

# Gitভার্সন কন্ট্রোল সিস্টেম

## গিট(git):

গিট হচ্ছে ভার্সন কন্ট্রোল সিস্টেম যা ওপেনসোর্স। এর মাধ্যমে আপনি যতবার তার পরিবর্তনগুলো কমিট(commit) করবেন ততবার গিট আপনার সম্পূর্ণ ফাইল সংরক্ষণ করে রাখবে। আপনি চাইলে ইন্টারনেট সংযোগ ছাড়াও কাজ করতে পারবেন।

Git এর কাজ হলো ভার্সন কন্ট্রোল করা। একারণে গিটকে বলা হয় ভার্সন কন্ট্রোলিং সিস্টেম বলা হয়। ভার্সন কন্ট্রোল হচ্ছে এমন একটি পদ্ধতি যা একটি প্রজেক্টের বিভিন্ন সময়ের পরিবর্তনগুলো সংরক্ষণ করে রাখে। ভার্সন কন্ট্রোল সিস্টেমের মাধ্যমে প্রজেক্টের পূর্বের যে কোন সময়ের অবস্থায় ফিরে যেতে পারে। গিট খুবই অর্গানাইজ এবং মিনিংফুল। এই ভিন্ন ভিন্ন ভার্সন গুলোর ভিন্ন ভিন্ন নাম দিয়ে ভার্সন অনুযায়ী গিটে রাখতে পারা যায়।

এছাড়াও গিটের আরো অনেক সুবিধা আছে। এর মধ্যে কয়েকটি হচ্ছে-

- এটি ব্যবহার করা সহজ।
- প্রজেক্ট কোলাবোরেশন ও রিসার্চ।
- ইন্ডাস্ট্রিয়াল সাপোর্ট।
- অর্গানাইজ এন্ড মিনিংফুল।
- মাল্টিপল ডেভেলপারের কাজ করা সুবিধা।
- টিম হয়ে কাজ করা।
- ওপেন সোর্স প্ল্যাটফর্ম।

গিট নিয়ে কাজ করার জন্য কমান্ডগুলো (command) গুলো সম্পর্কে জানতে হবে-

- **git init** একটি নতুন রিপোজিটরি(repository) তৈরি করার জন্য ব্যবহৃত হয়।
- **git status** গিটের বর্তমান status check করা।
- **git add** রিপোজিটরি(repository) তে এক বা সবগুলো ফাইল একসাথে যুক্ত করার জন্য ব্যবহৃত হয়।
- **git commit** অফলাইন(offline) রিপোজিটরিতে স্থায়ীভাবে কাজ সংযুক্ত করার জন্য ব্যবহৃত হয়।
- **git clone** পূর্ব থেকে বিদ্যমান কোন রিপোজিটরির সম্পূর্ণ তথ্য ডাউনলোড করার জন্য ব্যবহৃত হয়।
- **git pull** রিমোট(remote) রিপোজিটরি থেকে ফাইল ডাউনলোড করে অফলাইন রিপোজিটরির সাথে merge করার জন্য ব্যবহৃত হয়।
- **git log** গিট এর log check করার জন্য ব্যবহৃত হয়।
- **git push** অফলাইন রিপোজিটরি থেকে ফাইল রিমোট রিপোজিটরিতে আপলোড করার জন্য ব্যবহৃত হয়।
- **git branch** রিপোজিটরি(repository) তে নতুন branch তৈরির জন্য ব্যবহৃত হয়।
- **git checkout** master ব্রাঞ্চ থেকে অন্য ব্রাঞ্চে সুইচ করতে ব্যবহার হয়।
- **git merge** এক ব্রাঞ্চের সাথে অন্য ব্রাঞ্চে Merge করতে ব্যবহার হয়।

## What is git repository

সহজ ভাষায় বলতে গেলে গিট রিপজিটরি হলো একটি ফোল্ডার বা ডিরেক্টরি যেখানে প্রজেক্ট রাখা হয়। এটা আপনার লোকাল কম্পিউটারও হতে পারে বা কোন অনলাইন হোস্টিংও হতে পারে। এই রিপোজিটরি আপনার সম্পূর্ণ কোডবেজ এবং প্রত্যেকটি রিভিশন হিস্টরি সেভ করে রাখবে।

গিট রিপোজিটরি রাখার জন্য অনলাইন হস্টিং সার্ভিসগুলো হলো, [GitHub](#), [GitLab](#), এবং [Bitbucket](#)

## What is GitHub?

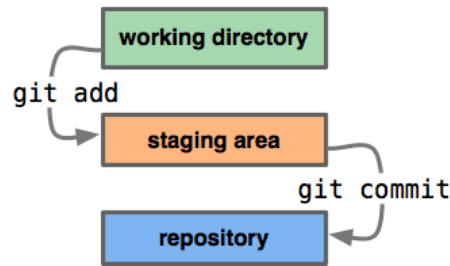
গিটহাব এমন একটি প্ল্যাটফর্ম যাকে আমরা কোড হোস্ট করার কাজে ব্যবহার করি। ধরা যাক, আমরা আমাদের লোকাল কম্পিউটারকে হোস্ট হিসেবে ব্যবহার করতছি। কিন্তু কোন কারণে যদি এই লোকাল সিস্টেমের কোন প্রব্লেম হয় তাহলে কিন্তু আমরা আমাদের ফাইলগুলোকে হারিয়ে ফেলবো। এতে আমাদের সব দিক দিয়েই ক্ষতি। তাহলে আমাদের অবশ্যই ক্লাউডে ফাইলগুলোকে রাখতে হবে।

এখন প্রশ্ন আসতেই পারে গিটহাব কেন? কারণ google Drive/onedrive এর মতো অনেক প্ল্যাটফর্ম তো আছেই। কেনই বা তাহলে Google drive/onedrive etc.. নয়!!

