Table of Contents

[Version Control Systems: 2](#_Toc110626065)

[Git and GitHub 2](#_Toc110626066)

[Why Git is popular? 2](#_Toc110626067)

[Popular GUI tools for GIT 2](#_Toc110626068)

[Recommended Basic Configuration 2](#_Toc110626069)

[Basic Commands 3](#_Toc110626070)

[Staging Area 4](#_Toc110626071)

[Entering detailed description while commit 4](#_Toc110626072)

[How to ignore files from staging or commit 4](#_Toc110626073)

[How to remove files only from staging 5](#_Toc110626074)

[Git Diff Tools 5](#_Toc110626075)

[Global Settings to setup VS Code as default diff tool 5](#_Toc110626076)

[Branching and Merging 5](#_Toc110626077)

[Rename / Delete the branch: 6](#_Toc110626078)

[Merging, Conflicts and abort merging 6](#_Toc110626079)

[How to update the last commit without doing new commit 6](#_Toc110626080)

[Rollback to specified commit 6](#_Toc110626081)

[How to discard local changes / how to overwrite files from higher level 7](#_Toc110626082)

[How to do get repo from specific commit 7](#_Toc110626083)

[GIT HUB 7](#_Toc110626085)

[Git Clone 8](#_Toc110626086)

[Git remote 8](#_Toc110626087)

[Git Push 8](#_Toc110626088)

[Git Fetch 8](#_Toc110626089)

[Git Pull 8](#_Toc110626090)

[What are Pull Requests? 8](#_Toc110626091)

[Tags 9](#_Toc110626092)

[Steps to setup SSH in your local to push changes to Git Hub 9](#_Toc110626093)

[References: 10](#_Toc110626094)

# Version Control Systems:

VCS can be classified into 2 types

1. Centralized 🡺 single point of failure. If the server goes offline, you can’t work. Ex: TFS, VSS, SVN
2. Distributed 🡺 The every user has their own local copy of the repository. They can even work in offline. Ex: Git

# Git and GitHub

* Git is a distributed version control system for tracking changes in source code during software development.
* GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

Other popular web based repository is Bitbucket, Gitlab, Gerrit etc.

# Why Git is popular?

* Free
* Open Source
* Fast
* Scalable

# Popular GUI tools for GIT

* Gitkraken
* Source Tree

# Recommended Basic Configuration

* Name
* Email
* Default Editor
* Line Ending

Configurations can be done at 3 levels

1. System: Applicable for all users at system level
2. Global: All repositories of current user
3. Local: The current repository

$ git config --global user.name "theertha"

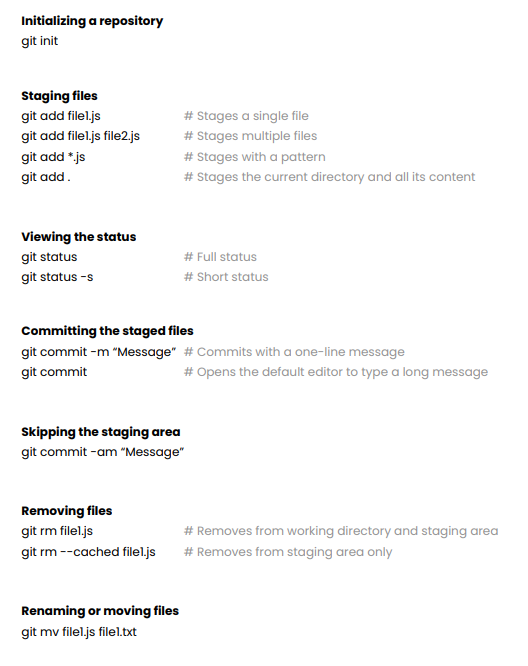
$ git config --global user.email theerthaks@gmail.com

$ git config --global core.editor "code --wait" 🡺 setting for vs-code

$ git config --global –e 🡺 this commands opens up the config in the default editor

$ git config --global core.autocrlf true 🡺 this is very crucial to configure end of line indicator

