



# Git for Beginners

Johnathan Moe

# Chapter 1

## About this book

### About Git Version Control System

မင်္ဂလာပါ။အခုလို ဒီစာအုပ်လေး ဖြစ်မြောက်လာရတဲ့ ရည်ရွယ်ချက်ကတော့ လေ့လာစာ developer များအတွက် git နှင့်ပတ်သတ်သော လေ့လာစာများကို လွယ်ကူလျှင်မြန်စွာလေ့လာနိုင်စေရန်ဖြစ်ပါသည်။ စာရေးသူလေ့လာစဉ်က မှတ်သားထားသည်များကို စေတနာအပြည့်ဖြင့် ဖော်ပြအပ်ပါသည်။

Git ဆိုတာ version control system တစ်ခုဖြစ်ပြီး လူတိုင်း free ရယူသုံးစွဲနိုင်သော source code management system တစ်ခုဖြစ်ပါတယ်။ Git version control system ကို linux ဖခင်ကြီး (တီထွင်ခဲ့သူ) Linus Torvalds က linux kernel development အတွက် ရည်ရွယ်၍ စတင် developed လုပ်ခဲ့ပါတယ်။

Version Control System တစ်ခုဆိုတာက လက်ရှိသုံးနေတဲ့ version ကနေ နောက် update version တမျိုးကိုဖြစ်စေ လက်ရှိ version ကနေ အပြောင်းအလဲတစ်စုံတစ်ရာလုပ်တဲ့အခါမှာ အပြောင်းအလဲများကို မှတ်သားထားပြီး ထိုအပြောင်းအလဲ မှတ်သားထားသည်များကို လိုအပ်သော အခြေအနေသို့ပြန်လည်ခေါ်ဆိုနိုင်သော စနစ်ဖြစ်ပါတယ်။ဥပမာ...အကယ်၍ သင်ဟာ designer တစ်ယောက် ဆိုရင် project တစ်ခုစတင်ရန် file or folder တစ်ခုဖန်တီးရမည်။ ထို folder ထဲတွင် projectX.psd ဆိုသော project file တစ်ခုကို စတင်လိုက်သည်။ထို projectX ဆိုသောဖိုင်တွင် နောက်ခံအရောင်ကို အပြာရောင်ထားထားသည် ဆိုပါစို့။

Client ကိုပထမတကြိမ် ပြသောအခါ နောက်ခံအရောင်ကို အဝါရောင် အပြင် တခြားအပြောင်းအလဲများ ပြောင်းခိုင်းတဲ့အခါ ထိုဖိုင်ထဲတွင်ပင် save ထားလိုက်တယ်။နောက်တစ်ချိန်ပြန်ပြသော် အရင် နောက်ခံအရောင်နှင့် အရင်အတိုင်း ပြန်ပြင်လိုသောအခါ အရင် version သို့ ပြန်ရနိုင်ရန် အချိန်ကုန်နိုင်သည်။

ထို ပြဿနာများကို ဖြေရှင်းနိုင်ရန် version control system ကိုသုံးရန်လိုအပ်လာပါသည်။ Git သည် version control system တစ်ခုဖြစ်ပြီး github ကဲ့သို့သော cloud based server ကို အသုံးပြု၍ project file များ သိမ်းထားနိုင်သောကြောင့် computer system hardware တခုခုချို့ယွင်းသောကြောင့်ဖြစ်စေ operation system(window, mac, linux) တခုခုအမှားအယွင်းကြောင့်ဖြစ်သော data ပျက်ယွင်းမှုများကို လည်း ကာကွယ်နိုင်ပါတယ်။

Developer များအတွက် project တခုကိုရေးသောအခါ တစ်ယောက်တည်းဖြစ်စေ developer အများကြီးဖြစ်စေ ဝိုင်းရေးရသောအခါတွင်လည်း Git သည်အလွန်အသုံးဝင်သော စနစ်တစ်ခုဖြစ်သည်။ထို့ကြောင့် developer တစ်ယောက်အဖြစ် အသက်မွေးရန် ရည်ရွယ်ထားလျှင် git ကိုမဖြစ်မနေ တက်မြှောက်ထားရန်လိုအပ်ပါတယ်။

*Johnathan Moe*

# Chapter 2

## Git-Basic Concepts

### Version Control System

**Version Control System(VCS)** ဆိုတာက developer အများကြီး project တခုကို စီမံတဲ့အခါ အတူတကွအလုပ်လုပ်နိုင်ရန်နှင့် လုပ်ဆောင်ချက်များကို မှတ်သားထားနိုင်သော software တမျိုးဖြစ်သည်။

Version control system ရဲ့ လုပ်ဆောင်နိုင်ချက်များ

- Developer များ တပြိုင်နက်တည်း အတူတကွလုပ်ဆောင်နိုင်ခြင်း
- မိမိလုပ်ဆောင်နေသော အချက်များကို တခြား developer မှ လုပ်ဆောင်သောအချက်များကို save သောအခါ overwrite ဖြစ်မသွားနိုင်ခြင်း
- Version တိုင်းရဲ့ အခြေအနေများကို ထိန်းသိမ်းထားနိုင်ခြင်း

Version Control System(VCS) မှာ အမျိုးအစား ၂ မျိုးရှိတယ်။

- Centralized Version Control System ( CVCS)
- Distributed/Decentralized Version Control System ( DVCS)

ဒီ စာအုပ်မှာတော့ Distributed Version Control System ကိုပဲ အဓိကထားပြောပြသွားပါမယ်။

### Distributed Version Control System

Centralized version control system ဟာ central server တခုတည်းမှာတင် ဖိုင်အားလုံးသိမ်းထားပြီး developer team အကုန်လုံးကို အဲ့ server မှာတင် collaboration လုပ်ခွင့်ရှိပါတယ်။ Central server တခုတည်းသုံးရတဲ့အတွက်ကြောင့် အဆင်မပြေမှုတွေတွေ့ရှိနိုင်ပါတယ်။ဥပမာ ... central server သည် ၁နာရီကြာ down သွားခဲ့သည်ဆိုပါတော့ အဲ့အချိန်တွင် မည်သူမျှ collaborate မလုပ်နိုင်တော့ပါ။ အဆိုးဆုံးက server ပြန်ရခဲ့ရင်တောင် project ရဲ့ process history တစ်ခုလုံး ဆုံးရှုံးသွားနိုင်ပါတယ်။အဲ့ ပြဿနာတွေကို ဖြေရှင်းနိုင်ရန် DVCS ကိုသုံးလာကြပါတယ်။

DVCS ဟာ repository တခုရဲ့ နောက်ဆုံးအခြေအနေကိုသာမက process တခုလုံးကို trace လိုက်ထားတဲ့အတွက် server down သွားရင်တောင် client ကနေ ဘယ်အခြေအနေကိုမဆို restore ပြန်လုပ်နိုင်ပါတယ်။Client ဘက်က checkout မှန်သမျှကို အဲ့ repository မှာ backup လုပ်ပြီးသားဖြစ်နေမှာပါ။ Git ဟာ central server ပေါ်မူတည်မနေတဲ့ အတွက်ကြောင့် developer တယောက်ဟာ offline ဖြစ်နေရင်တောင် လုပ်ဆောင်ချက်များစွာကို လုပ်ဆောင်ထားနိုင်ပါတယ်။ Developer ဟာ လုပ်ဆောင်ချက်တွေကို changes အနေနဲ့ commit လုပ်ထားနိုင်ပါတယ်။Branch တွေခွဲပြီး သတ်မှတ်ထားနိုင်ပါတယ်။ထိုအပြင် တခြား developer တွေရဲ့ လုပ်ဆောင်ချက်တွေကို offline ဖြစ်နေတဲ့ အချိန်မှာလည်း ကြည့်နိုင်ပါတယ်။ မိမိ လုပ်ဆောင်ချက်တွေကို နောက်ဆုံးအနေနဲ့ သိမ်းထားပြီး repository ပေါ်တင်ကာမှာ internet ကိုလိုအပ်ပါတယ်။

## Advantage of Git

အရှင်းဆုံးအချက်ကတော့ Git က free ရယူနိုင်ပြီးတော့ open source ပဲဖြစ်ပါတယ်။ Internet ရှိရုံဖြင့် အခမဲ့ download ဆွဲနိုင်ပါတယ်။ Git ကိုအသုံးပြုပြီး project ပေါင်းမြောက်များစွာကို manage ပြုလုပ်နိုင်ပြီး အခမဲ့သုံးစွဲနိုင်ပါတယ်။ ပီးတော့ open source ဖြစ်တာကြောင့် သူ့ရဲ့ source code ကို download ဆွဲနိုင်ပြီး ကိုယ်အလိုရှိသလို ပြင်ဆင်အသုံးပြုနိုင်ပါသေးတယ်။

နောက်တချက်ကတော့ မြန်ဆန်ပြီး requirement မများတာပါ။ထို့အပြင် data ဆုံးရှုံးနိုင်မှုလည်း အများကြီးသက်သာသွားပါတယ်။ Git က secure hash function(SHA1) လို့ခေါ်တဲ့ common cryptographic hash function ကို သုံးပြီး သူ့ရဲ့ database ထဲကို သိမ်းပေးထားပါတယ်။ Checkout လုပ်လိုက်တိုင်းမှာ ပြောင်းလဲလိုက်တဲ့ ဖိုင်နဲ့ commit တိုင်းဟာ git ကနေ စစ်ဆေးပြီး ပြန်ထုတ်ပေးရပါတယ်။ဒါကြောင့်မလို့ git ကနေမသိလိုက်ပဲ git ရဲ့ database ထဲက file တွေပြောင်းတာ၊ အချိန်တွေပြောင်းတာ၊ commit ပေးတာတွေလုပ်မရလို့ security ပိုင်းလည်း စိတ်ချရပါတယ်။

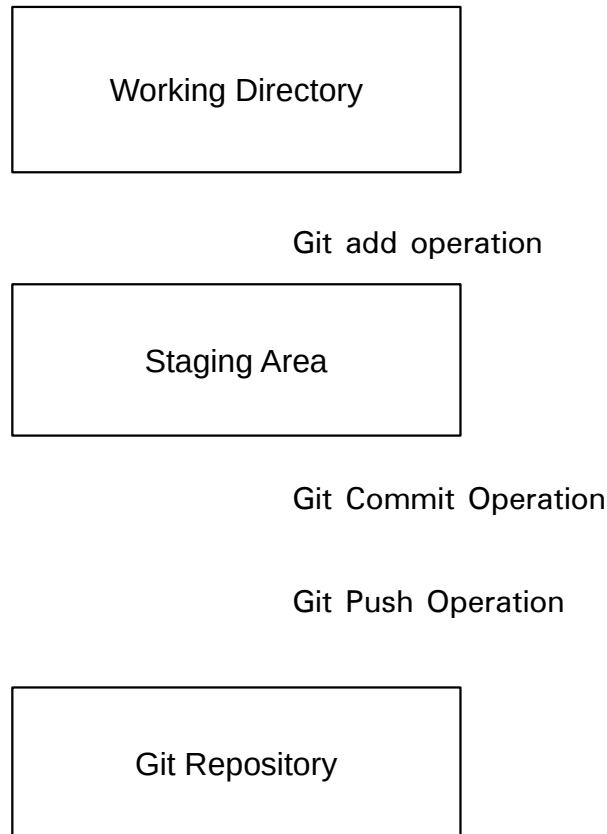
## Working Directory and Staging Area

Working Directory ဆိုတာက ဖိုင်တေ checkout စလုပ်တဲ့ နေရာကိုပြောတာပါ။ CVCS တွေမှာ ဆိုရင် developer တွေက ပြင်ဆင်ချက်တွေလုပ်ပြီးရင် တခါတည်းတိုက်ရိုက် repository ထဲသိမ်းလေ့ရှိပါတယ်။Git မှာတော့ တခြား ပုံစံတစ်မျိုးကို သုံးထားပါတယ်။ Git က developer ကနေပြင်လိုက်တဲ့ လုပ်ဆောင်ချက်တိုင်းကို track မလိုက်ထားပဲ developer က အဲ့လုပ်ဆောင်ချက်တွေကို commit လုပ်လိုက်တာမှာ working directory ကနေ staging area ထဲကိုပြောင်းပေးလိုက်ပါတယ်။ ဒီနေရာမှာ သိထားရမှာက staging area ထဲကဖိုင်တွေပဲ commit လုပ်ထားတယ်လို့သတ်မှတ်ပါတယ် ထပ်ပြီးပြင်ဆင်ထားတဲ့ code တေဖိုင်တွေ မပါဝင်ပါဘူး။

## Work Flow of Git

Git ရဲ့ အခြေခံ work flow ကို အပိုင်း ၃ ပိုင်းခွဲထားပါတယ်။

1. The Working Directory ( Working directory မှာ ဖိုင်တွေကို ပြင်ဆင်နိုင် ရေးသားနိုင်ပါတယ်)
2. The Staging Area( Git add operation ဖြင့်working directory မှာဖိုင်များကို staging သို့ ပြောင်းပေးလိုက်ပါတယ်)
3. Git Repository ( Git Commit operation ဖြင့် staging area ထဲမှ ဖိုင်များကို မှတ်ထားလိုက်တယ်။ထိုနောက် Git push operation ဖြင့် changes များကို git repository ထဲသို့ သိမ်းထားလိုက်တယ်)



