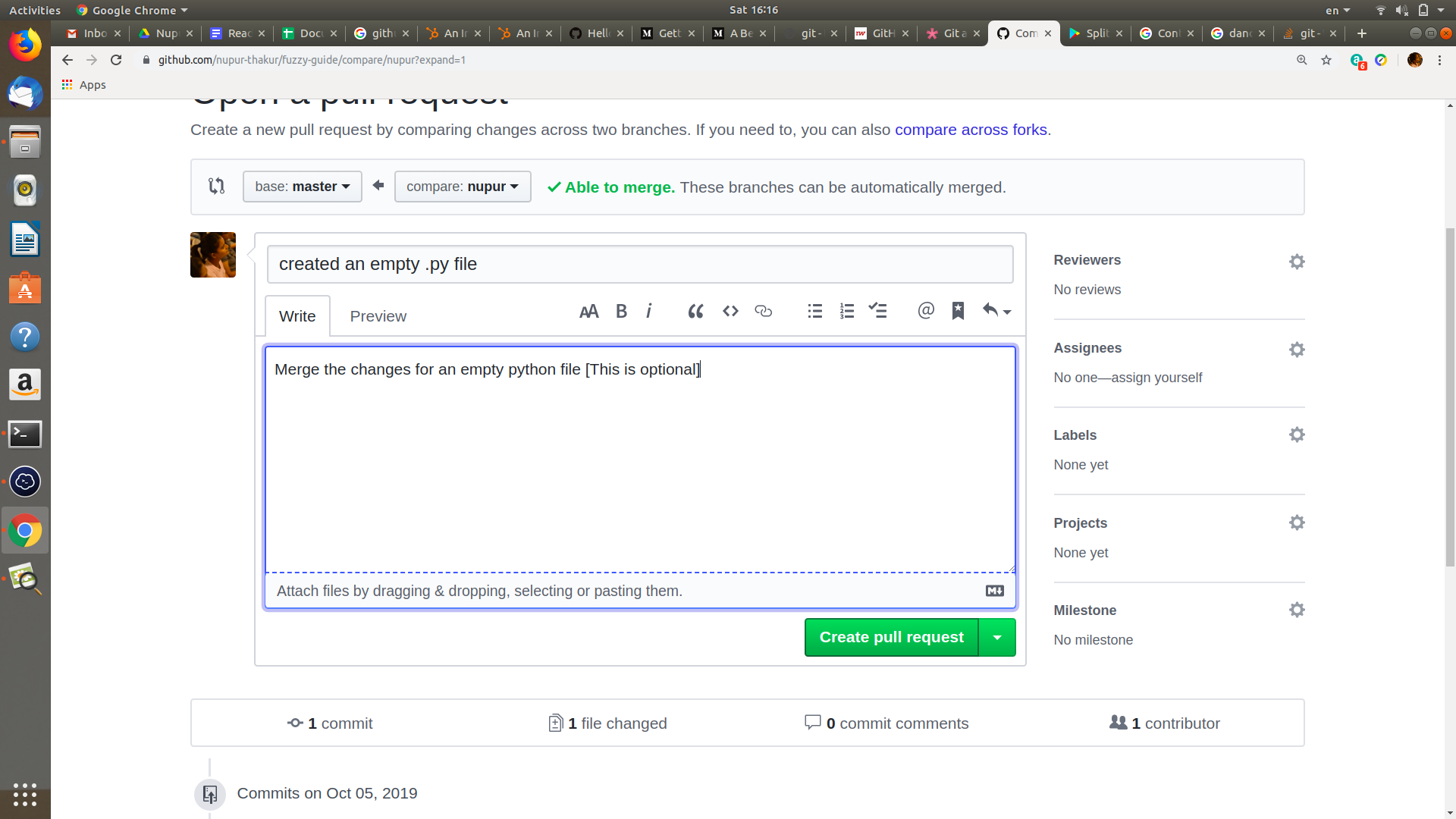
**(You can also use git push -u origin master) //-u is the shorthand for --set-upstream sets the default remote branch for the current local branch.**

To create a pull request either visit the link highlighted in the above screenshot or visit the repository on GitHub itself and change the branch to the one, of which you want to create a pull request.