# Basic Commands



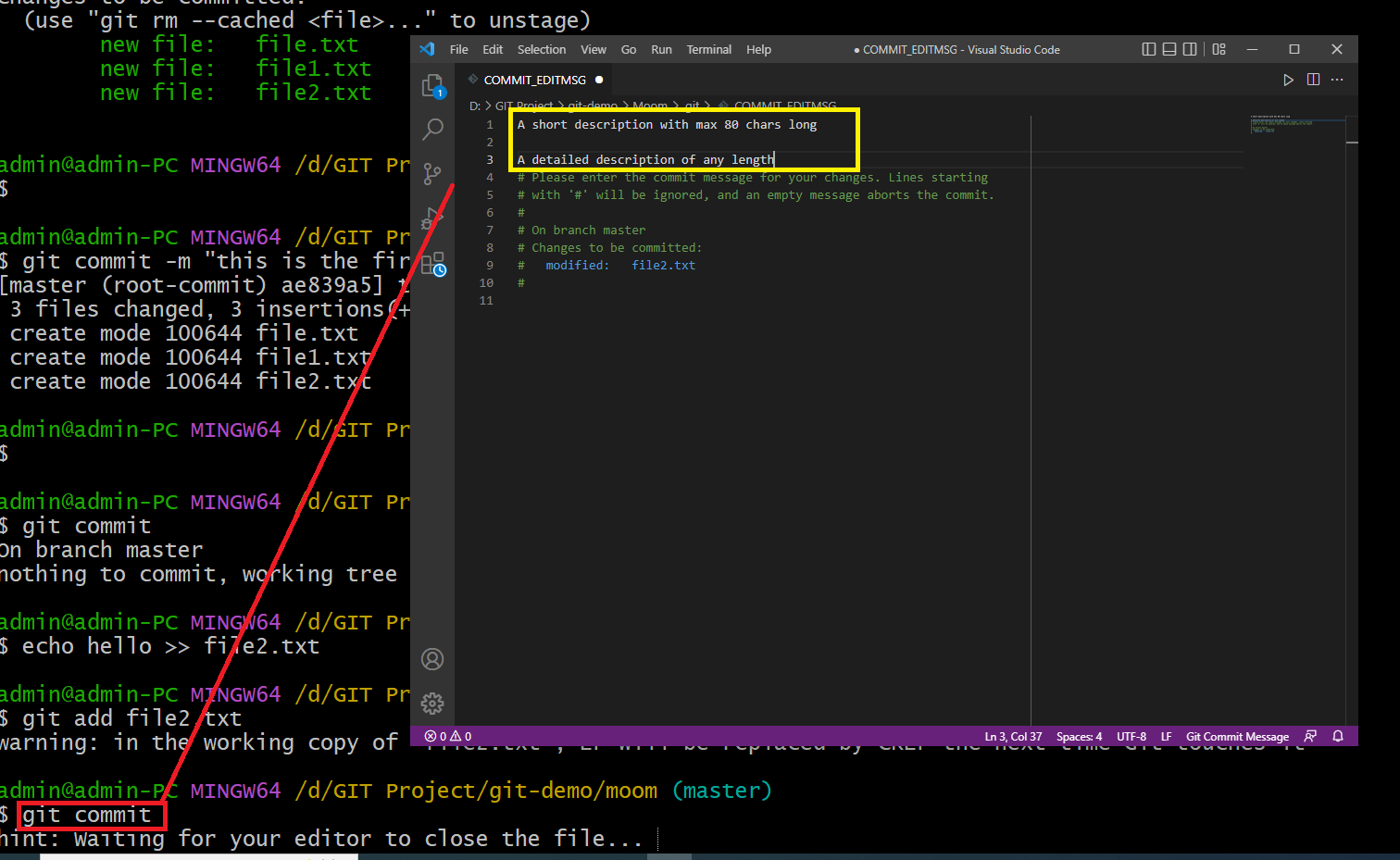
# Staging Area

The staging area is intermittent storage where work can be saved before pushing to repo. Staging is optional, if you are confident, then directly can push to Repo.

Once the commit is done from staging to repo, the files still remain in staging.

If you remove the file from work folder as if that is not required, the same should be removed from staging.