## Blobs

Blobs ဆိုတာက Binary Large Object တွေကိုဆိုလိုပါတယ်။ Blobs တခုမှာ ဖိုင်ရဲ့ data ကို မှတ်ထားနိုင်တယ် ဒါပေမယ့် အဲဒီဖိုင်ရဲ့ မည်သည့် metadata မှပါဝင်နေမှာမဟုတ်ပါဘူး။ Git database မှာ SHA1 Hash နဲ့သိမ်းထားတဲ့ binary file တခုပဲဖြစ်ပါတယ်။

## Trees

Tree ဆိုတာ directory တခုကို ညွှန်ပြနေတဲ့ object တခုပဲဖြစ်တယ်။ Blobs တွေနဲ့ တခြား sub-directories တွေပါဝင်နေတဲ့ binary file တခုပဲဖြစ်ပါတယ်။

## Commits

Repository တခုရဲ့ current state ကို commit လုပ်ပီးမှတ်သားထားနိုင်ပါတယ်။ Commit တွေကိုလည်း SHA1 hash နဲ့ Git ထဲမှာ နာမည်ပေးပီးသိမ်းပါတယ်။ Git Commit operation ဖြင့် project ရဲ့ လက်ရှိပြင်ဆင်ချက်တွေကို snapshot တစ်ခုအနေနဲ့ မှတ်သားထားလိုက်ပီး ထိုအခြေအနေကို project safe version အနေနဲ့သတ်မှတ်ထားလိုက်လို့ရပါတယ်။ နောက်ထပ် commit ထပ်လုပ်ပီးရင်လည်း အရင် မှတ်ထားတဲ့ safe version ကိုပြန်သွားနိုင်ပါတယ်။

## **Branches**

လက်ရှိ **project** တခုရဲ့ နောက် **version** တခုခုကိုဖြစ်စေ နောက်ထပ်ခြားနားသောလုပ်ဆောင်ချက်များကို ထပ်ထည့်သောအခါဖြစ်စေ **branch** များ ခွဲပီးအလုပ်လုပ်ပါတယ်။ **Default** အနေနဲ့ **repository** တခုဆောက်ပီးလျှင် **master/main branch** တွင် အလုပ်လုပ်ပါတယ်။လက်ရှိ **project** ကနေ **new feature** ထပ်ထည့်လိုသောအခါ **branch** အသစ်ဆောက်ပီး ထို **new feature** ပီးသွားသောအခါ **main branch** သို့ **merged** လုပ်နိုင်ပါတယ်။

## **Merge**

ခွဲထားတဲ့ **branches** တွေကနေ **main branch** ကို ပြန်ပေါင်းတဲ့အခါသုံးပါတယ်။

## **Tags**

**Tags** တွေဟာလည်း **branches** တွေလိုပါပဲ မတူတာက **tags** တွေက **immutable** ဖြစ်ပါတယ်။ ဆိုလိုတာက **tag** က လည်း **branch** တခုပါပဲ သို့ပေမယ့် အဲ့ **branch** မှာ **modify** ထပ်မလုပ်တော့တဲ့အခါသုံးပါတယ်။ **Tag** တခုကို **create** လုပ်ပီး **commit** နောက်တခု အသစ်ပေးတဲ့အခါ အဲ့ **commit** က **update** မလုပ်တော့ပါ။ထို့ကြောင့် များသောအားဖြင့် **developer** တွေဟာ **tags** တွေကို **project** တခုလုံးရဲ့ အပြီးသတ် **version release** လုပ်တဲ့အခါပဲသုံးပါတယ်။

## **Clone**

တခြား **developer** ရဲ့ **project repository** တခုကို ပုံတူကူးတဲ့အခါဖြစ်စေ မိမိ **project** ကို နောက် **computer** တလုံးရဲ့ **local** မှာသုံးချင်တာဖြစ်စေ **clone operation** ကိုသုံးရပါတယ်။

## **Pull**

**Project repository** တခုတည်းမှာ တခြား **developer** ရဲ့ ပြင်ဆင်ထားတဲ့ **code** တွေကို **local** မှာ ပြန်ယူပြီး စမ်းသပ်တဲ့အခါဖြစ်စေ မိမိကိုယ်တိုင်ရေးထားတဲ့ **latest changes** ကို **local** မှာ ပြန်ယူတာဖြစ်စေ **pull operation** ကိုသုံးရပါတယ်။

## **Push**

**Local** မှာ မိမိကိုယ်တိုင်ပြင်ထားတဲ့ **code changes** တွေကို **project repository** ထဲသို့ ပို့ပြီးသိမ်းသောအခါတွင် သုံးရပါတယ်။

## **Head**

**Head** ဆိုတာက **pointer** တခုအနေနဲ့သတ်မှတ်ပါတယ်။ **Branch** တခုရဲ့ **latest changes** ဖြစ်သွားတဲ့ **commit** ကို **point** လုပ်ပေးပါတယ်။ **Developer** တယောက်က **commit** လုပ်လိုက်တိုင်း **Head** ထဲမှာ **update** ဖြစ်သွားမှာပါ။**Branches** တွေရဲ့ **head** တွေကို ဒီ **path** ထဲမှာ တွေ့နိုင်ပါတယ် **.git/refs/heads** ။

## Revision

Revision တွေကို source code ရဲ့ version အနေနဲ့သတ်မှတ်ပါတယ်။

## URL

URL ကတော့ ထုံးစံအတိုင်း project ရဲ့ git repository location ကိုဖော်ပြပေးပါတယ်။ git url ကို config file ထဲမှာ သိမ်းထားပါတယ်။

## Checkout

လက်ရှိ changes ကနေ မိမိလိုအပ်တဲ့ အခြေအနေကို သွားချင်တဲ့အခါ checkout ကို သုံးပါတယ်။ checkout condition မှာ ပြင်မရပါ။

## Revert

မိမိလိုချင်တဲ့ အခြေအနေအထိပဲ လိုချင်ပြီး အဲ့နောက်က changes တွေကိုဖျက်ချင်တဲ့အခါ revert ကို သုံးပါတယ်။ Revert ကို သုံးတဲ့အခါ commit တစ်ခုပါဖန်တီးပြီးတဲ့ အတွက် revert commit အခြေအနေကို ပြန်ခေါ်သုံးလို့ရပါတယ်။

## Reset

မိမိလိုချင်တဲ့ commit ကလွဲပြီး ကျန်တာအကုန်ဖျက်ချင်တဲ့အခါ reset ကိုသုံးရပါတယ်။ အရေးကြီးတာက reset သုံးပြီးတဲ့အခါ revert လို commit မချန်ခဲ့တဲ့အတွက် လိုချင်တဲ့အခြေအနေကို ပြန်မရနိုင်ပါ။ သေချာမှ အသုံးပြုဖို့လိုပါတယ်။

လက်ရှိတော့ သိထားရမယ့် git basic term လေးတွေကို စုံလင်အောင်ဖော်ပြပေးထားပါတယ်။ ဒီအခြေခံတွေကို သိပီဆိုရင် စတင်အသုံးပြုမယ့် environment setup အကြောင်းဆက်ဖော်ပြပေးသွားပါမယ်။

# Chapter 3

## Git – Environment Setup

Git ကို အသုံးပြုခင် လိုအပ်တဲ့ basic configuration လေးတွေလုပ်ကြပါစို့။ Ubuntu နှင့် Centos linux တွေမှာ Git Client install လုပ်ပုံကို အရင်ဖော်ပြပေးသွားပါမယ်။

### Installation of Git Client in Linux

Debian base GNU/Linux distribution တေအတွက်ဆိုရင် apt-get command ဖြင့် အလွယ်တကူ install နိုင်ပါတယ်။ သုံးရမယ့် command ကတော့

01. Start by updating the package index:

```
$ sudo apt update
```

02. Run the following command to install Git:

```
$ sudo apt install git
```

03. Verify the installation by typing the following command which will print the Git version:

```
$ git --version
```

git install အဆင်ပြေမပြေ git --version ကို ရိုတ်ပီးစစ်ဆေးနိုင်ပါတယ်။

### Installation of Git on window

Window မှာ install လုပ်ဖို့ကတော့ ရှင်းပါတယ်။ ကိုယ်သုံးထားတဲ့ window ရဲ့ version ပေါ်မူတည် ပီး 32bit နဲ့ 64bit ကို ကိုက်ညီတဲ့ exe file download ဆွဲပီး install လိုက်ရုံပါပဲ။

install လုပ်ရမယ့် exe file ကို ဒီလင့်မှာ ရယူနိုင်ပါတယ်။ <https://git-scm.com/download/win>  
window မှာ linux တို့ mac တို့လို terminal မရှိတဲ့အတွက် window command line ကနေလည်းသုံးလို့ ရသလို git bash ကို install လုပ်ပီးလည်းသုံးနိုင်ပါတယ်။ install လုပ်ပီးပါက git --version ဖြင့်စစ်ဆေး ကြည့်နိုင်ပါတယ်။



## Installation of Git on Mac

Mac မှာလည်း git ကို install လုပ်နိုင်တဲ့နည်းတွေအများကြီးထဲကမှ အလွယ်ကူဆုံးနည်းကိုပဲပြောပြပေးလိုက်ပါမယ်။ <https://sourceforge.net/projects/git-osx-installer/> ဒီလင့်ကနေ download ဆွဲယူပြီး အဆင့်လိုက် install လိုက်ရပါပဲ။

## First time Setup Configuration

Git installation အောင်မြင်ပြီးရင် configuration လုပ်ရပါမယ်။အောက်ပါ command တွေသုံးပြီး တစ်ခါတည်း configure လုပ်လိုက်ရအောင်ပါ။

```
$ git config --global user.name "Your Name"
$ git config --global user.email "youremail@yourdomain.com"
```

configuration အောင်မြင်ပြီးရင် ဒီ command သုံးပြီး တစ်ချက်စစ်ကြည့်ရအောင်။

```
$ git config --list
```

The output should look something like this:

```
Output
user.name=Your Name
user.email=youremail@yourdomain.com
```

ဒီ output ထဲကလို မိမိရဲ့ username နဲ့ email ပေါ်လာရပါမယ်။ဒါဆိုရင် configuration လုပ်တာ အောင်မြင်ပါလို့ဆိုလိုရပါပဲ။

# Chapter 4

## Git Basics

Git Basic Concepts မှာ ဖော်ပြခဲ့တဲ့ concepts တွေနဲ့ term တွေကို ဖတ်ထားပြီးဖြစ်လို့ လက်တွေ့ အသုံးပြုမယ့်အချိန်မှာ မှတ်သားရပိုလွယ်မယ်လို့ထင်ပါတယ်။ ဒီ chapter မှာတော့ တကယ့်လက်တွေ့အသုံးချ ရမယ့် basic အားလုံးကို လက်တွေ့သုံးရင်း ပြောပြပေးသွားပါမယ်။

### Getting A Git Repository

Git repository တစ်ခုကိုတည်ဆောက်တော့မယ်ဆိုရင် အောက်ပါ ၂ နည်းထဲကနေ တည်ဆောက်နိုင် ပါတယ်။

1. Version Control System ထဲမှာ မတည်ဆောက်ရသေးတဲ့ project directory ကိုသွားပြီး repository တစ်ခုတည်ဆောက်နိုင်သလို
2. ရှိပြီးသား repository တစ်ခုကနေလည်း clone လုပ်နိုင်ပါတယ်။

### Initializing a Repository

ပထမဆုံးအနေနဲ့ version control system ထဲမှာ မရှိသေးတဲ့ project အသစ်တစ်ခုမှာ စတင်ပြီး repository တစ်ခုစီကြည့်ရအောင်ပါ။ မိမိနှစ်သက်ရာ လမ်းကြောင်းမှာ project folder အသစ်တစ်ခု create လုပ်လိုက်ပါ။ Linux နှင့် Mac အတွက်ဆိုရင် terminal ကိုဖွင့်ပြီး မိမိ project ရှိရာကို လမ်းကြောင်းရွေးပြီး ဝင်လိုက်ပါ။ Window အတွက်ဆိုရင် git bash or cmd ဖြင့် project ရှိရာကိုဝင်လိုက်ပါ။

for Linux:

```
$ cd /home/user/my_project
```

for macOS:

```
$ cd /Users/user/my_project
```

for Windows:

```
$ cd C:/Users/user/my_project
```

နောက်တစ်နည်းအနေနဲ့ project ဖိုင်ရှိရာသို့ဝင်၍ open in terminal ဖြင့် ဖွင့်ပါ။ Terminal ထဲတွင် code . Command ကို သုံး၍ vs code ကိုဖွင့်ပြီး vs code ထဲမှ terminal ဖြင့် သုံး၍လည်းရပါတယ်။ Terminal ဖွင့်ပြီးပီဆိုလျှင် git init ဆိုသော command ကိုသုံး၍ repository ကိုတည်ဆောက်နိုင်ပါတယ်။ Project folder ထဲတွင်.git ဟူသော repository file ကိုထည့်သွင်းသွားပါလိမ့်မယ်။

## Git init

Repository တစ်ခုအစတင်ရာတွင်အသုံးပြုပါတယ်။project file ထဲတွင် git ကို စသုံးရန် command ပေးလိုက်ခြင်းဖြစ်ပါတယ်။ထိုအခြေအနေမှာတော့ ဘယ် changes ကိုမှ track လုပ်ထားမှာ မဟုတ်ပါဘူး။

ထိုအခြေအနေသည် အထက်ပါ ဖော်ပြထားသော git work flow တွင် working directory အခြေအနေ တွင်ရှိနေသေးပါတယ်။