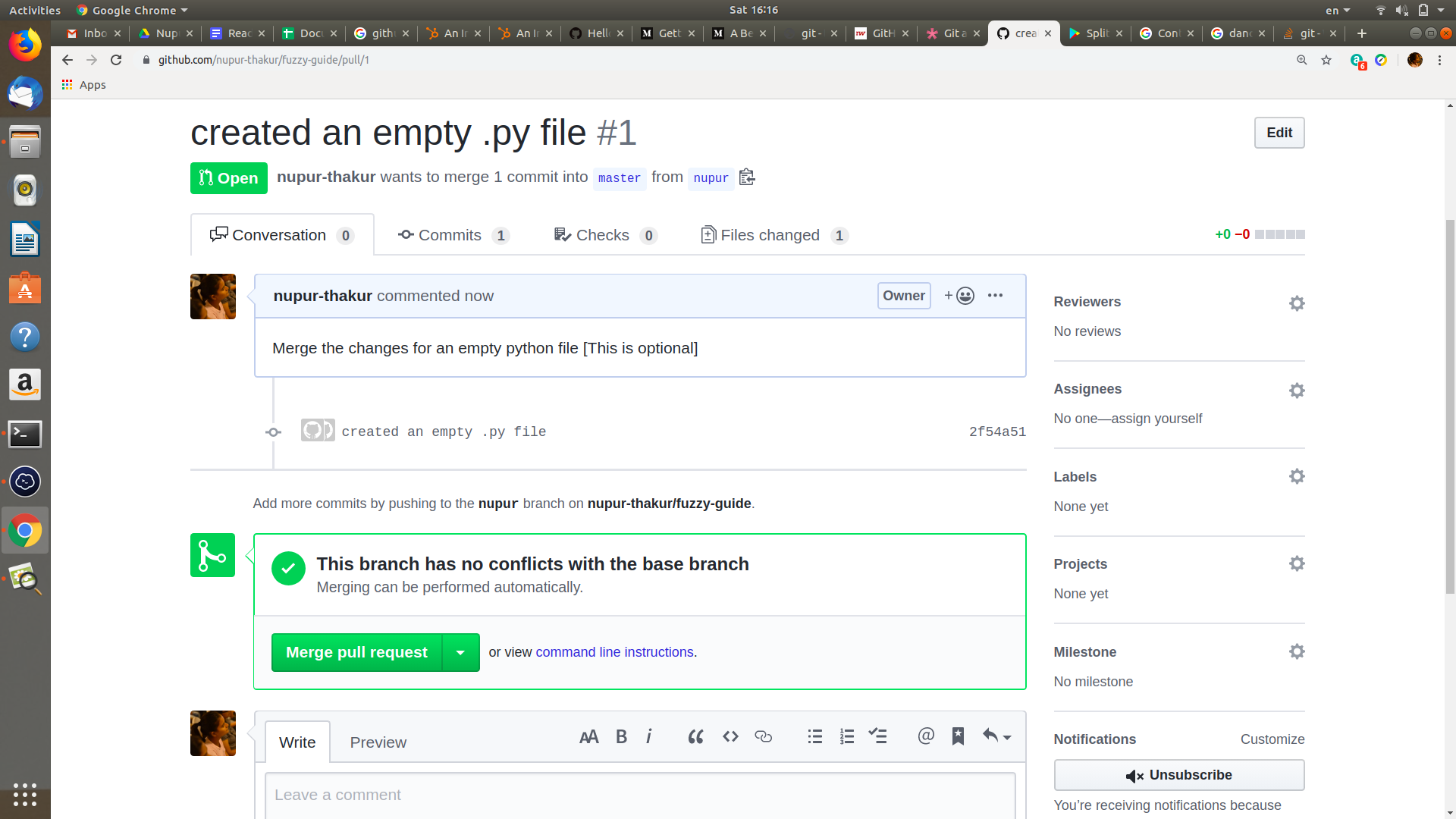
Also, You can switch back to the master branch with git checkout master (there’s no checkout -b this time because you only use that on creation of a new branch, not swapping.