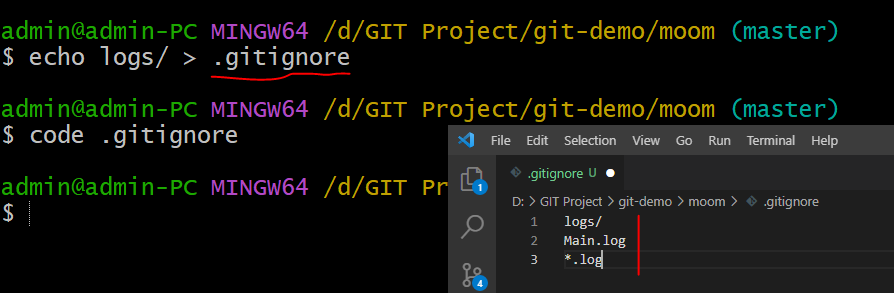
# Entering detailed description while commit



# How to ignore files from staging or commit

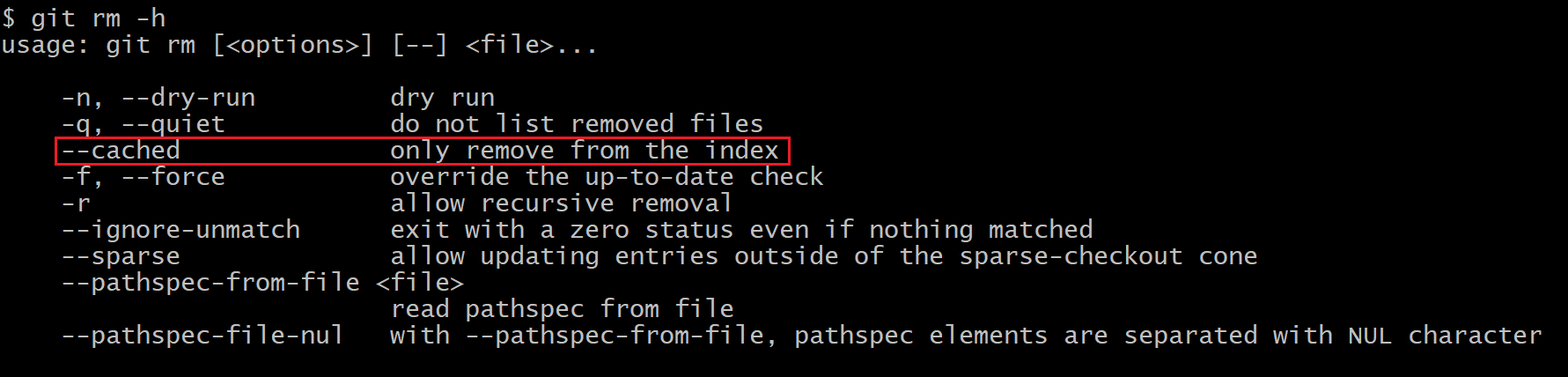
.gitignore is the file to have a list of files to ignore while committing.

**Note:** Do not forget to commit the .gitignore file to repo



[**https://github.com/github/gitignore**](https://github.com/github/gitignore) **🡺 Y**ou can get language specific standard templates

# How to remove files only from staging



$ git rm --cached -r

# Git Diff Tools

Most of the modern editor has inbuilt tools to compare the code versions. However there exist few popular tools, can use if required

* KDiff3
* P4Merge
* WinMerge
* VSCode

# Global Settings to setup VS Code as default diff tool

$ git config --global diff.tool vscode

$ git config --global diff.tool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"

**Note:** Sometime the above changes might not properly write to config. Hence open config file and ensure these entries are present

$ git difftool 🡺 opens up the VS Code with staged file and file in working directory

$ git difftool --staged 🡺 opens up the VS Code with staged file and file checked in during last commit

$ git difftool head 🡺 opens up the VS Code with local file and repo file

# Branching and Merging

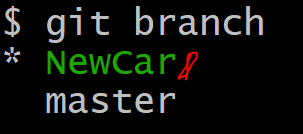
**Branching** is a way to take a copy from the master node and start working on that copy. When changes are done, **Merge** the changes to master node so that if next developer takes a new branch from master, he gets master node version which has changes done by previous developer.