## Git add

Project ထဲတွင် မိမိပြောင်းလဲလိုက်သော code များနှင့် file များကို စတင် track လုပ်လိုလျှင် git add ဆိုသော command ဖြင့် မိမိ track လုပ်လိုသောဖိုင်များကို ထည့်သွင်းပေးရပါတယ်။ထိုနောက်မှာgit commit ဆိုသော command ကို သုံးပြီး add ထားသောဖိုင်များကို staging area ကိုဆွဲခေါ်သွားပါတယ်။

eg. \$ git add \*.c

eg. \$ git add . ဖြင့်လည်း project အတွင်းရှိ ဖိုင်များအားလုံးကို add နိုင်ပါတယ်။

## Git Commit

add လုပ်ထားသောဖိုင်များကို staging area သို့ဆွဲခေါ်သွားနိုင်ရန် git commit ကိုသုံးပေးရတယ်။

eg. \$ git commit -m "First Commit"

-m ကတော့ message ကို ရည်ညွှန်းပြီးတော့ နောက်က “ “ မှာ ကိုယ်ပေးချင်တဲ့ message ကို ပေးနိုင်ပါတယ်။

## Git Status

Project ထဲက ဘယ်ဖိုင်တွေ ဘယ် state မှာရောက်နေလည်းဆိုတာ သိရဖို့ git status command ကိုအသုံးပြုရပါတယ်။

eg. \$ git status

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

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

လက်ရှိမှာတော့ git add မလုပ်ရသေးတဲ့အခြေအနေပဲဖြစ်ပါတယ်။ git add လုပ်ပြီးစမ်းကြည့်ရအောင်ပါ။

```
No commits yet

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

git add လုပ်ပီးနောက်မှာ home.html ဟာ အစိမ်းရောင်ပြောင်းသွားပီး modified အခြေအနေကိုရောက်ရှိနေပြီးဖြစ်ကြောင်း သိနိုင်ပါတယ်။ Git commit လုပ်ပြီး add လုပ်ထားတဲ့ဖိုင်ကို staging area ထဲကိုဆွဲပို့လို့ရသွားပါပြီ။

## Git Log

Git Log command ဖြင့် မိမိလိုအပ်သော commit တွေ ကို ကြည့်နိုင်ပါတယ်။

eg. \$git log

```
junemoe@junemoe:~/Desktop/Git Test$ git log
commit 1a7b69eda2e7829ebd0df1fec6bb0a403788484d (HEAD -> master)
Author: Juneict <junemoenyinyiict@gmail.com>
Date: Thu Sep 15 13:33:51 2022 +0630

    first commit
```

log တွင် branch,author,date တွေနဲ့ commit တွေကိုကြည့်နိုင်ပါတယ်။ထို့အပြင်

eg. \$ git log --oneline

```
junemoe@junemoe:~/Desktop/Git Test$ git log --oneline
1a7b69e (HEAD -> master) first commit
```

log မှာ commit တွေများလာတဲ့အခါ ခုနကလို git log နဲ့တင်ဆို အမြင်ရှုပ်လာနိုင်ပါတယ်။ oneline ဖြင့် အသုံးပြုသောကြောင့် ပိုမိုရှင်းလင်းအောင်မြင်ရပါတယ်။

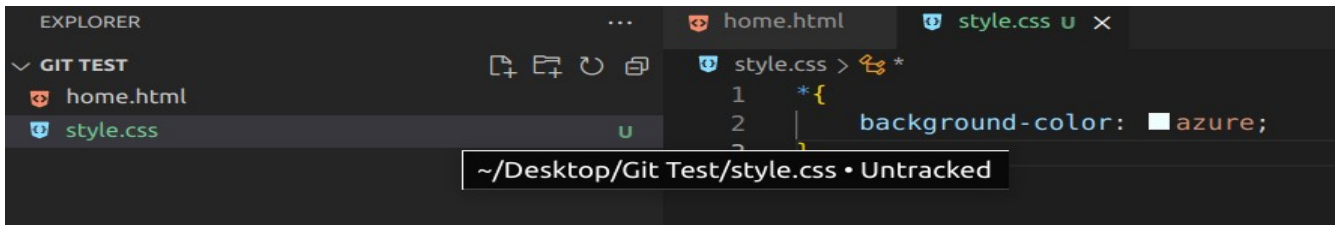
## Git Checkout

မိမိသွားလိုသော commit condition ကို checkout ကိုသုံးပီးသွားလို့ရပါတယ်။

eg. git checkout 1a7b69e(\*ကြည့်လိုသော commit ရဲ့ id ကိုကူးယူပီးထည့်ရပါတယ်)

Checkout ကို စမ်းကြည့်ရအောင်ပါ။

အပေါ်မှာ first commit condition တစ်ခုပြီးတဲ့ထိ လုပ်ပြီးလို့ Project ထဲမှာ နောက်ထပ် style.css နာမည်နဲ့ file တခုဆောက်လိုက်ပါမယ်။ style.css ထဲမှာ css တစ်ခုခုကိုကြိုက်ရာရေးထည့်ပါ။



ထို့နောက် terminal မှာ git status ဖြင့် condition ကိုစစ်ကြည့်ပါ။

```
• junemoe@junemoe:~/Desktop/Git Test$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    style.css
```

ခုနက ထည့်လိုက်တဲ့ css file ဟာ working directory ထဲမှာပဲရှိပါသေးတယ်။ git add . ကို သုံးပြီး modified လုပ်လိုက်ရအောင်ပါ။

```
• junemoe@junemoe:~/Desktop/Git Test$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   style.css
```

Git commit ကို သုံးပြီး staging area ထဲဆွဲခေါ်လိုက်ပါမယ်။

```
• junemoe@junemoe:~/Desktop/Git Test$ git commit -m "second commit"
[master 9c347d9] second commit
1 file changed, 3 insertions(+)
create mode 100644 style.css
```

second commit လုပ်ပြီးသွားရင် git log --oneline ဖြင့် log တွေကို စစ်ကြည့်ရအောင်ပါ။

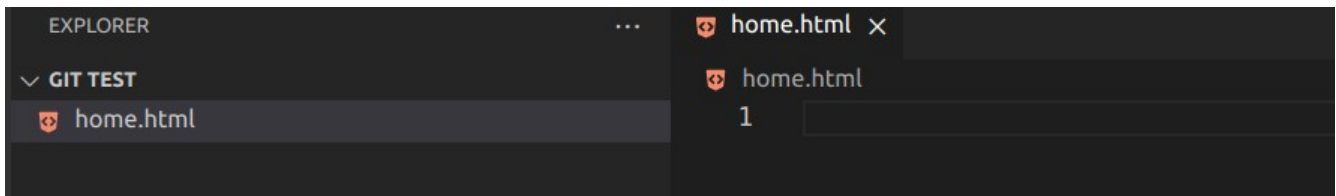
```
• junemoe@junemoe:~/Desktop/Git Test$ git log --oneline
9c347d9 (HEAD -> master) second commit
1a7b69e first commit
```

အခုဆိုရင် commit condition နှစ်ခုရပါပြီ။ Checkout command ဖြင့် first commit အခြေအနေကို ပြန်သွားကြည့်ရအောင်ပါ။