আমাদের মাথায় রাখতে হবে গিট এর কাজ কি। গিট এর কাজ ভার্সন কন্ট্রোল করা। যা আমরা google drive এর মাধ্যমে পারবো না। গিট এর ফিচারগুলো github এ রয়েছে। এখানে আমাদের রিমোট রিপোজিটরি থাকে এবং যেকোন সাইজের প্রজেক্টে টিমওয়ার্ক করার সকল টুলসই আছে গিটহাবে। গিটহাবে কোন প্রজেক্টকে রিপোজিটরি বলা হয়ে থাকে। গিটহাবে আমাদের কোন রিপোজিটরি বা প্রজেক্টে কাজ করতে হলে অবশ্যই একটি গিটহাব একাউন্ট এর দরকার হবে। আমাদের কোডবেস যদি গিটহাবের মতো কোন রিমোট রিপোজিটরি তে থাকে তবে নিচের সুবিধাগুলো পাবো,

- ১। প্রজেক্টের সকল কন্ট্রিবিউটর একটি কমন রিমোট রিপোজিটরি থেকে লেটেস্ট ভার্সন এর কোড নিতে পারবে।
- ২। কন্ট্রিবিউটররা একসাথে গিটহাবের অটোমেশন টুলস ব্যবহার করে দ্রুততার সাথে কাজ করতে পারবে।

## গিট যেভাবে কাজ করে



### Working directory:

ওয়ার্কিং ডিরেক্টরি বলতে যে ফোল্ডারে বা আমাদের প্রজেক্টকে ইনিশিয়ালাইজড করা হয়।

### Staging Area details:

Stage হচ্ছে একটি ইন্টারমেডিয়েট অবস্থা। ওয়ার্কিং ডিরেক্টরিতে ফাইলগুলোতে চেঞ্জ বা মডিফাই করা হয়। ফাইনাল commit করার আগে গিটকে বুঝানো হয় যে মডিফাইড ফাইলগুলো এখন প্রস্তুত commit করার জন্য। আমরা যখন লোকাল (local) রিপোজিটরিতে কোন পরিবর্তন করি তখন আমরা ওয়ার্কিং ডিরেক্টরিতে থাকি। git add কমান্ড দেয়ার পর সেটা staging area তে যায় এবং git commit কমান্ড দেয়ার পর সেটা স্থায়ীভাবে লোকাল রিপোজিটরিতে যুক্ত হয়। পরবর্তিতে চাইলে সেটা রিমোট রিপোজিটরিতে git push কমান্ড দিয়ে আপলোড করে দেয়া যায়।

### Repository details :

কোন প্রজেক্ট তৈরি করার জন্য প্রথমে আমাদের কোন একটি লোকেশনে রাখতে হয়। সেটা দুইভাবে হতে পারে

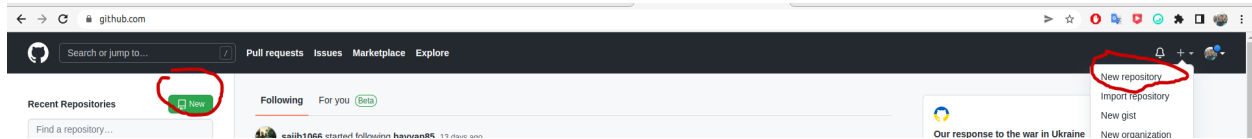
১। লোকাল ( Local )

আমরা যখন আমাদের লোকাল কম্পিউটারে যেকোনো লোকেশনে একটি ফোল্ডার তৈরি করে গিট init এর মাধ্যমে ইনিশিয়ালাইজ করি তখন তা রিপোজিটরিতে তৈরি হয়।

২। ক্লাউড ( Remote বা Cloud )

গিট এর রিমোট অথবা ক্লাউড হিসেবে কিছু ওয়েবসাইট আছে যারা গিটকে ম্যানেজ করে। যেমন: [GitHub](#), [GitLab](#), এবং [Bitbucket](#) ইত্যাদি। এই ওয়েবসাইটগুলোতে গিট এর ফাংশনালিটি আছে। প্রথমে [www.github.com](https://www.github.com) এ লগইন করতে হবে।

হোম ড্যাসবোর্ডে New এবং New repository তে ক্লিক করলেই একটি New page ওপেন হবে। সেখানে repository name , Description (optional) দিয়ে create repository তে ক্লিক করলেই ক্লাউডে একটি নতুন রিপোজিটরি তৈরি হয়ে যাবে।



## Git Install

কম্পিউটারকে গিট ব্যবহারের উপযোগী করে নিতে কম্পিউটারে গিট ইন্সটল করতে হবে। কিভাবে গিট ইন্সটল এবং কনফিগার করা হয় এখন আমরা তা জানবো।

মাইক্রোসফট উইন্ডোজ:

Git Download এর জন্য <https://git-scm.com/downloads> লিঙ্ক এ যেতে হবে এবং আপনার সিস্টেম অনুযায়ী ফাইলটি ডাউনলোড করে এবং ইন্সটল করতে পারবেন।