1. $ git branch JPNagar\_Branch
2. $ git checkout -b JayaNagar\_Branch

Above are the 2 ways to create a branch from the branch you are presently checkout. Option 1 just creates a branch but still not checked out for edit, 2 is create and check out for edit.

Checkout is very important as it moves to HEAD to specified branch to track your changes.

$ git branch 🡺 will list down all the branch names and highlights the current one in green color



## Rename / Delete the branch:

Rename: Checkout to the branch and execute *$ git branch -m New\_Car*

Delete: Checkout to different branch and execute *$ git branch -d New\_Car*

## Merging, Conflicts and abort merging

Once you are done with changes in your branch, merge the changes to master or any other branch. In order to do so, first check out to TARGET branch and then issue merge command

*$ git merge YourLocalBranch*

Suppose if you have made some changes to TARGET Branch and changes to same files in SOURCE branch, then if you try to merge it, it will force you to resolve the conflicts by your own and merge it. You can check git status and *resolve the conflicts* in the error files

Just in case if you want to cancel merging in case of conflicts, then execute

*$ git merge –abort*

# How to update the last commit without doing new commit

Just in case if you committed the changes to local repository and soon after that you realize something needs to be corrected. Now either you can perform new commit or just update the last commit. It is highly recommended to do this only in your local repo.

It refers to updating the changes to repo without updating the last commit message. That’s it.

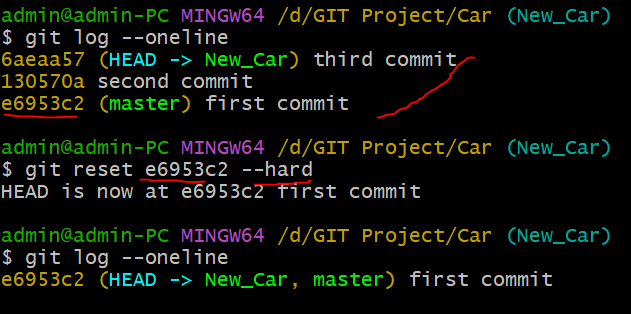
In order to do so, do your changes and push to staging and issue commit command without message but with option –amend

*$ git commit –amend*

# Rollback to specified commit

--hard 🡺 remove associated files from repo till working folder

*$ get reset ID\_OF\_COMMIT\_TILL\_WHERE\_RESET\_REQUIRED --hard*

**

# How to discard local changes / how to overwrite files from higher level

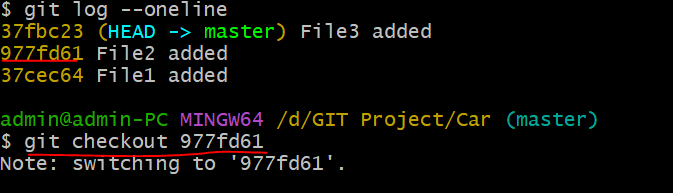
* $ git restore --staged file1.txt 🡺 Overwrite files from repo to staged
* $ git restore file1.txt 🡺 Overwrite files from staged to local
* $ git restore . 🡺 . refers to all content