```
• junemoe@junemoe:~/Desktop/Git Test$ git checkout 1a7b69e
Note: switching to '1a7b69e'.
```

first commit ရဲ့ id ဖြစ်တဲ့ 1a7b69e ကို shift+ctrl+c ဖြင့်ကူးပြီး checkout နောက်မှာ shift+ctrl+v ဖြင့် paste လိုက်ပါတယ်။ ဒီနေရာမှာ သတိထားရမှာက terminal ထဲမှာ ctrl+c , ctrl+v နဲ့ သုံးမရပါဘူး။ terminal ထဲမှာ ctrl+c ဟာ server တွေကိုရပ်တဲ့နေရာမှာ သုံးပါတယ်။

first commit condition ကို ရောက်သွားပြီဖြစ်လို့ ခုနကဖန်တီးခဲ့တဲ့ second commit ထဲမှာရှိတဲ့ style.css file ကရှိနေမှာမဟုတ်တော့ပါဘူး။ အဲဒီနည်းဖြင့် ကျနော်တို့ဟာ ကိုယ်လိုအပ်တဲ့ condition ကို လွယ်ကူစွာသွားရောက်စစ်ဆေးနိုင်ပါတယ်။



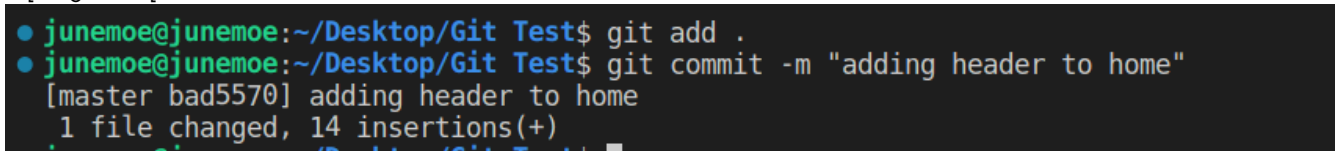
style.css ကိုတည်ဆောက်ခဲ့တဲ့ condition ကိုပြန်ရောက်ဖို့ဆိုရင် git checkout master ဖြင့်အလွယ်တကူ ရရှိနိုင်ပါတယ်။ထို့အပြင် branches တွေ ခွဲရာတွင်လည်း branch တစ်ခုကနေ နောက်တစ်ခုကိုပြောင်း သောအခါသုံးနိုင်ပါတယ်။

## Git Revert

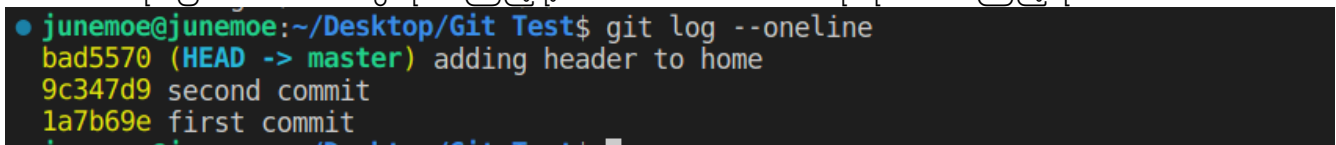
မိမိ project ထဲတွင် မလိုချင်တော့သော commit condition များကိုဖျက်သောအခါသုံးရပါတယ်။ လက်တွေ့ သုံးကြည့်ရအောင်။အထက်ပါဖော်ပြထားတဲ့ checkout မှာဆိုရင်လည်းမိမိလိုချင်တဲ့ commit condition ကိုရောက်အောင် သွားကြည့်လို့ရပါတယ်။ဒါပေမယ့် checkout မှာ ကြည့်ရုံပဲရပါတယ်။ ဒါကြောင့် ကိုယ်လိုချင်တဲ့ အခြေအနေထိပဲ ထားချင်ပြီး ပြင်ဆင်တာတွေလုပ်ချင်တဲ့အခါ revert ကိုသုံးပြီးလုပ်ရပါတယ်။



အရင်ဆုံး home.html မှာ header tag လေးထည့်ပီးစမ်းကြည့်ရအောင်ပါ။ထို့နောက် git add . ကိုသုံးပြီး ဖိုင်ကို add လိုက်ပါမယ်။အထက်ပါနည်းတွေအတိုင်း git add . လုပ်ပြီး git commit သုံးပြီး staging area ကို ဆွဲခေါ်လိုက်ပါမယ်။



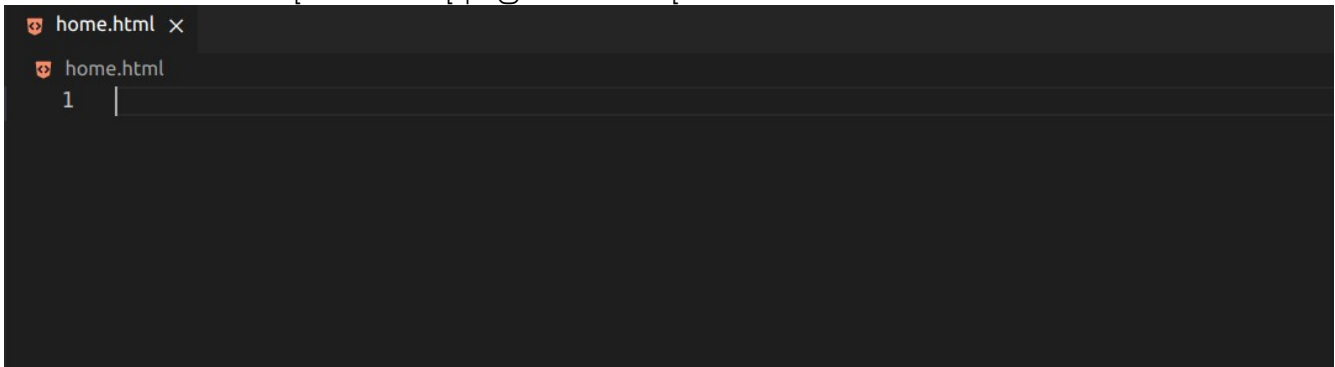
commit လုပ်ပြီးပါက log တွေကိုစစ်ကြည့်ဖို့ git log --oneline ကိုသုံးပြီး စစ်ကြည့်ရအောင်ပါ။



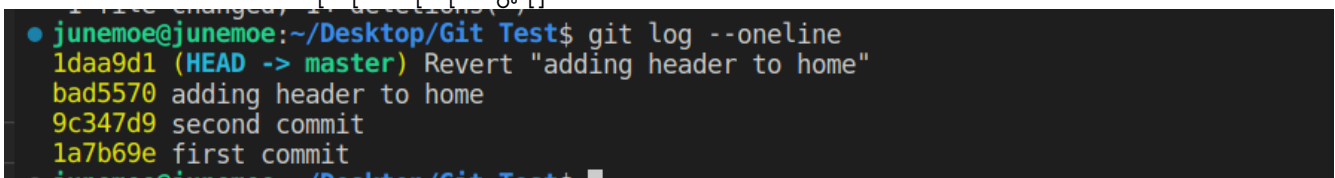
နောက်ထပ် commit condition တစ်ခုထပ်တိုးလာတာကိုတွေ့ရမှာပါ။အခုတော့ git revert ကိုသုံးပြီး နောက်ဆုံး home.html မှာ header မထည့်ရသေးတဲ့အခြေအနေကို ပြန်သွားကြည့်ရအောင်။

\$ git revert bad5570

checkout မှာအတိုင်း မိမိဖျက်မယ့် commit condition ရဲ့ id ကို revert နောက်မှာထည့်လိုက်ရုံပါပဲ။  
confirm box ပေါ်လာရင် ctrl+x ကိုနှိပ်ပြီး yes ပေးလိုက်ပါမယ်။



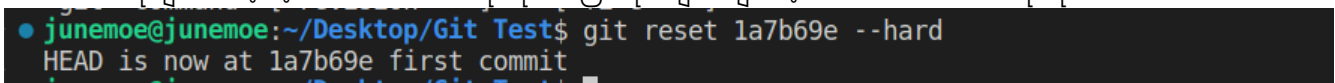
ပုံမှာတွေ့တဲ့အတိုင်းပဲ header မှာ ဘာမှမဖြည့်ထားတဲ့ commit condition ကိုပြန်ပြောင်းသွားပြီး ထိုနောက်မှ ပြင်ဆင်ထားသောအချက်များကို ဖျက်လိုက်ပါတယ်။ဒီမှာ မှတ်ထားရမှာက revert လုပ်ပီးတဲ့အချိန်မှာ revert လုပ်လိုက်တဲ့ အခြေအနေကို commit လုပ်လိုက်တဲ့အတွက် git log စစ်ကြည့်တဲ့အခါ revert လုပ်ထားတဲ့ commit condition တစ်ခုကို အခုလိုတွေ့ရမှာပါ။



ဖျက်လိုက်တယ်ဆိုပေမယ့် revert commit လုပ်ပီးသိမ်းထားလို့ လိုအပ်လာတဲ့အခါ ထိုအခြေအနေကို ပြန်လည်ရယူနိုင်ပါတယ်။

## Git Reset

မလိုချင်တော့တဲ့ commit တစ်ခုကို အပြီးတိုင်ဖျက်ချင်တဲ့အခါ git reset ကိုသုံးပါတယ်။



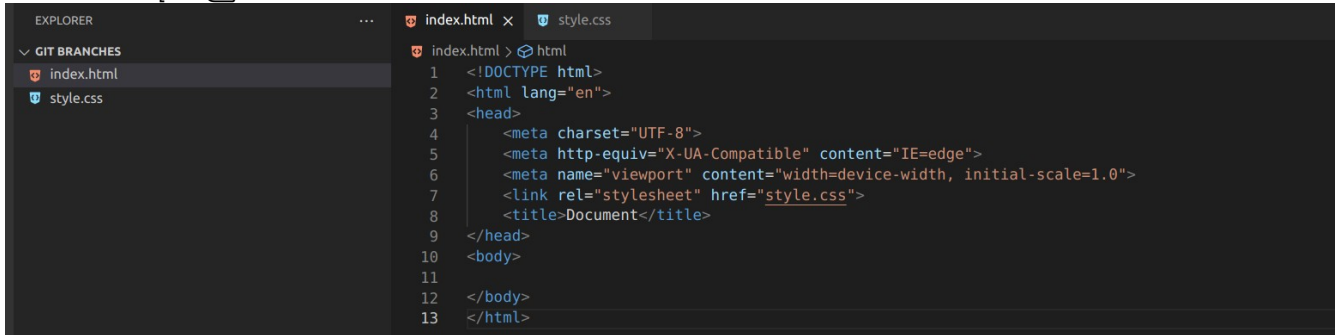
reset နောက်မှာ လိုအပ်တဲ့ commit ရဲ့ id ကို ထည့်ပေးပြီး --hard command ဖြင့်ဖျက်လိုက်ပါတယ်။  
reset သည် hard delete ဖြစ်တဲ့အတွက်ကြောင့် မလိုတော့ကြောင်းသေချာမှသာ အသုံးပြုသင့်ပါတယ်။

## Git Branches

git မှာ အမှိုက်ဆုံး function တွေထဲကမှ branches ဟာလည်း အသုံးအများဆုံး function တစ်ခုပဲ ဖြစ်ပါတယ်။Project repository တစ်ခုစတင်လိုက်တာ နဲ့ master/main branch အနေနဲ့ default ပါဝင် လာပါတယ်။Main branch မှာ အမှန်တကယ်အလုပ်လုပ်နေတဲ့ project function တွေနဲ့ production version အနေနဲ့ပဲထားသင့်ပါတယ်။ ဒါ့ကြောင့် လက်ရှိရေးနေတဲ့ code တွေနဲ့ မတူညီတဲ့ developer တွေ ပေါင်းရေးရတဲ့အခါ branches တွေခွဲပီးရေးဖို့လိုလာပါတယ်။Project တစ်ခုတည်းမှာ developer 2 ယောက် ပေါင်းပီးရေးရတဲ့ ပုံစံမျိုးနဲ့ စမ်းရေးကြည့်ရအောင်ပါ။



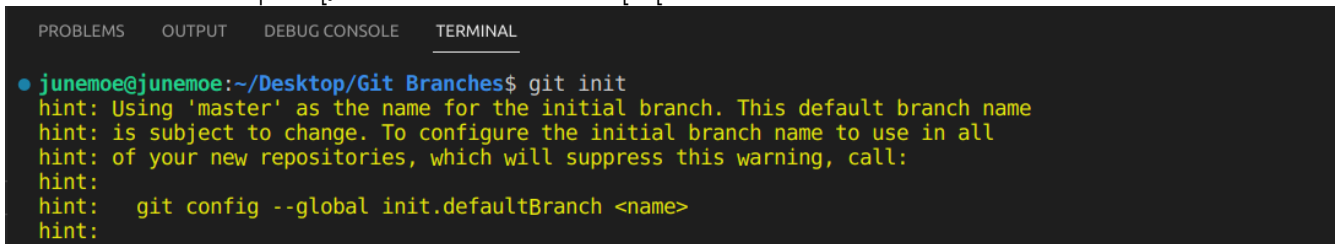
ပထမဆုံးအနေနဲ့ new folder ဆောက်ပြီး project folder တစ်ခုဆောက်ကြည့်ရအောင်ပါ။  
ထို့နောက် project directory ထဲဝင်ပြီး vs code နဲ့ run ပါမယ်။ project folder ထဲမှာ index.html file နှင့် style.css ကိုတည်ဆောက်ပါမယ်။



```
EXPLORER
  GIT BRANCHES
    index.html
    style.css

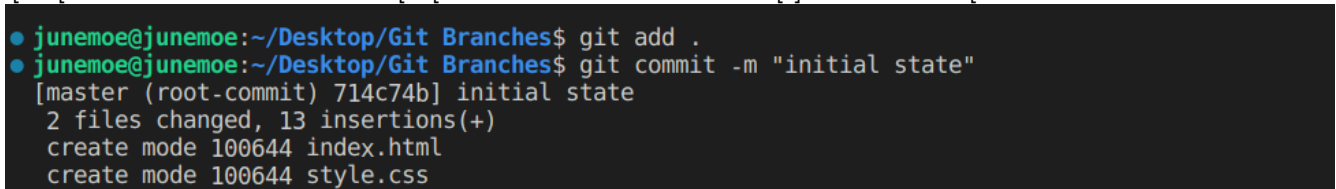
index.html x style.css
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="style.css">
8   <title>Document</title>
9 </head>
10 <body>
11
12 </body>
13 </html>
```

index.html file ထဲမှာ html boilerplate ကိုရေးထားလိုက်ပါမယ်။ ထို့နောက် terminal ကိုဖွင့်ပြီး git repository စတင်ဖန်တီးဖို့ git init command ကိုသုံးပါမယ်။



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
• junemoe@junemoe:~/Desktop/Git Branches$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
```

git repository တစ်ခုစလှိုက်ပြီးဆိုတာနဲ့ default Branch ကို master branch အဖြစ် သတ်မှတ်ပြီးသားပဲ ဖြစ်ပါတယ်။ git add . ဖြင့် စတင်ရေးသားထားသောဖိုင်များကို add ထားလိုက်ပါ။ ထို့နောက် commit တစ်ခု ရေးသားပြီး initial state တစ်ခုကို commit condition တစ်ခုမှတ်သားထားရအောင်ပါ။



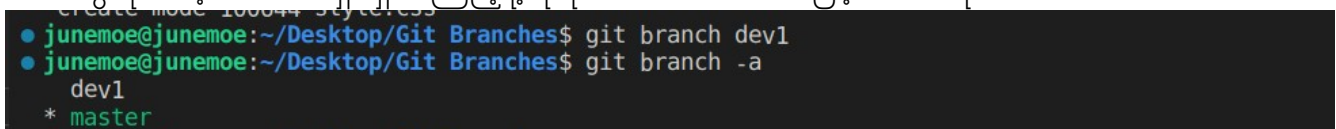
```
• junemoe@junemoe:~/Desktop/Git Branches$ git add .
• junemoe@junemoe:~/Desktop/Git Branches$ git commit -m "initial state"
[master (root-commit) 714c74b] initial state
2 files changed, 13 insertions(+)
create mode 100644 index.html
create mode 100644 style.css
```

project တစ်ခုရဲ့ stable state ကနေ developer တစ်ယောက်က စတင်ရေးသားတော့မယ်ဆိုရင် branch တစ်ခုကို ခွဲပြီးရေးသားဖို့လိုလာပါပီ။ developer 1 အနေ ရေးသားထားတဲ့ ကုဒ်တွေကို စီမံဖို့ branch တစ်ခု ခွဲကြည့်ရအောင်ပါ။

\$git branch dev1

ဆိုတဲ့ command ဖြင့် မိမိခွဲလိုသော branch နာမည်ကို git branch နောက်က ထည့်ပေးလိုက်ရုံပါပဲ။

အသစ်ခွဲလိုက်တဲ့ branch ရှိမရှိစစ်ကြည့်ဖို့ဆိုရင် git branch -a ဖြင့်စမ်းသပ်နိုင်ပါတယ်။



```
• junemoe@junemoe:~/Desktop/Git Branches$ git branch -a
• junemoe@junemoe:~/Desktop/Git Branches$ git branch -a
dev1
* master
```

ခုနက ဖန်တီးလိုက်တဲ့ dev1 ဆိုတဲ့ branch ကို master branch ပေါ်မှာတွေ့ရပါလိမ့်မယ်။ \* ကိုကြည့်ပြီး လက်ရှိရောက်နေတဲ့ branch ကို သိနိုင်ပါတယ်။ developer အတွက် branch ကို ခွဲပြီးပီဆိုတော့ အဲ့ branch ကို ရောက်အောင် git checkout ကို သုံးနိုင်ပါတယ်။ စမ်းကြည့်ရအောင်ပါ။



\$ git checkout dev1(\*သွားလိုသော branch name ကိုရိုက်ရပါမယ်)

```
• junemoe@junemoe:~/Desktop/Git Branches$ git checkout dev1
Switched to branch 'dev1'
• junemoe@junemoe:~/Desktop/Git Branches$ git branch -a
* dev1
  master
```

တစ်ခါတည်း git branch -a ဖြစ်လက်ရှိရောက်နေတဲ့ branch ကို စစ်ကြည့်ရင်လည်း \* က dev1 မှာပြနေ ပီဖြစ်ပါတယ်။ဆိုလိုတာက လက်ရှိ dev1 branch မှာပဲ အသစ်ရေးရမယ့် code တွေကို စမ်းသပ်ရေးရမှာပဲ ဖြစ်ပါတယ်။

```
index.html > html > body > header
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="style.css">
8   <title>Document</title>
9 </head>
10 <body>
11   <!-- developer1 -->
12   <header>
13     <h1>Project Titel</h1>
14   </header>
15 </body>
16 </html>
```

body tag မှာ developer 1 အနေနဲ့ header ကိုရေးပီး စမ်းကြည့်ရအောင်ပါ။

```
• junemoe@junemoe:~/Desktop/Git Branches$ git add .
• junemoe@junemoe:~/Desktop/Git Branches$ git commit -m "header add by dev1"
[dev1 ede4de3] header add by dev1
1 file changed, 4 insertions(+), 1 deletion(-)
```