লিনাক্স:

Linux এর বিভিন্ন ডিস্ট্রো এর জন্য গিট ইন্সটল পদ্ধতি বিভিন্ন রকম। কিন্তু খুবই সহজ। ডিস্ট্রো অনুযায়ী terminal-এ নিচের কমান্ডগুলো লিখুন।

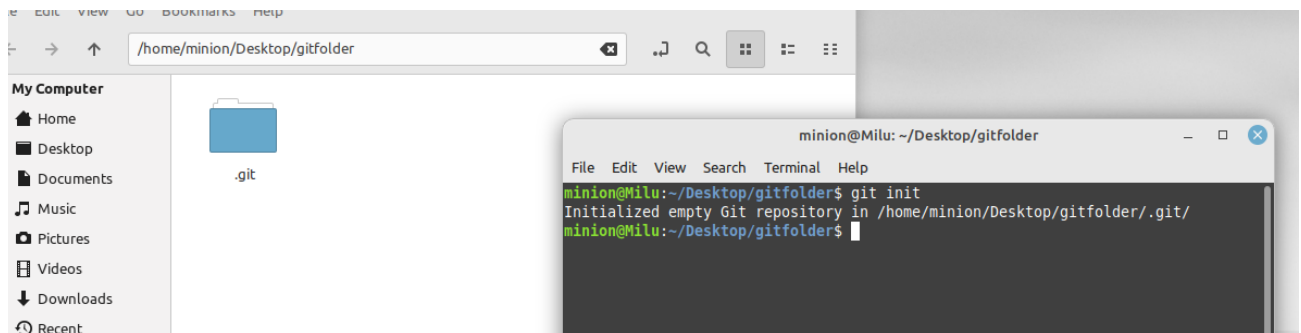
- Debian/Ubuntu  
# apt-get install git
- Fedora  
# yum install git
- Arch Linux  
# pacman -S git

- FreeBSD  
`$ cd /usr/ports/devel/git`  
`$ make install`
- Solaris 11 Express  
`$ pkg install developer/versioning/git`
- OpenBSD  
`$ pkg_add git`

## Git init

গিট সবসময় আমাদের কাজ কওরা ফাইলগুলোর দিকে চেয়ে থাকে। আমরা যখনই কোন কিছু চেঞ্জ করতে চাইবো তখন init এর মাধ্যমে সেভ করতে পারে, ডিলিট করতে পারে।

এজন্য আমাদের প্রথমে Git কে একটিভ করতে হবে। সুতরাং গিট নিয়ে কাজ করতে হবে প্রথমে একটি repository তৈরি করতে হয়। প্রথমে একটি ডিরেক্টরি/ফোল্ডার তৈরি করতে হবে। এই ডিরেক্টরিকে লোকাল repository বানাতে git init কমান্ড ব্যবহার করতে হয়। তাহলে সে ডিরেক্টরির এর মধ্যে একটি Hidden ফোল্ডার .git তৈরি হবে।



আমরা আগেই জেনেছি git bash এবং Terminal কি। যদি আমরা উইন্ডোজে কাজ করতে চাই তাহলে Git Bash ব্যবহার করতে হবে আর যদি লিনাক্সকে কাজ করতে চাই তাহলে linux terminal এ কমান্ডটি লিখতে হবে এবং git bash বা linux terminal ইউজ করার আগে অবশ্যই git install করে নিতে হবে।

## Git Configure

কনফিগারেশনের জন্য আপনাকে শুধু আপনার username এবং email address বলে দিতে হবে। এতে আপনি যখন গিট এর মাধ্যমে কমিট(commit) করবেন তখন কমিটের সাথে সে আপনার তথ্য সংরক্ষণ করে রাখবে।

সিস্টেমে যত প্রজেক্টে গিট ইউজ করবো তার সবগুলোতে ইউজারের নাম আর ইমেইল হিসাবে এগুলোই ইউজ করবে। এখানে যে email address দিবেন তা অবশ্যই আপনার সার্ভার অ্যাকাউন্টের email address এর সাথে মিল থাকতে হবে।

Username -কেউ যখন আপনার প্রজেক্টটি দেখবে তখন বুঝতে পারবে এই কাজগুলো কে করেছে এবং কমিটগুলো কে করেছে। এবং email কমান্ডে email লেখার পর এবার গিট এর কাছে আমাদের ইমেইল সেভ হয়ে যাবে। এখন গিট জানলো আমার ইমেইল অ্যাড্রেস কি। এখন যদি কেউ জানতে চায় এই গিটটি যে করেছে তার ইমেইল অ্যাড্রেস কি তাহলে সহজেই বুঝতে পারবে।

Windows এ git ইন্সটল করার পর ডেস্কটপে git bash নামে একটি শর্টকাট টার্মিনাল তৈরি হবে এবং লিনাক্স টার্মিনালে নিচের কমান্ড গুলো লিখতে হবে।

```
git config --global user.name "Your Name Here"
git config --global user.email "your_email@youremail.com"
```

Example:

```
git config --global user.name "Zahid Hasan Milu"
git config --global user.email "Zahidhasan.milu@gmail.com"
```

এবং চেক করার জন্য কমান্ড:

```
git config --global user.name
git config --global user.email
```