**(Click on “Compare & pull request”)**



**(Click on “Create pull request”)**



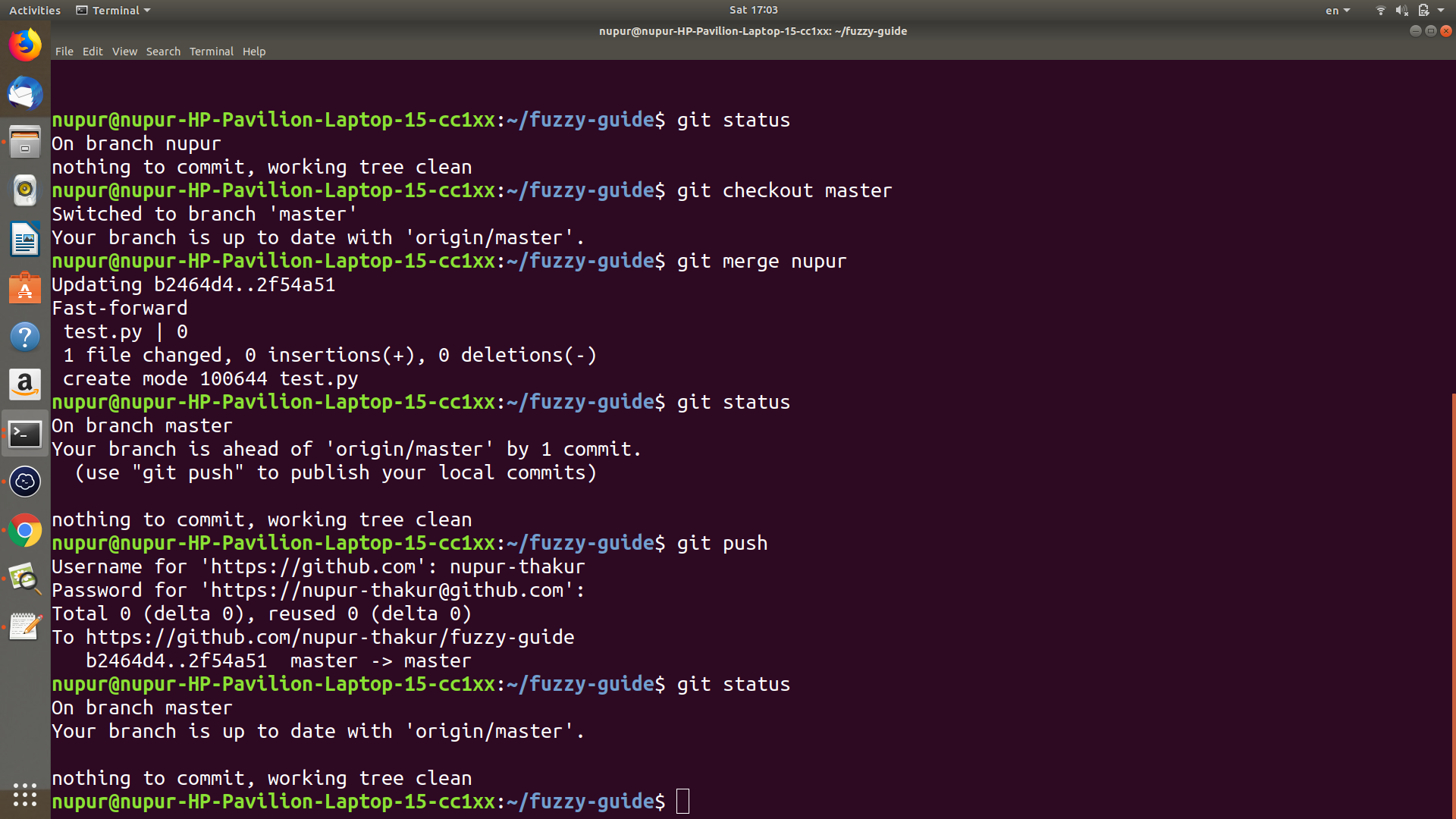
**[NOTE : NEVER MERGE A PULL REQUEST ON YOUR OWN BEFORE CONFIRMING IT WITH OTHER TEAM MEMBERS OR CONTRIBUTORS]**