ထုံးစံအတိုင်း ထပ်တိုးရေးသားထားသည့် changes တိုင်း မှတ်သားနိုင်ရန် git add . ကိုအသုံးပြုပြီး git commit လုပ်ပေးရပါတယ်။ Developer 1 ရေးသားထားတဲ့ code တွေ stable ဖြစ်ပီ master branch ထဲ ကိုပြောင်းချင်ပီဆိုရင် merge ပြုလုပ်ရပါတယ်။ Merge မလုပ်ခင် master branch ကိုပြန်ပြောင်းပြီးမှ လုပ် လို့ရပါမယ်။ \$ git checkout master ဖြင့် master branch ကိုပြောင်းလိုက်ပါမယ်။ Master branch ကို ပြောင်းပီးတဲ့အခါ developer 1 ရေးထားသော header ကိုမြင်ရအုန်းမှာမဟုတ်ပါဘူး။ ထို့နောက် merge လုပ်ပြီး စမ်းကြည့်ရအောင်ပါ။

\$git merge dev1(\*ပေါင်းလိုသော branch name)

```
• junemoe@junemoe:~/Desktop/Git Branches$ git merge dev1
Updating 714c74b..ede4de3
Fast-forward
 index.html | 5 ++++-
 1 file changed, 4 insertions(+), 1 deletion(-)
```

ဒါဆိုရင် master branch ထဲမှာလည်း developer 1 ရေးထားတဲ့ code တွေပေါင်းထည့်ပြီးဖြစ်ပါတယ်။

ထိုနည်းအတိုင်းပဲ developer 2 အနေနဲ့ ကိုယ်တိုင်စမ်းကြည့်ဖို့ တိုက်တွန်းပါတယ်။ Branch တွေကိုပြန်ဖျက် ချင်တဲ့အခါ \$ git branch -D dev1(\*ဖျက်လိုသော branch name)ဖြင့်ဖျက်လိုရပါတယ်

## Merge Conflict

Project တစ်ခုတည်းမှာ developer 2 ယောက်ပေါင်းရေးထားတာတွေကို master branch တစ်ခုတည်းကို ပြန်ပေါင်းတဲ့အခါ 1 file တည်းကို 2 ယောက်လုံးပြင်ထားမိပြီး conflict ဖြစ်တာမျိုးတွေလည်းဖြစ်တက်ပါတယ်။ထိုကဲ့သို့ဖြစ်လာနိုင်တဲ့ error မျိုးကို စမ်းကြည့်ရအောင်ပါ။အထက်မှာ dev 1 branch ခွဲထားပြီးဖြစ်လို့ dev 2 branch ထပ်ခွဲပီးစမ်းကြည့်ပါမယ်။ dev 2 branch က index.html မှာ section တစ်ခုထက်တိုးရေးထားလိုက်ပါမယ်။

```
index.html > html > body > section
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="style.css">
8      <title>Document</title>
9  </head>
10 <body>
11     <!-- developer1 -->
12     <header>
13         <h1>Project Titel</h1>
14     </header>
15     <!-- developer2 -->
16     <section>
17         <p>
18             Lorem, ipsum dolor sit amet consectetur adipisicing elit. Laboriosam a quisquam, iste praesentium, re
19         </p>
20     </section>
21 </body>
22 </html>
```

ထို့နောက် git add . လုပ်ပီး commit ပြုလုပ်ပေးရပါမယ်။ dev2 မှာ ပြင်သလိုပဲ dev1 ကို checkout သွားပီး section တစ်ခုထပ်ထည့်ကြည့်ပါမယ်။

```
index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="style.css">
8      <title>Document</title>
9  </head>
10 <body>
11     <!-- developer1 -->
12     <header>
13         <h1>Project Titel</h1>
14     </header>
15 </body>
16 </html>
```

```
junemoe@junemoe:~/Desktop/Git Branches$ git add .
junemoe@junemoe:~/Desktop/Git Branches$ git commit -m "section from dev2"
[dev2 1fe941a] section from dev2
1 file changed, 6 insertions(+)
junemoe@junemoe:~/Desktop/Git Branches$ git checkout dev1
Switched to branch 'dev1'
junemoe@junemoe:~/Desktop/Git Branches$
```

dev1 branch ကိုပြန်ရောက်သွားတဲ့အခါမှာ dev2 branch ထဲမှာ ရေးထားတဲ့ section ပေါ်မှာမဟုတ်ပါဘူး dev1 ရဲ့ branch ထဲမှာလည်း section tag တစ်ခုထပ်ရေးပါမယ်။

```
index.html > html > body > section > p
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="style.css">
8   <title>Document</title>
9 </head>
10 <body>
11   <!-- developer1 -->
12   <header>
13     <h1>Project Titel</h1>
14   </header>
15   <section>
16     <p>i am from dev1 paragraph</p>
17   </section>
18 </body>
19 </html>
```

git add. နဲ့ commit လုပ်လိုက်ပါ။ထို့နောက် merge လုပ်ရန် master branch ကို checkout လုပ်ရပါမယ်။

```
junemoe@junemoe:~/Desktop/Git Branches$ git checkout master
Switched to branch 'master'
junemoe@junemoe:~/Desktop/Git Branches$ git branch -a
dev1
dev2
* master
junemoe@junemoe:~/Desktop/Git Branches$
```

master branch ကိုရောက်တာသေချာပီဆိုရင် စပြီး dev2 ကရေးထားတဲ့ code ကို merge လုပ်ကြည့်ပါမယ်။

```
index.html | style.css
index.html > html > body > ?
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="style.css">
8   <title>Document</title>
9 </head>
10 <body>
11   <!-- developer1 -->
12   <header>
13     <h1>Project Titel</h1>
14   </header>
15 <<<<<<< HEAD (Current Change)
16 =====
17 <section>
18   Lorem ipsum dolor sit amet consectetur adipisicing elit. Iusto sed sapiente quam dolore natus voluptatum al
19 </section>
20 >>>>>>> dev1 (Incoming Change)
21 </body>
22 </html>
```

အဲ့အချိန်မှာ conflict error စဖြစ်ပါပီ။Accept Current Change ဆိုရင် နုကိုအတိုင်းရှိနေမှာဖြစ်ပြီး Accept Incoming Change ဆိုရင် dev2 ကပြောင်းလိုက်တဲ့ အခြေအနေကိုလက်ခံမှာဖြစ်ပါတယ်။Accept Both Changes ဆိုရင်တော့ လက်ရှိအခြေအနေကော dev2 ကဝင်လာတဲ့ အခြေအနေကောကို လက်ခံမှာဖြစ်ပါတယ်။ကိုယ့် project ရဲ့ လိုအပ်ချက်အပေါ်မူတည်ပြီး အပေါ်က အချက်တွေကို သေချာရွေးပြီးမှ လက်ခံရမှာပါ။ဒီလို conflict မျိုးဟာ developer 2 ဦးထက်ပိုပြီး developed လုပ်တဲ့အခါကြုံရလေ့ရှိတဲ့ conflict ပဲဖြစ်ပါတယ်။

အခုဆိုရင် git နဲ့ပတ်သတ်တဲ့ basic ပိုင်းတွေစုံသလောက်တွေရှိသွားပါပြီ။အခုလုပ်ခဲ့တာတွေအားလုံးဟာ local repository တစ်ခုအတွက်ဖြစ်လို့ ဒီ function တွေအားလုံးကို internet မလိုဘဲသုံးလို့ရနေမှာဖြစ်ပါတယ်။နောက်chapter မှာတော့ internet ကိုအသုံးပြုပြီး github server နဲ့ ချိတ်ဆက်ပီးသုံးပုံတွေကိုလေ့လာသွားမှာဖြစ်ပါတယ်။

## Summary

အထက်မှာလေ့လာခဲ့တဲ့အတိုင်း git add လုပ်ပုံလုပ်နည်းတွေ git commit လုပ်ရတဲ့နည်းတွေဟာ git init စပြီးတာနဲ့ changes တစ်ခုကို commit လုပ်ထားခြင်းတိုင်း git add . နှင့် git commit ပြုလုပ်ရတာကိုမှတ်သားထားရန်လိုအပ်ပါတယ်။Git branch မှာဆိုရင်လည်း checkout မထွက်ခင်တိုင်း changes တစ်ခုတစ်ခုရှိရင် commit ပြုလုပ်ပြီးမှ ထွက်ရန်လိုအပ်ပါတယ်။ master branch သည် default branch ဖြစ်ပြီး production version နှင့် stable ဖြစ်သော code များကိုသာ master branch မှာထားသင့်ပါတယ်။ Git Reset သည် hard delete ဖြစ်တဲ့အတွက်ကြောင့် ဖျက်ရန်သေချာမှသာ သုံးရန်လိုအပ်ပါတယ်။ Branch တွေကိုဖျက်တဲ့အခါ master branch ကိုသွားပြီးမှ ဖျက်လို့ရပါမယ်။ Branch merge လုပ်တဲ့အခါလည်း master branch မှာပဲ merge လုပ်ရမှာကိုသတိထားရပါမယ်။

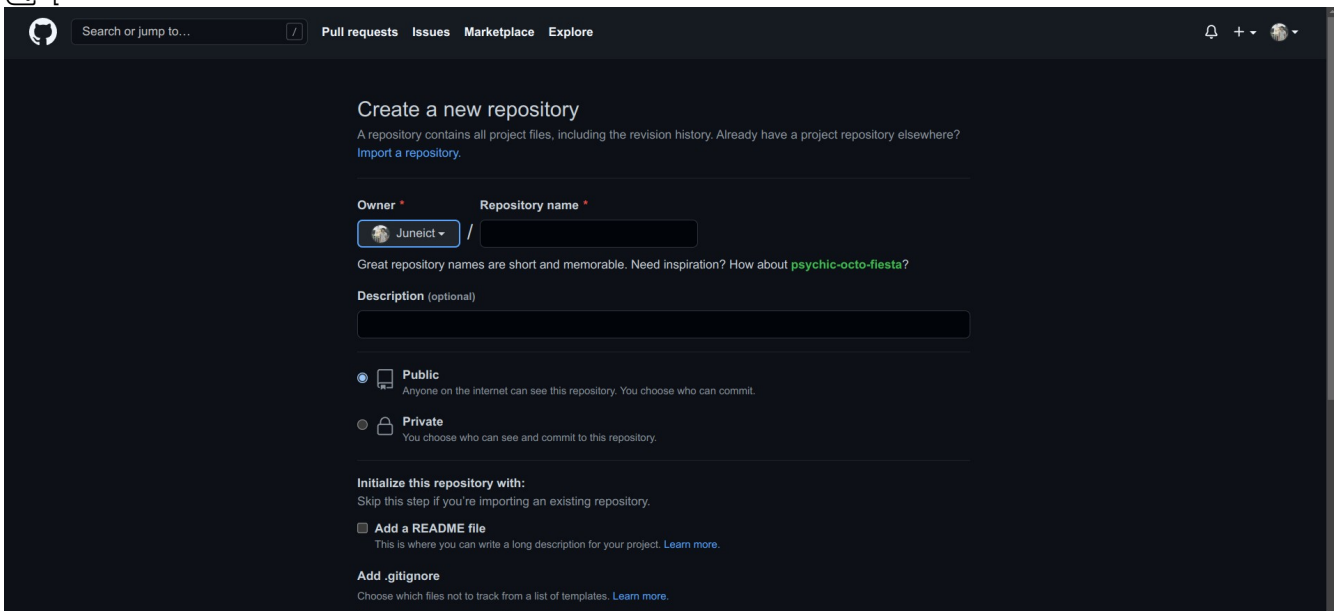
# Chapter 5

## Using Git with GitHub

### Creating a new repository on github

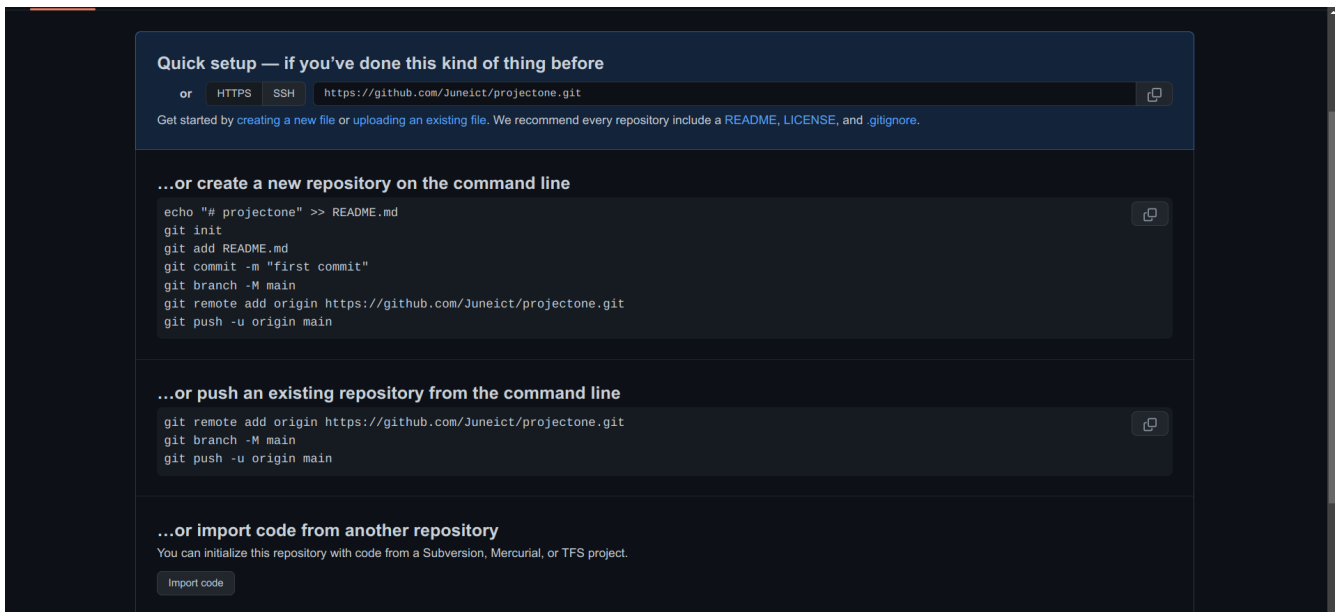
Project တစ်ခုကို developer တစ်ယောက်ထက်ပိုပြီး management လုပ်တဲ့အခါဖြစ်စေ developer မိမိ project ကနေ နောက် device တစ်ခုကိုပြောင်းပြီး developed လုပ်တဲ့အခါဖြစ်စေ online server တစ်ခုကို လိုအပ်ပါတယ်။ ဒါကြောင့် github server နဲ့ချိတ်ဆက်တက်အောင်လေ့လာထားဖို့လိုပါလိမ့်မယ်။ github ဟာ free server တစ်ခုဖြစ်ပါတယ်။ Project တစ်ခုကို github server ပေါ်တင်ပြီး လက်တွေ့စမ်းသပ်ကြည့်ရအောင်ပါ။