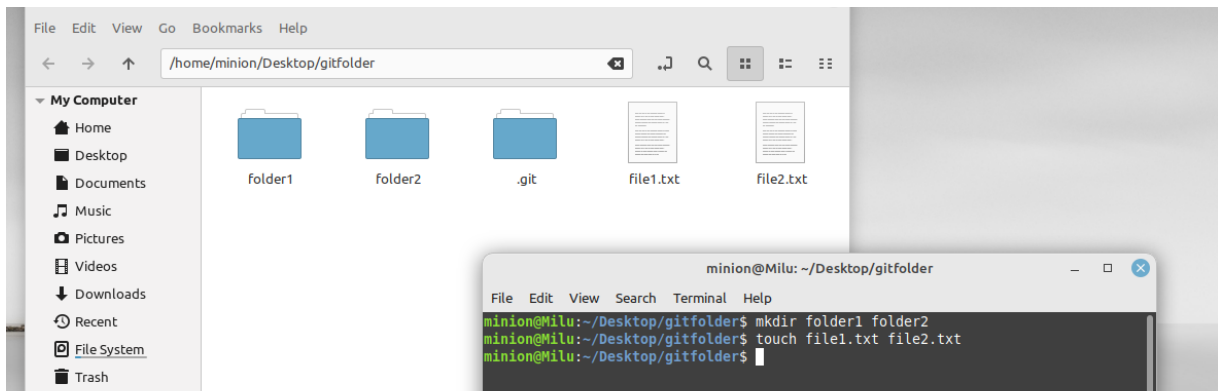
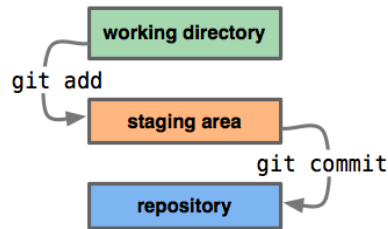
লিনাক্স ব্যবহারকারীরা Terminal ব্যবহার করবে এবং Windows ব্যবহারকারীর git bash ব্যবহার করবে।

## Git status

**git status** কমান্ড দিয়ে git এর বর্তমান অবস্থা জানা যায়।

## Git add

`git add` . মানে হচ্ছে সব গুলো ফাইল গিট রিপোজিটোরিতে যুক্ত করা। আমাদের প্রজেক্টে মাত্র দুইটা ফাইল রয়েছে। তাই দুইটাই যুক্ত হবে। মানে আমরা এখন `git add` এর মাধ্যমে স্টেজে চলে যাবে।



## git add কয়েকভাবে করা যায় :

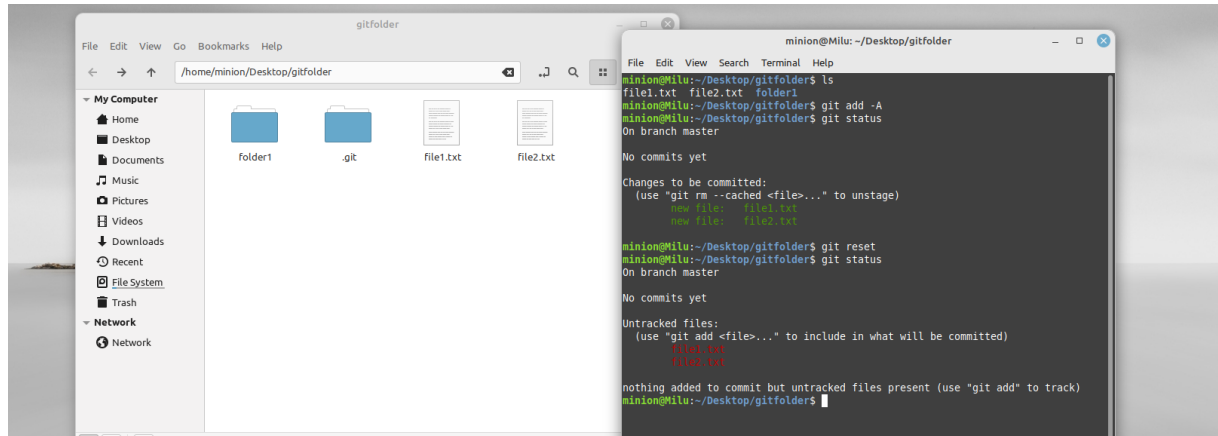
`git add relativepath` ফাইলের রিলেটিভ পথ ব্যবহার করে ফাইন নির্দিষ্ট করে গিট add করা যায়।

```
git add file1.txt
```

`git add .` কারেন্ট ডিরেক্টরিকে কেন্দ্র করে তার ভেতর ফাইলে যত চেঞ্জই থাক না কেন সবাই stage এ উঠে যাবে ,মানে add হয়ে যাবে।

`git add -A` ডিরেক্টরিকে কেন্দ্র না করে সব ফাইল ফোল্ডারকেই stage এ তুলে দেয়।

`git add --all` এর কাজটাও `git add -A` মত ।



আমরা এখানে প্রথমে `ls` করে ফোল্ডারে মধ্যে যা আছে তা দেখে নিলাম । এবার `git add -A` করে সব ফাইলকে স্টেজে তুলে নিলাম । এবার `git status` দিয়ে চেক করলাম । টার্মিনালে সবুজ `new file: file1.txt` , `new file: file2.txt` হয়ে আছে । এবং `git reset` দিয়ে আবার আগের মত `unstage` এ চলে গেছে । আবার `git status` চেক করতে দেখা যাবে টার্মিনালে লাল `new file: file1.txt` , `new file: file2.txt` হয়ে আছে ।