**git status –** As always.

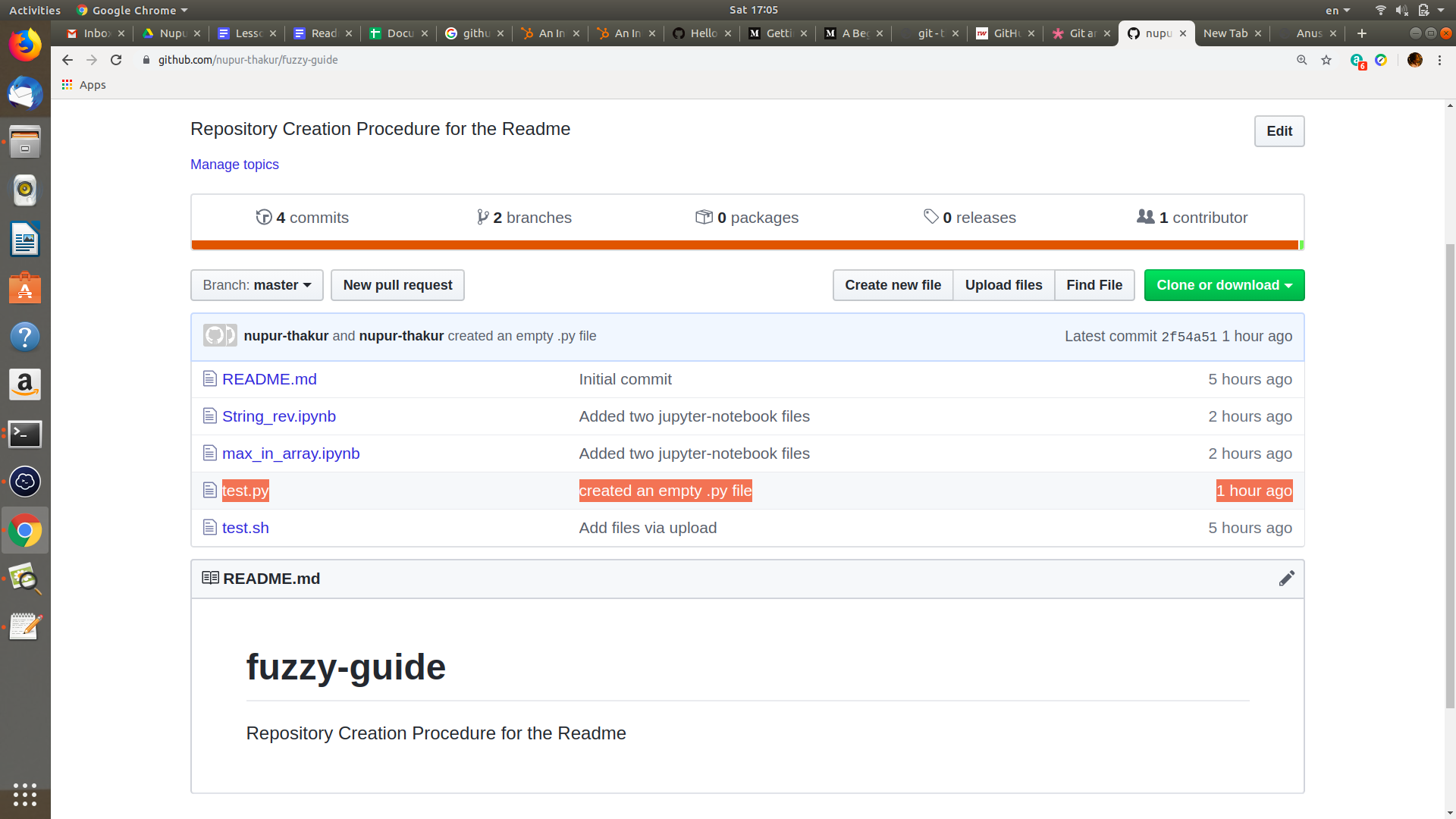
**git merge <branch-name> –** Since everything in that branch is fine, we can go ahead and merge.

If we git status again, we will see that we’re ahead of origin/master by 1 commit. That means that we haven’t pushed all the changes we’ve made to Github.

**git push** to sync things up!



**Now Check the Github repository–you will see everything you have done is reflected there. You can see the different commit messages, number of branches and commits that we’ve made, as well as any contributors to the repo and what they did, too.**

****

# ADDITIONAL INFORMATION

1. **origin** is an alias on your system for a particular remote repository. It's not actually a property of that repository.

By doing: **git push origin <branchname>**

you're saying to push to the origin repository. There's no requirement to name the remote repository origin: in fact the same repository could have a different alias for another developer.

Remotes are simply an alias that store the URL of repositories. You can see what URL belongs to each remote by using:**git remote -v**

In the push command, you can use remotes or you can simply use a URL directly. An example that uses the URL: **git push git@github.com:git/git.git master**

1. **Difference between PUBLIC and PRIVATE Repository**

Public Repositories : They're visible to any user on your GitHub Enterprise instance.

Private Repositories : They're only available to the repository owner. You can add collaborators of your choice to share with.

1. **Resolving a merge conflict using the command line**

[**https://help.github.com/en/articles/resolving-a-merge-conflict-using-the-command-line**](https://help.github.com/en/articles/resolving-a-merge-conflict-using-the-command-line)

# 