အရင်ဆုံး github.com ကိုသွားပြီး အကောင့်ဖွင့်ရပါမယ်။ Sign up ကိုဝင်ပြီး လိုအပ်သည်များကိုဖြည့်ပြီး အဆင့်လိုက်သွားရပါမယ်။ Github အကောင့်ဖွင့်ပြီးရင် login ဝင်ပြီး repository တစ်ခုကို ဖန်တီးကြည့်ရအောင်။



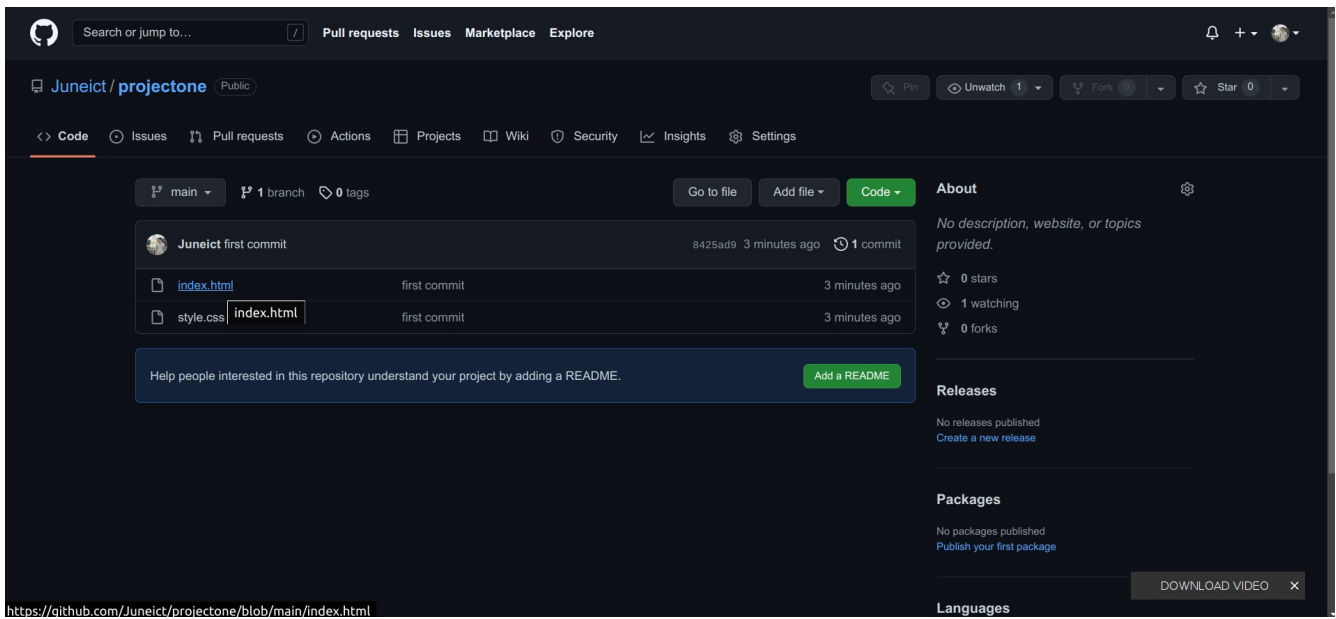
မိမိ လုပ်လိုသော repository နာမည်ကိုဖြည့်ပြီး repository တစ်ခုကိုဖန်တီးလို့ရပါတယ်။ဒီနေရာမှာ repository name ကို ကြိုက်နှစ်သက်ရာပေးနိုင်ပါတယ်။project ရဲ့ နာမည်နဲ့တူနေပါက ပိုအဆင်ပြေပါတယ်။

Description နေရာမှာ မိမိလုပ်မဲ့ project ရဲ့အကြောင်းအရာတွေကို ဖြည့်နိုင်ပါတယ်။ Public ကိုတော့ project ကို open source ပေးချင်တဲ့အခါဖြစ်စေ အများဝင်ကြည့်နိုင်အောင်ဖြစ်စေ ရွေးပေးလို့ရပါတယ်။ Private ကို ရွေးပြီး မိမိတစ်ယောက်တည်းပဲမြင်အောင်ထားနိုင်ပါတယ်။များသောအားဖြင့်တော့ private နဲ့ပဲ သုံးကြပါတယ်။ Add a Readme file ကို project ရဲ့ documentation တွေ installation တွေနဲ့ ကိုယ်နှစ်သက်ရာဖြည့်နိုင်ဖို့ mark ပေးပြီးရွေးနိုင်ပါတယ်။ထိုနောက် create repository ကိုနှိပ်လိုက်ပါက repository တစ်ခုကိုဖန်တီးပြီးဖြစ်ပါတယ်။



Repository တစ်ခုကိုဖန်တီးပီးရင် local မှာ လုပ်ထားတဲ့ project နဲ့ ချိတ်ဆက်ကြည့်ရအောင်ပါ။အရင်ဆုံး local မှာ project တစ်ခုကိုဖန်တီးပါမယ်။ ကြိုက်နှစ်သက်ရာ နာမည်နဲ့ project folder တစ်ခုဆောက်ပါ။ repository မှာပေးထားတဲ့ နာမည်နဲ့တူရင်ပိုကောင်းပါတယ်။ထို project folder ထဲမှာ index.html ဖိုင်နှင့် style.css ကိုတည်ဆောက်လိုက်ပါမယ်။VS code နဲ့ဖွင့်ပြီး vs code က terminal ဖြင့်စမ်းသပ်ကြည့်ရအောင်ပါ။ local မှာ git စမ်းတုန်းကအတိုင်း project မှာ git စတင်မယ်ဆိုရင် git init ဖြင့်စတင်ရပါမယ်။ git init ဖြင့် git ကိုစတင်ပီးသွားရင် github ပေါ်တင်ကြည့်ကြပါမယ်။ အပေါ်ပုံမှာ ပြထားတဲ့အတိုင်း git repository တစ်ခုတည်ဆောက်လိုက်ပါဆိုတာနဲ့ လုပ်ရမယ့် နည်းတွေတစ်ခါတည်းပါလာပါတယ်။ အဲ့ထဲကမှာ အခုလက်ရှိ မိမိ project ကို ပြင်ဆင်ရေးသားထားတာမရှိသေးတဲ့အတွက် github server ပေါ်အရင်ဆုံး ချိတ်ကြည့်ပါမယ်။ git remote add origin <https://github.com/Juneict/projectone.git> ဖြင့် git server ကိုချိတ်ဆက်လိုက်ပါမယ်။ အဲ့ဒီမှာ origin ဆိုတာက နောက်က url ဖြစ်တဲ့ မိမိ project repository ကိုရည်ညွှန်းတာဖြစ်ပါတယ်။နောက် command တွေသုံးတဲ့ အခါ url ကိုရိုတ်မနေတော့ပဲ origin ကိုသုံးပီး တင်လို့ရပါတယ်။ထိုနောက် branch ကို main branch အဖြစ်ထားလိုက်ပါမယ်။git push u origin main ဖြင့် တင်လိုက်ပါမယ်။ ဒီနေရာမှာ နောက်က url အစား origin ကိုသုံးပီးပဲ repository link ရှိရာကို ညွှန်လို့ရတာတွေ့ရမှာပါ။ main ဆိုတာကတော့ ခုနကတည်ဆောက်ထားတဲ့ branch ကိုရည်ညွှန်းတာဖြစ်ပါတယ်။ main branch မှာ push လုပ်မယ်ဆိုပီးပေးလိုက်တာပါ။push လုပ်ပြီးသွားရင် github မှာ page ကို refresh လုပ်လိုက်ပါက ဒီပုံအတိုင်း ချိတ်ဆက်ပြီး အောင်မြင်ပီးဖြစ်ပါတယ်။



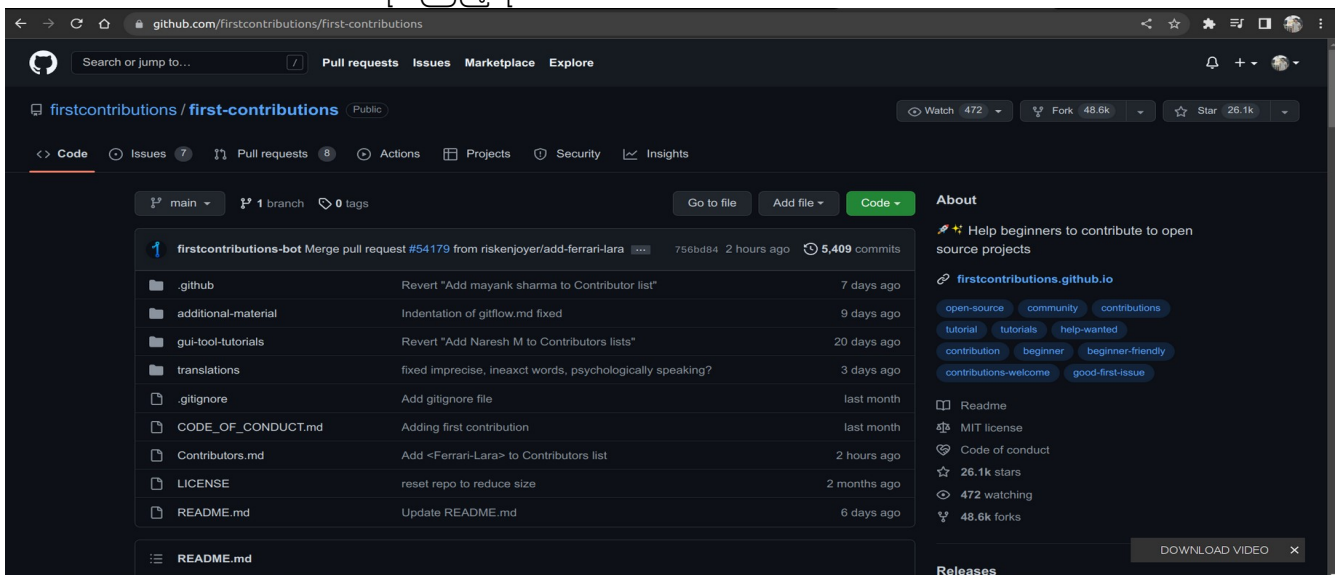


ထိုနောက်မှာတော့ chapter 4 မှာဖော်ပြခဲ့တဲ့အတိုင်း မိမိပြင်ဆင်ပီးသည့်များကို git add ပြုလုပ်ပီး git commit ပေးတာ၊ ပြင်ဆင်ပီးသည့်များကို github ပေါ်ပြန်တင်သောအခါ git push origin master ဖြင့် ပြန်တင်ပီး ကိုယ်တိုင်စမ်းသပ်လေ့လာနိုင်ပါတယ်။ Branches တွေခွဲပီးလည်း စမ်းကြည့်သင့်ပါတယ်။

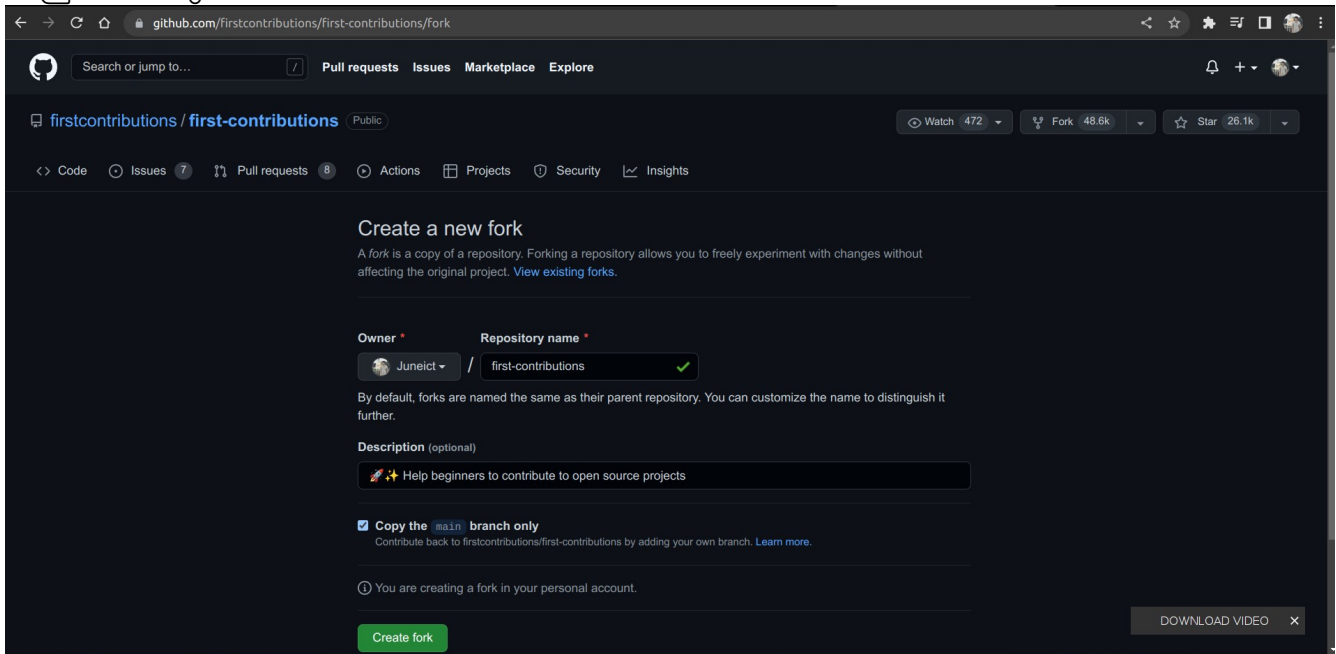
## Cloning A Git Repository Developed by other Developer