`git add --all` ঠিক এভাবেই কাজ করে ।

এখানে যে কমান্ডগুলো ব্যবহার হয়েছে ।

`git add -A`

`git status`

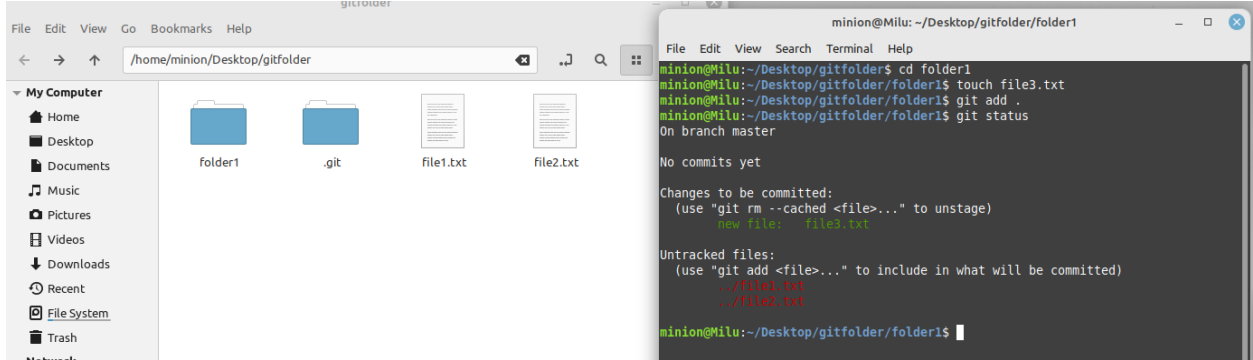
`git reset`

`git status`



## Git add .

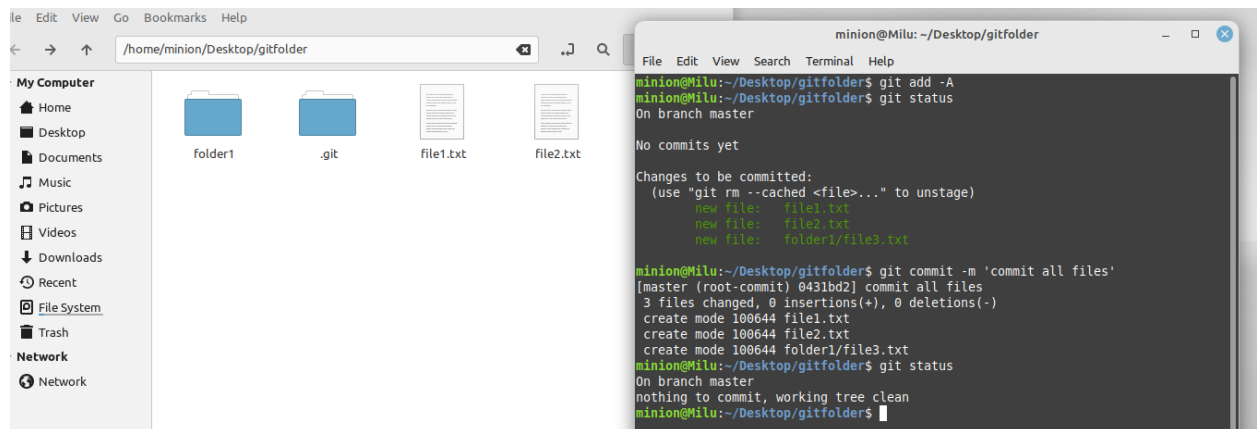
এখানে আমরা `cd folder1` করে `folder1` এর মধ্য ঢুকে `touch file3.txt` তৈরি করলাম। এবং `folder1` এ থাকা অবস্থায় যদি `git add .` কমান্ড ব্যবহার করি তাহলে শুধু `file3.txt` স্টেজে উঠে যাবে। কিন্তু `file1.txt` এবং `file2.txt` স্টেজে যাবে না।



## Git commit

Stage এ তোলা ফাইলের চেকসগুলোকে রিপোজিটোরিতে রাখতে হলে কমিট করতে হয়। আর কমিট করতে চাইলে, প্রত্যেক কমিটের সাথে একটা ম্যাসেজও দিতে হয়। এতে করে বুঝতে সুবিধা হয় কোন কমিটটা ঠিক কি কারণে করা হয়েছিলো। file যতবার মডিফাই করা হবে ততবারই সেই ফাইলকে স্টেজে তুলে commit করতে হবে।

`git commit -m "New Message Added"`



## Git Reset

স্টেজে থাকা অবস্থায় রিসেট করতে চাইলে এই `git reset` এই কমান্ড করতে পারবো। কিন্তু মনে রাখতে হবে commit পর ফাইলকে unstage অথবা commit কে reset করতে অন্য কমান্ড ব্যবহার করতে হয় -

```
git reset HEAD~
```

```
git reset path
```