# How to do get repo from specific commit

# It is same as moving to different branch using check out. 2 ways do that are:

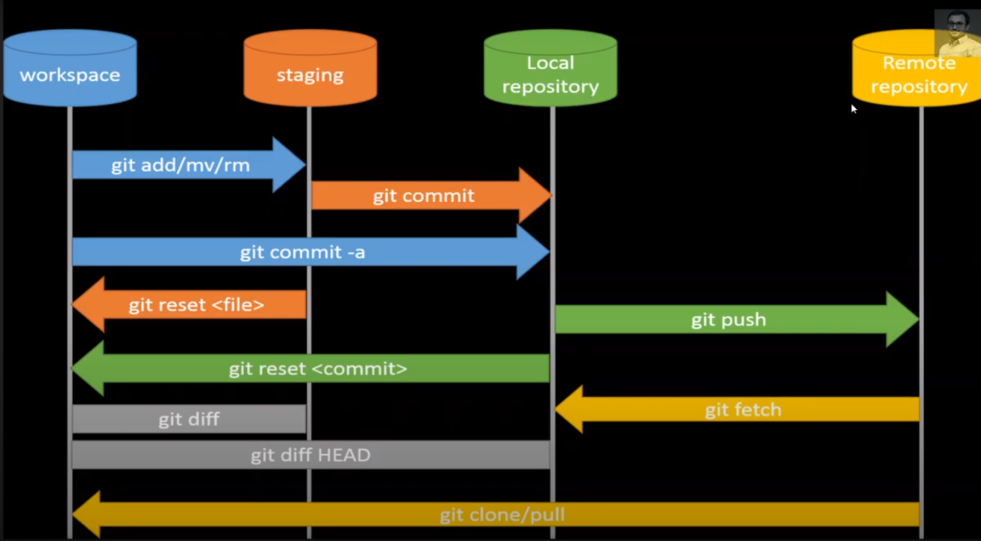
$ git checkout HEAD~2

$ git checkout 977fd61



# GIT HUB

Github allows us to store the git repository in the cloud / server. So that can be accessed from anywhere and shared between users



# Git Clone

To work with remote repository, first we need to have that repo in our local. Git clone command gets entire repo and its history to your local machine to work with. It matches default branch with local master branch.

*$ git clone <<REPO URL>>*

Note: When you cloned, no need to add remotes as it already set to Origin by default

To see all remote branches, use

*$ git branch –r*

In order to map other remote branch to local branch, use

*$ git switch RemoteBranch* 🡺 it will create a local branch with same name and map it

# Git remote

Remotes are nothing but a remote repository hosted in git hub

*$ git remote* -v 🡺 list down the repository to which your local repo is connected to.

In order to connect your local repo with remote repository, use:

*$ git remote add origin <<URL>>*

where origin is just an alias to that Url. You can use different alias as you wish. But commonly people use it as origin. No special meaning behind it

*$ git remote rename <<old name>> <<new name>>*

*$ git remote remove <<alias name>>* 🡺 detach your repo from remote repo

# Git Push

*$ git push Remote\_Alias\_Name Local\_Branch\_Name*

Ex: *$ git push origin master*

Note that you can push only one branch at a time to Git hub

# Git Fetch

Bring the remote data to local repo. User can review it and merge it with local work if required

*$ git fetch origin master*

# Git Pull

Overwrites the local working directory and repo from the remote repository

*$ git pull origin master*

# What are Pull Requests?

When user has pushed the changes to Git hub, it awaits for admin to review and merge the changes to master branch.

After pushing the changes from local, user has to login to Git hub and create a pull request. After it has been reviewed and approved, admin user can merge the changes to repository

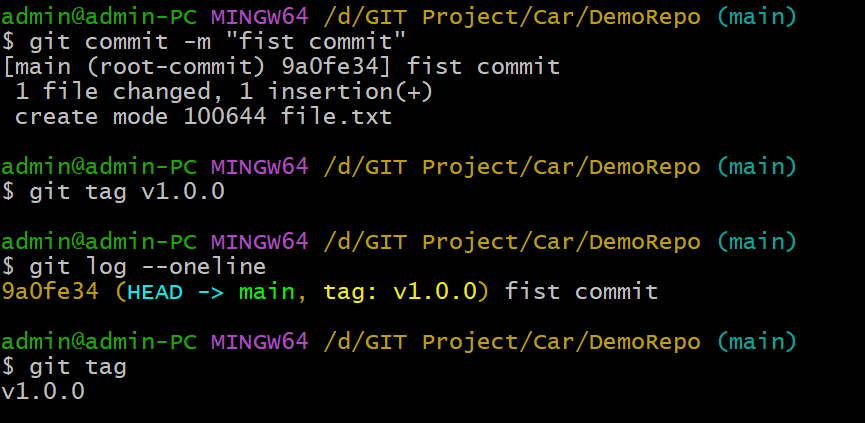
# Tags

They are nothing but assigning a version name to last commit.