တခြား developer တစ်ယောက်က ရေးထားတဲ့ project ကို ပြန်ယူရေးသားတဲ့အခါဖြစ်စေ collaborate လုပ်ပီးရေးသားတဲ့အခါဖြစ်စေ repository တစ်ခုကို clone လုပ်ဖို့လိုလာပါ။ စပီးလေ့လာကြည့်ရအောင်ပါ။

<https://github.com/firstcontributions/first-contributions> ဒီလင့်ကနေ repository တစ်ခုက clone ပီး collaborate ပါဝင်လုပ်ကြည့်ရအောင်ပါ။



အရင်ဆုံး fork လုပ်ရပါမယ်။ fork လုပ်လိုက်တာနဲ့ မိမိအကောင့်မှာ နာမည်တူ repository တစ်ခု တည်ဆောက်သွားပါလိမ့်မယ်။

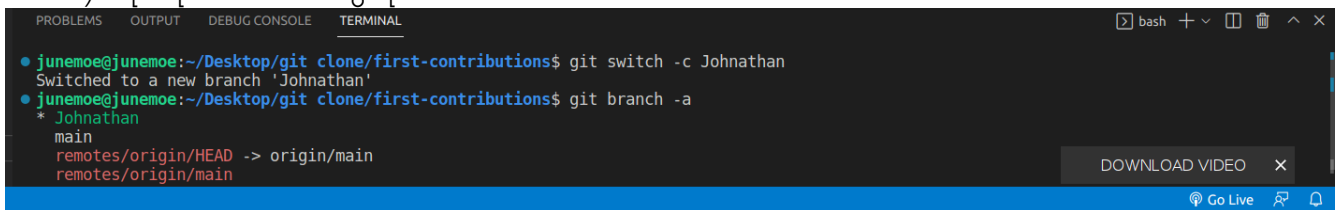


ထိုနောက် မိမိ repository ကို local ထဲကို စတင် clone ကြည့်ရအောင်ပါ။

git clone <https://github.com/Juneict/first-contributions.git>(\* git clone နောက်မှာ မိမိproject repository url ကို copy ယူပါ)

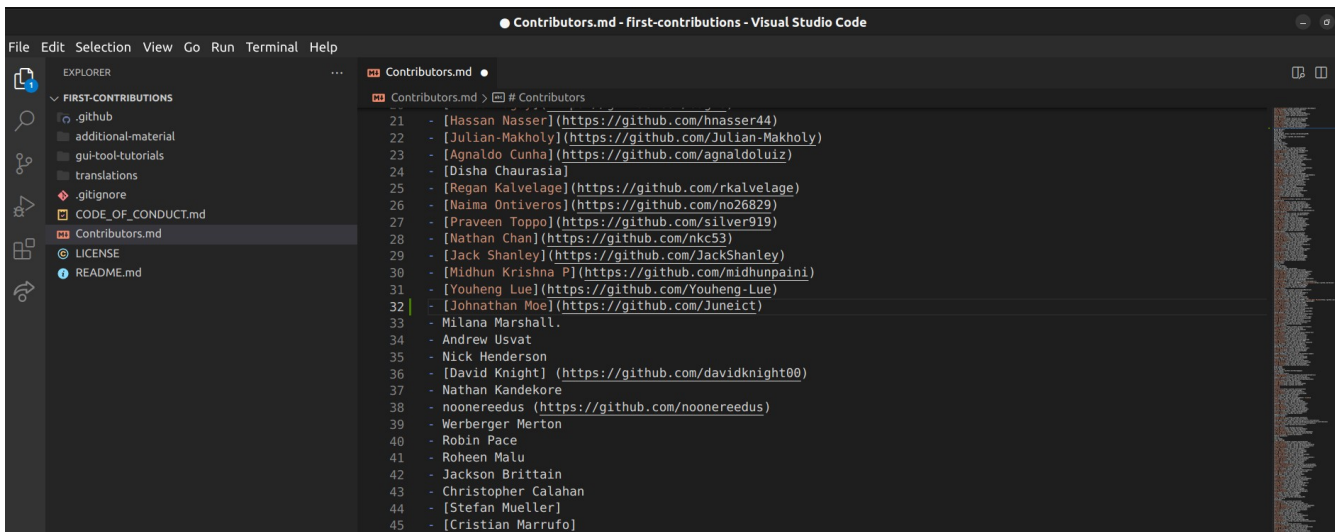
ကြိုက်နှစ်သက်ရာလမ်းကြောင်းပေါ်(eg. Desktop , Document ,etc)တွင် Terminal ဖွင့်ပြီး git clone (copy ယူထားတဲ့ repository url )ကိုရိုက်ပါ။ထိုလမ်းကြောင်းပေါ်တွင် project folder တည်ထောက်ပြီးသားဖြစ်နေပါလိမ့်မယ်။ ထို folder ထဲတွင် clone ယူထားသော project ထဲမှာ code များလည်းပါပြီးဖြစ်ပါတယ်။ထိုနည်းဖြင့် ကျွန်တော်တို့ တခြား developer များရဲ့ project ကို clone ရှိရနိုင်ပါတယ်။

ထို့အပြင် ထို project ထဲတွင် ကျွန်တော်တို့ ထပ်တိုးရေးသားထားသော code များကိုလည်း collaborate လုပ်ပြီး ရေးနိုင်ပါတယ်။collaborate လုပ်ရန် အရင်ဆုံး local တွင် clone ထားသော project ကို vs code ဖွင့်ပါ။ထိုနောက် terminal ဖွင့်ပြီး git switch -c Johnathan(\* မိမိထားလိုသော branch name) ကို သုံးပြီး branch ခွဲလိုက်ပါ။

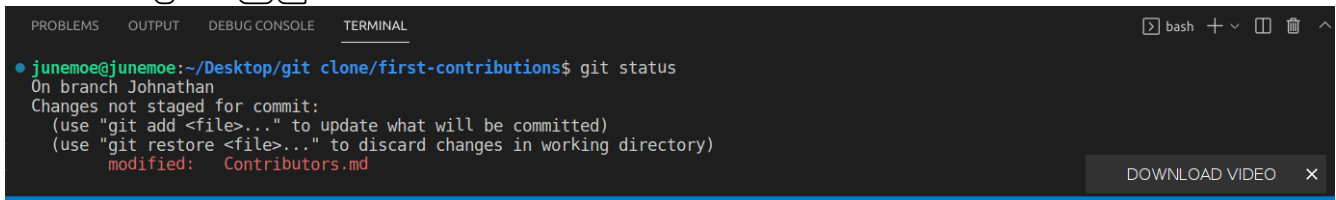


မိမိ branch ကိုရောက်ပါဆိုရင် project ဖိုင်ထဲက Contributors.md ဖိုင်ထဲကိုဝင်ပြီး contributor အနေ တဲ့ ထည့်သွင်းကြည့်ရအောင်ပါ။



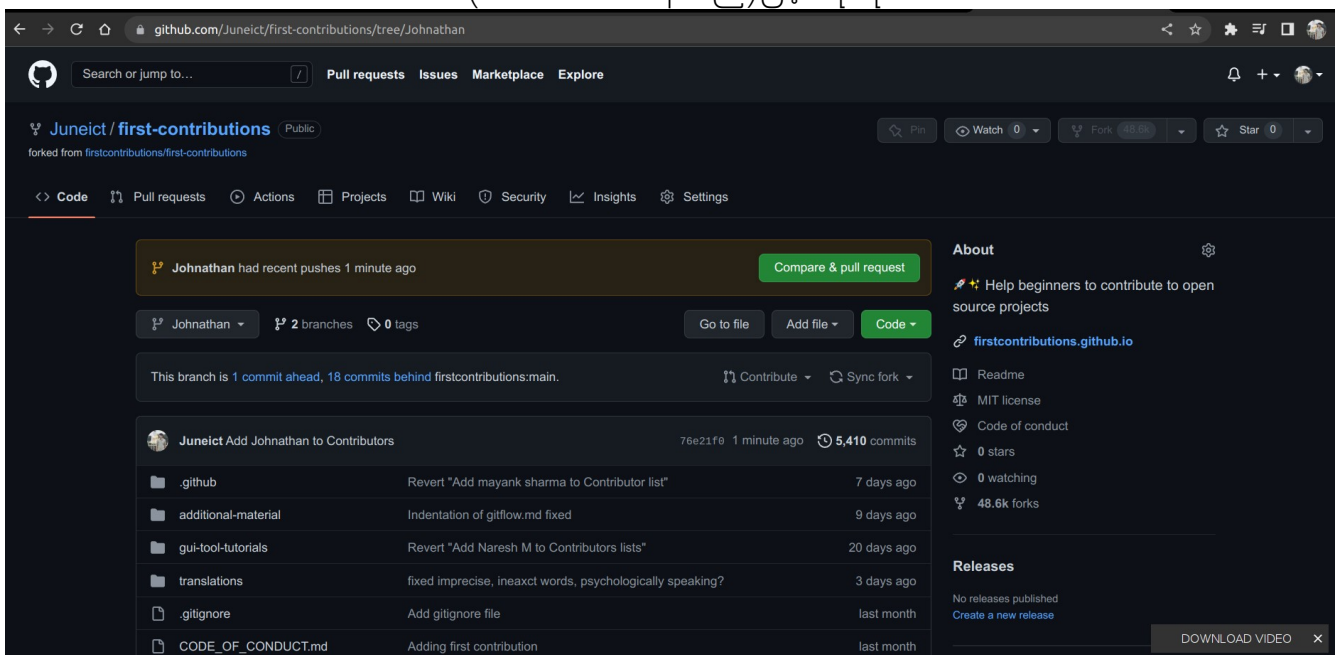


မိုင်ထဲမှာ မိမိကြိုက်နှစ်သက်ရာနေရာတွင် မိမိနာမည်နှင့် Github profile link ကို ကူးထည့်ကြည့်ပါ။ထို့နောက် git status ဖြင့်စစ်ကြည့်ပါ။



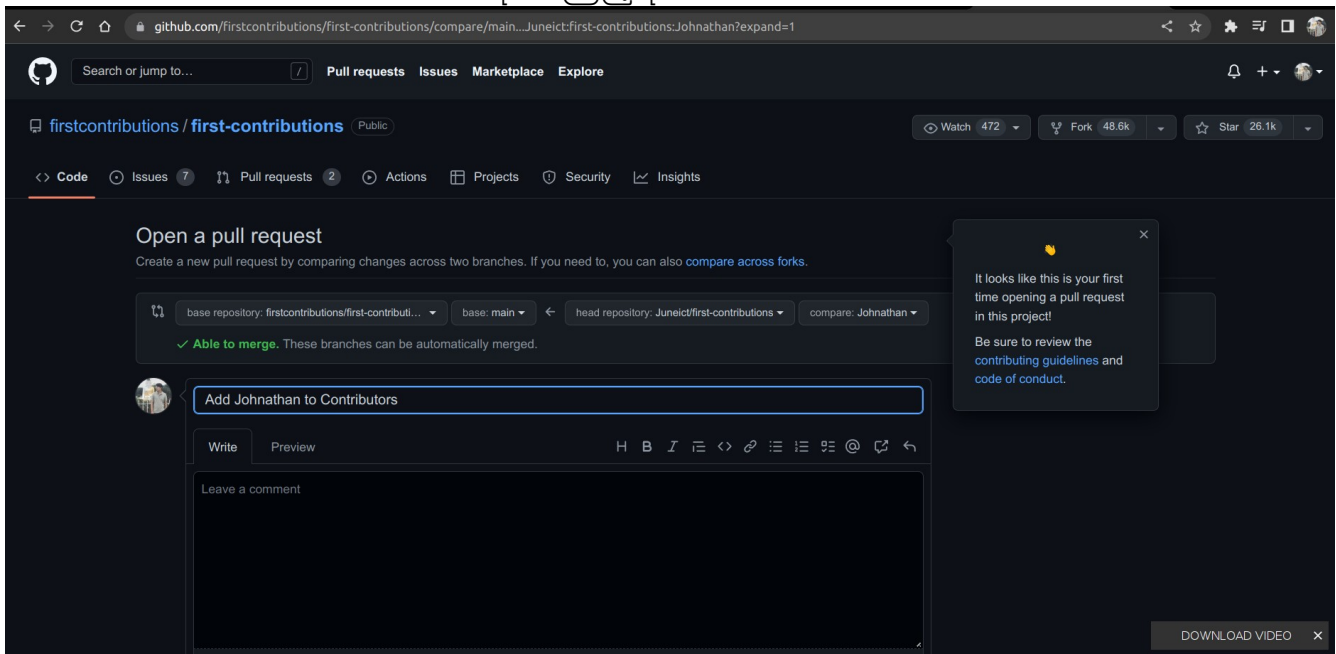
ခုနက Contributors ထဲမှာ changes ပြုလုပ်ထားလို့ modified stage ကိုတွေ့ရပါမယ်။ဒါ့ကြောင့် git add လုပ်ပြီး commit ဆက်လုပ်ကြပါမယ်။ git add . ဖြစ်စေ ဒီတစ်မိုင်တည်းပြင်ထားလို့ git add Contributors.md ကိုဖြစ်စေသုံးလိုရပါတယ်။ထို့နောက် commit လုပ်ရပါမယ်။ git commit -m “Add Johnathan to contributors”(\*message ကိုကြိုက်နှစ်သက်ရာသုံးပါ) ဖြင့် commit လုပ်ရပါမယ်။ commit လုပ်ထားသော changes များကို push လုပ်ရပါမယ်။

git push origin Johnathan(\*မိမိ branch နာမည်)ဖြင့် သုံးရပါမယ်။



Push လုပ်ပြီးပါက ဒီတိုင်းပေါ်လာပါလိမ့်မယ်။မိမိ branch မှာ contributor list ထဲတွင် မိမိနာမည် ထည့်သွင်းပီးပါပြီ။