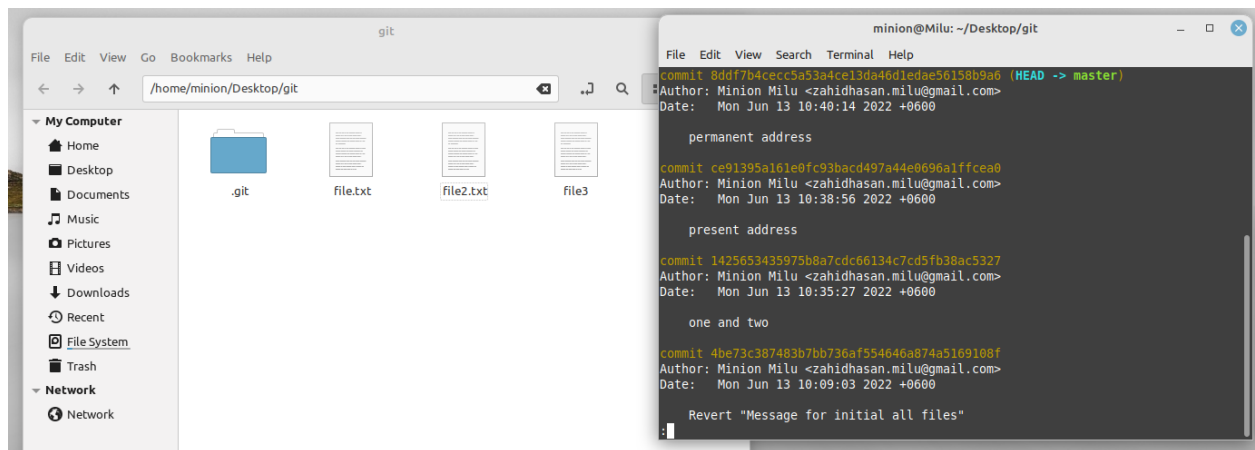
## Git log

গিট লগের মাধ্যমে আমরা দেখতে পাবো আমাদের কমিট করা ডিটেলস। গিটের ম্যাসেজ দেখে খুব সহজেই বুঝতে পারা যায় কোন কমিটে কি করা হয়েছিলো। আর সাথে কিছু এনক্রিপ্টেড ইউনিক কমিট আইডি আছে। চাইলে এগুলো ইউজ করে আবার পূর্বের ভার্সনগুলোয় ফিরে যাওয়া যায়।

গিট লগগুলো দেখার কমান্ডঃ

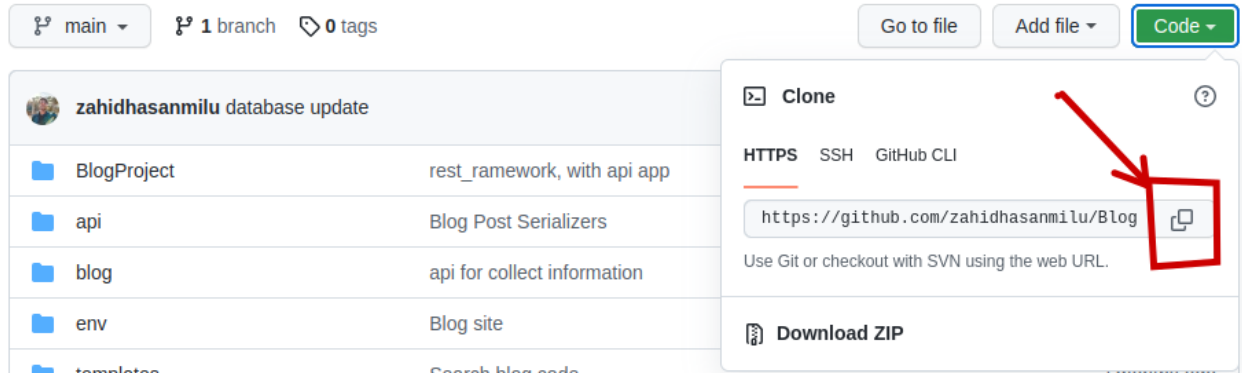
```
git log
```

```
git log -oneline
```



## Git clone

যদি গিটহাবে থাকা কোনো প্রজেক্টের একটা ক্লোন কপি আপনার লোকাল ম্যাশিনে নামিয়ে আনতে চাই তাহলে ক্লোন করতে হয়।



এবং তা লোকাল মেশিনে সেটআপ করার জন্য একটি কমান্ড লিখতে হয়।

```
git clone https://github.com/zahidhasanmilu/Blog-site-with-Django.git
```

লাল রঙের লিঙ্কটি উপরের ছবির ক্লোন কপি লিঙ্ক। তারপর জাস্ট Enter চাপলেই লোকাল মেশিনের একটা ডিরেক্টরি বা পথ দেখিয়ে দিলেই আস্তে আস্তে প্রজেক্টটি ক্লোন হয়ে যাবে।

## Git Push:

গিটকে গিটহাবে পাঠাতে চাইলে git push কমান্ডটি ব্যবহার করতে হয়। কিন্তু মনে রাখতে হবে git push করার আগে অবশ্যই আপনার ফাইলকে commit করতে হবে।

```
git push
```

```
git push 'গিট প্রজেক্ট লিঙ্ক'
```

## Git Branch

টিম হয়ে কাজ করলে কিছু কমান্ড জানা লাগে। কারণ যখন আমরা টিমে কাজ করি, তখন একই প্রজেক্টে অনেক জন ডেভেলপার এক সাথে কাজ করি। দেখা যায় এক জন এক এক ফিচারে কাজ করে। আর এ থেকেই এসেছে Branch এর ধারণা। এক এক জন এক এক ব্রাঞ্চে কাজ করে। আবার যখন এক একটা ফিচার এর কাজ কমপ্লিট হয়ে যায়, তখন সব কিছু আবার তা মূল প্রজেক্টের সাথে যুক্ত করতে হয়। একে বলা হয় Merge।