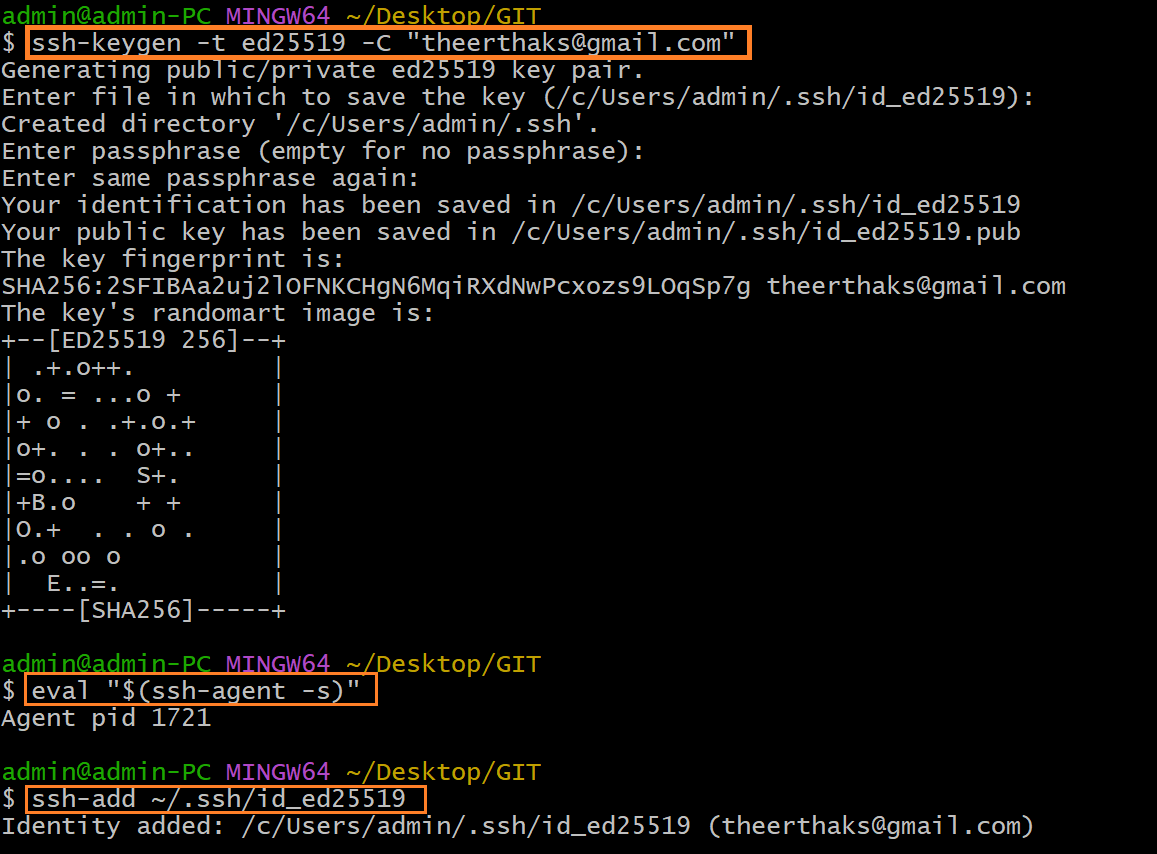
You can also push the repo to Git hub using tags.