Compare & pull request ဆိုတာလေးကို တွေ့ရာမှာဖြစ်ပါတယ်။ Pull request ဆိုတာက မိမိ ရေးထားတဲ့ code တွေကို main branch ,main repository ကို ဒီ code တွေကို main project မှာထည့် ဖိုစစ်ဆေးပေးပါဆိုတဲ့ သဘောမျိုးဖြစ်ပါတယ်။ Main project owner or project director က ဒီ code တွေသည် စစ်ဆေးပြီး အဆင်ပြေပါဆိုတဲ့အခါမှာ သူတို့ရဲ့ main project ကို လက်ခံမှာဖြစ်ပါတယ်။ Compare & pull request button ကိုစမ်းကြည့်ရအောင်ပါ။



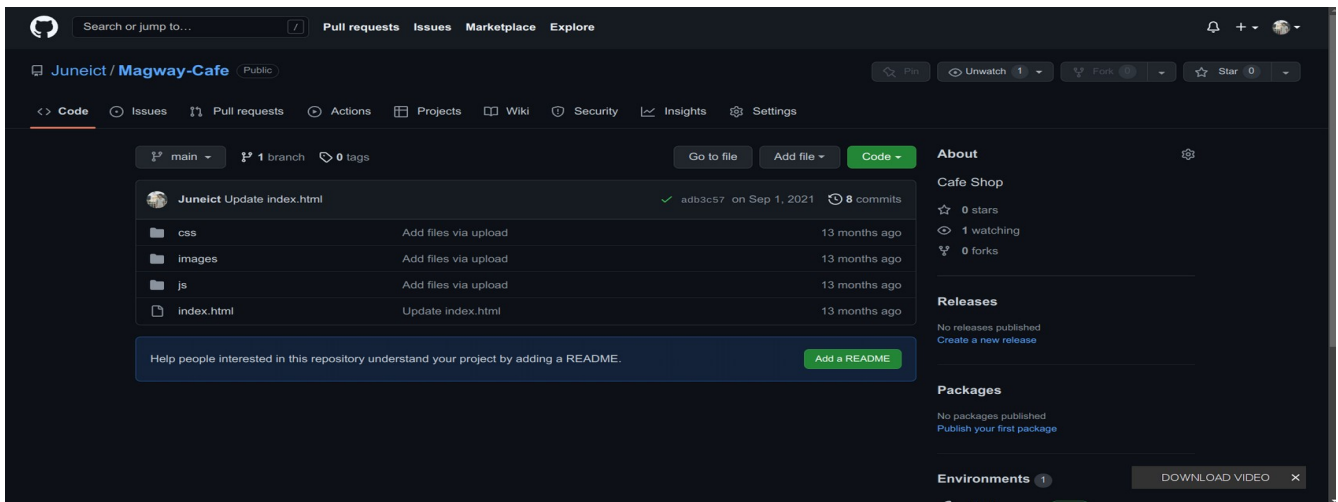
Able to merge ဆိုပြီး main project မှာ ပေါ်နေရင် project director ကဖြစ်စေ project owner က ဖြစ်စေ စစ်ဆေးပြီး သူတို့ project မှာတင်လို့ရပီးဖြစ်ပါတယ်။ create pull request ကိုနိပ်ပြီး request လိုက်လိုက်ပါတယ်။ထိုနည်းဖြင့် ကျွန်တော်တို့ project တစ်ခုမှာ developer တွေအများကြီး collaborate လုပ်နိုင်ပါတယ်။

အခုဆိုရင် git နှင့် github တို့အကြောင်းအခြေခံသဘောသဘာဝတွေကို စုံသလောက်လေ့လာပီးပါပြီ။ တကယ့်လက်တွေ့မှာတော့ developer တစ်ယောက်ရဲ့ personal workflow တစ်ခုအနေနဲ့ github repository တစ်ခုတည်ဆောက်ပီး git နဲ့ချိတ်ဆက်တာ git add လုပ်ပီး git commit လုပ်တာ branch တွေ ခွဲပီးတည်ဆောက်တာတွေကိုတော့ မဖြစ်မနေတက်မြှောက်ထားဖို့လိုပါမယ်။

## Creating a webpage using github server

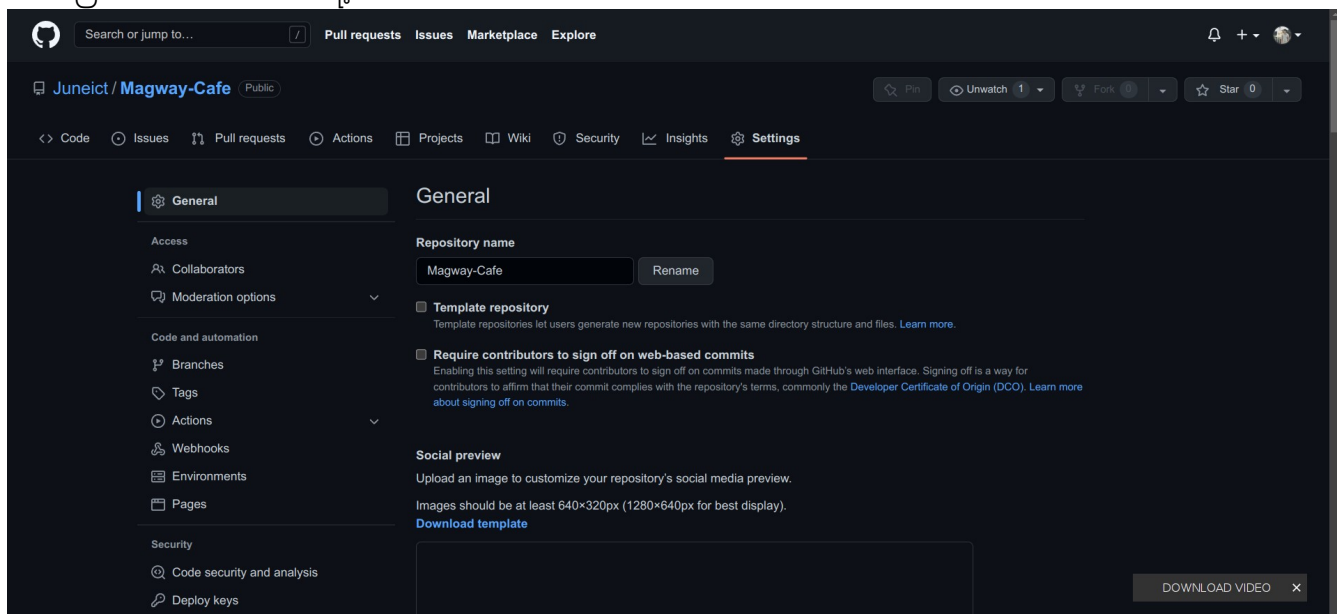
နောက်တစ်ချက်အနေနဲ့ github သည် source code တွေသိမ်းဖို့အပြင် frontend project များကို live server page အဖြစ်လည်း သုံးလို့ရပါတယ်။ web design များကို နမူနာပြသဖို့အတွက် အလွယ် အသုံးဝင်ပါတယ်။ Frontend project တွေကို web page အဖြစ်ဘယ်လိုပြောင်းမလည်း လေ့လာကြည့် ရအောင်ပါ။

အရင်ဆုံး html css ဖြင့် ရေးသားထားသော project တစ်ခုခုကို github repository တစ်ခုတည်



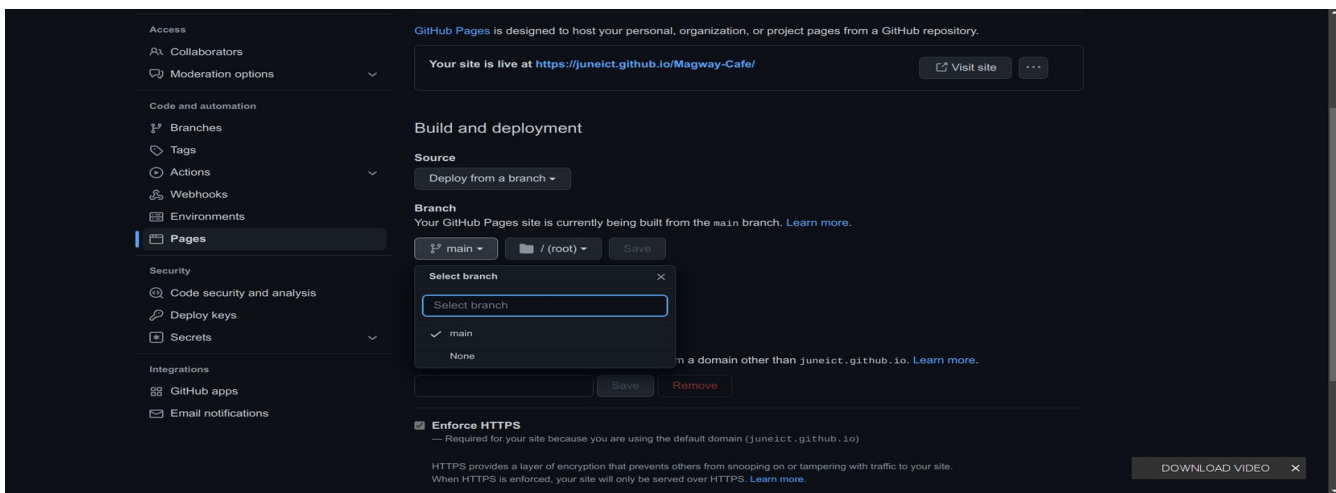
ဆောက်ပိုးတင်ထားပါ။

တင်ပြီးပါက setting ထဲသို့ဝင်ပါ။



Pages tag ထဲသို့သွားပါ။

Branch မှာ main ကိုရွေးပြီး save လိုက်လျှင် page url ပေါ်လာပါမယ်။ ခဏစောင့်ပြီးရင် web page ကို ဝင်ကြည့်လို့ရပါမယ်။



ထိုနည်းဖြင့် လွယ်ကူစွာ web page များလည်း တင်လို့ရပါတယ်။



အခုဆိုရင် လက်တွေ့အသုံးချနိုင်မယ့် git အခြေခံတွေကို လေ့လာပြီးဖြစ်ပါတယ်။ Git ဟာ အလွန် အသုံးဝင်တဲ့ system တစ်ခုဖြစ်လို့ တက်ကျွမ်းအောင်လေ့လာထားဖို့လိုပါလိမ့်မယ်။ဆက်လက်ပြီးတော့ git ignore ဖြင့် မလိုတဲ့ file တွေကို github ပေါ်မတင်ပဲထားတာတို့ ၊ git stash တို့ကို လေ့လာဖို့ကျန်ပါသေး တယ်။ဒီစာအုပ်ဟာ စာရေးသူလည်း လေ့လာရင်း ရေးထားတာဖြစ်တဲ့အတွက် လေ့လာစဉ်မှာ မှတ်သား ထားသည်အားလုံးကို စုစည်းထားတာဖြစ်တဲ့အတွက် အခြေခံတွေကိုအချိန်ကုန် သက်သက်သာသာဖြင့် လေ့လာနိုင်စေရန် ရည်ရွယ်ပါတယ်။အခြေခံတွေကို ပိုင်နိုင်ပြီဆိုကာမှ ကျန်တဲ့အပိုင်းတွေကို ဆက်လက် လေ့လာဖို့ အကြံပြုပါတယ်။

အားလုံးကိုကျေးဇူးတင်ပါတယ်။

Johnathan Moe ( Innosys Technology)

22.9.2022 တွင် ရေးသားပြီးစီးပါသည်။

ref: github.com

ref: tutorialpoint.com