কতগুলো ব্রাঞ্চ আছে তা দেখার জন্য

```
git branch
```

সহজ ভাষায়, যদি আমরা একটা ওয়েব সাইট তৈরি করি। তখন একজন কাজ করে index পেইজ নিয়ে। আরেকজন কাজ করে authentication পেইজ নিয়ে।

তখন যে index কাজ করে, সে index নামে একটা ব্রাঞ্চ তৈরি করে নিবে।

আর যে authentication এ কাজ করে, সে authentication নামে একটা ব্রাঞ্চ তৈরি করে নিবে।

কাজ শেষে মূল প্রজেক্টের সাথে merge করে নিতে হবে এবং merge করার আগে অবশ্যই index ,authentication ব্রাঞ্চের এর চেঞ্জগুলোকে commit করে নিতে হবে। এর চেয়ে সহজ ভাষা আর কি হতে পারে।

নতুন ব্রাঞ্চ তৈরি করার জন্য

```
git branch branch_name
```

Example:

```
git branch newbranch
```

## Check Your Active Branch

একটি ব্রাঞ্চ কোনটা তা বোঝার জন্য git branch কমান্ড এ এমন দেখা যাবে

```
*master ( স্টার মার্ক করা থাকে একটি ব্রাঞ্চএ )
```

```
newbranch
```

## Checkout the Branch( ব্রাঞ্চে প্রবেশ করা )

git এ master নামে একটি ডিফল্ট ব্রাঞ্চ তৈরি থাকে, অন্য একটা ব্রাঞ্চ তৈরি করার পর master ব্রাঞ্চ থেকে ঐ ব্রাঞ্চে সুইচ করার জন্য লিখতে হয়ঃ

```
git checkout branch_name
```

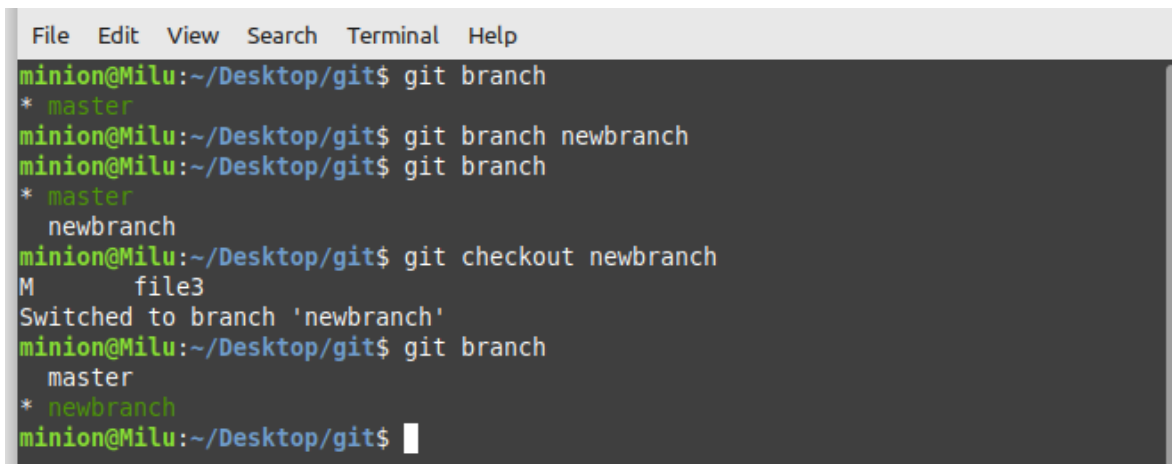
Example:

```
git checkout newbranch
```

এখন যদি git branch কমান্ডটি লিখি তাহলে দেখতে পাবো newbranch ব্রাঞ্চটি একটিভ হয়ে গেছে।

```
master
```

```
*newbranch
```

A screenshot of a terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a dark background. The terminal shows the following commands and output:

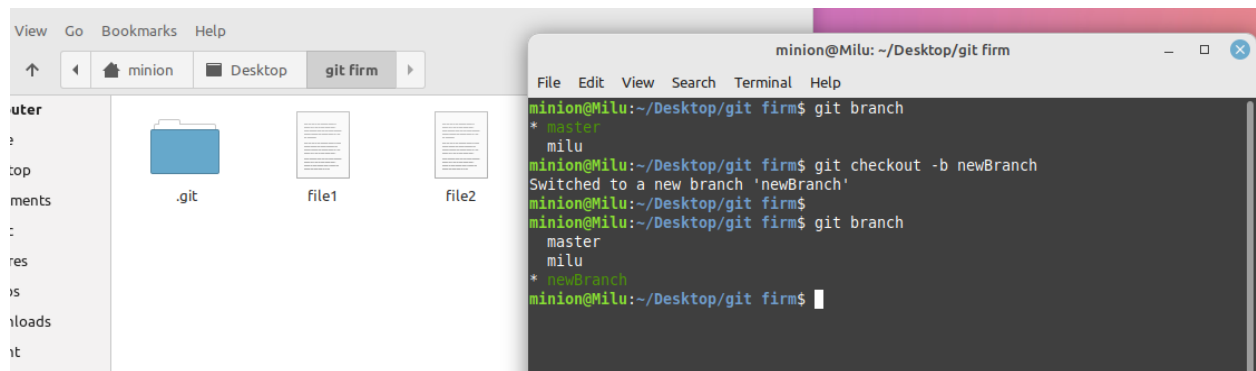
```
minion@Milu:~/Desktop/git$ git branch
* master
minion@Milu:~/Desktop/git$ git branch newbranch
minion@Milu:~/Desktop/git$ git branch
* master
  newbranch
minion@Milu:~/Desktop/git$ git checkout newbranch
M      file3
Switched to branch 'newbranch'
minion@Milu:~/Desktop/git$ git branch
  master
* newbranch
minion@Milu:~/Desktop/git$
```