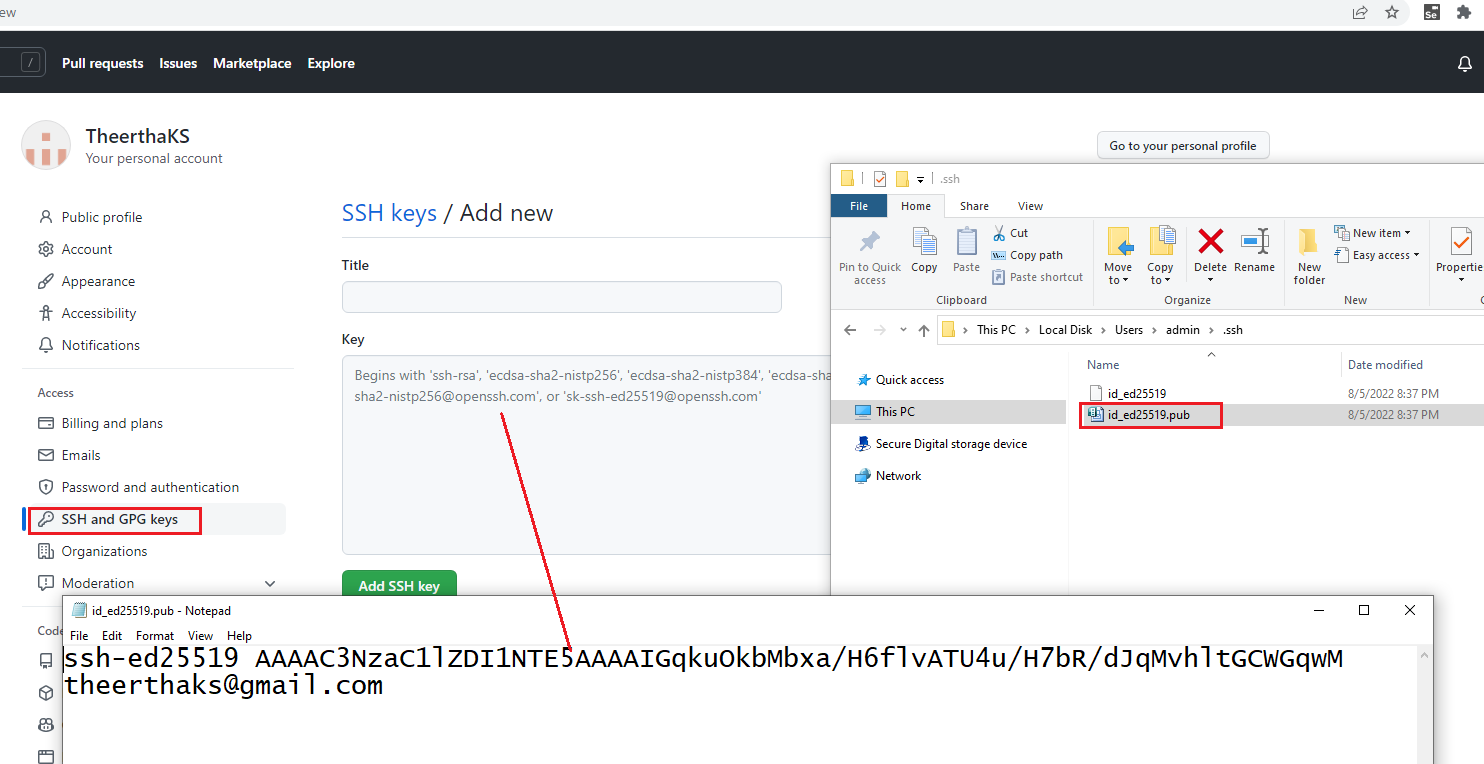
*$ git push origin V1.0.0 🡺 any specific tag*

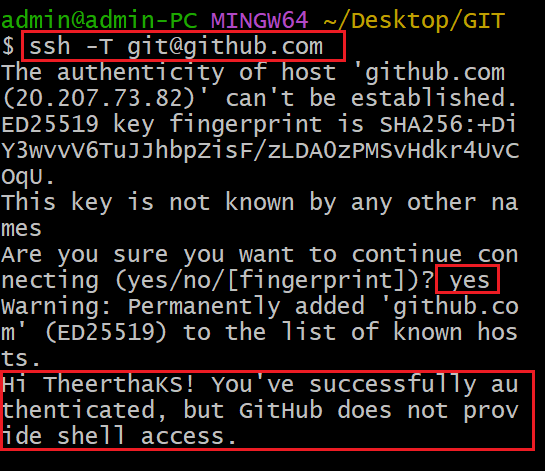
*$ git push origin --tags 🡺 all tags from local*



# Steps to setup SSH in your local to push changes to Git Hub







# References:

[**https://www.youtube.com/watch?v=vMdSqMf6BPY&list=PL\_euSNU\_eLbegnt7aR8I1gXfLhKZbxnYX**](https://www.youtube.com/watch?v=vMdSqMf6BPY&list=PL_euSNU_eLbegnt7aR8I1gXfLhKZbxnYX)