আমরা দেখেছি গিট ব্রাঞ্চ তৈরি করে আলাদাভাবে চেকআউট কমান্ড লিখে নতুন ব্রাঞ্চটি একটিভ করতে হয় । কিন্তু আমরা চাইলে ব্রাঞ্চ তৈরি করে সাথে সাথেই নতুন ব্রাঞ্চে একটিভ হতে পারবো ।

সেকারণে আমাদের এই কমান্ডটি ব্যবহার করতে হবেঃ

Create and checkout the new branch in one command:

```
git checkout -b newBranch
```



## Git merge

git merge আসলে কি ?

আমরা একটু আগেই কিছু ব্রাঞ্চ তৈরি করেছি। সেগুলোকে চেকআউটও করেছি। মনে করে ডিফট ব্রাঞ্চ master এখানে পুরো প্রজেক্ট আছে। এখন এই প্রজেক্টে অনেকে কাজ করতে পারে। কিন্তু সবাই মিলে যদি একই ব্রাঞ্চের ফাইলগুলোকে মডিফাই করে তাহলে কোডে ঝামেলা তৈরি হয়ে যাবে।

তাই আলাদা ব্রাঞ্চ তৈরি করে যদি মেইন ফাইলগুলো modify করার পর commit করে রাখা হয় তাহলে master ব্রাঞ্চে যে কম্যান্ডের মাধ্যমে Modified file গুলোকে add করানো হয় সেটাই হলো merge।

এক কথায় এক ব্রাঞ্চের ফাইল, অন্য ব্রাঞ্চে নিয়ে যাওয়া হয় merge এর মাধ্যমে।

```
git merge branch_name
```

## Work with New offline repository

অফলাইনে গিট রিপোজিটরি ব্যবহার করার জন্য নিচের ধাপগুলো অনুসরণ করুন

1. ধরুন, আপনি যদি একটি routine management system তৈরি করতে চান এবং আপনার রিপোজিটরি ফোল্ডারের নাম যদি rms দিতে চান, প্রথমে

```
mkdir rms
```

কমান্ড টি লিখুন। এখন cd ব্যবহার করে আপনার রিপোজিটরিতে প্রবেশ করতে লিখুন

```
cd rms
```

2. এখন

```
git init
```

কমান্ড লিখতে হবে। এতে করে রিপোজিটরিটি গিট ব্যবহারের উপযোগি হবে এবং .git নামে একটি লুকানো ফোল্ডার তৈরি হবে। আপনার রিপোজিটরিতে .git ফোল্ডারটি তৈরি না থাকলে গিটের কমান্ডগুলো কাজে লাগাতে পারা যাবে না।

3. রিপোজিটরিটি এখন কাজের উপযুক্ত হয়েছে। এখন এতে যে কোন ধরনের ফাইল, ফোল্ডার তৈরি, সম্পাদনা ও মুছে ফেলতে পারা যাবে।

4. রিপোজিটরি প্রত্যেকবার স্থিতিশীল(stable) অবস্থায় পৌঁছলে সেগুলো Staging Area তে পাঠাতে হবে। Staging area তে পাঠানোর মানে হচ্ছে যে যে ফাইলগুলোকে নিশ্চিতভাবে রিপোজিটরিতে অন্তর্ভুক্ত করবেন তার তালিকা তৈরি করা। সে জন্য git add কমান্ড ব্যবহার করতে হবে। যদি আপনি বর্তমান সবগুলো ফাইলকে Staging area তে সংযুক্ত করতে চান তবে

```
git add *
```

অথবা

```
git add -A
```

অথবা

```
git add --all
```

অথবা

```
git add .
```

কমান্ড ব্যবহার করতে হবে। সেটা অবশ্যই আপনি কি করতে চাচ্ছেন তার উপর নির্ভর করে কমান্ডটি ব্যবহার করতে হবে।

যদি একটি ফাইলকে সংযুক্ত করতে চান তবে git add কমান্ডের পর ফাইলের নাম লিখতে হবে। ধরুন আপনি test.txt নামে একটি ফাইল তৈরি করেছেন যা আপনি Staging area তে পাঠাতে চান। তখন আপনার কমান্ড হবে git add test.txt

আমরা কিন্তু ইতিপূর্বে এই কমান্ডগুলো সম্পর্কে বিস্তারিত জেনেছি।

5. সর্বশেষ যে কাজটি করতে হবে তা হচ্ছে, staging area তে সংযুক্ত করা ফাইলগুলোকে মূল রিপোজিটরির সাথে নিশ্চিতভাবে সংযুক্ত করা। সে জন্য git commit কমান্ডটি ব্যবহার করতে হবে। প্রত্যেক commit এর সাথে একটি বার্তা সংযুক্ত করে দিতে হবে। তখন সম্পূর্ণ কমান্ডটি হবে-

```
git commit -a -m "Your Message"
```

এখানে "Your Message" এর যায়গায় ইচ্ছেমত যেকোন টেক্সট দিতে পারেন।