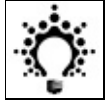


## সূচিপত্র

শুরুর আগে	0
ভূমিকা	1
ইন্সটলেশন এবং কনফিগারেশন	2
নতুন রিপোজিটরি	3
রিমোট রিপোজিটরি	4
মাল্টিপল ডেভেলপার	5
একাধিক রিমোট রিপো	6
গিট-ফ্লুও পরিচিতি	7
গিট-ফ্লুও ইন্সটলেশন এবং কনফিগারেশন	7.1
গিট-ফ্লুওফিচার	7.2
গিট-ফ্লুও রিলিস	7.3
গিট-ফ্লুও হটফিক্স	7.4



howtocode.com.bd

পৃষ্ঠা লাইক করুন 10হাজার পছন্দগুলি

আপনার বন্ধুদের মধ্যে আপনিই এটা প্রথম পছন্দ করুন

কোর্স এর মূল পাতা | [HowToCode মূল সাইট](#) | [সবার জন্য প্রোগ্রামিং ব্লগ](#) | [পিডিএফ ডাউনলোড](#)

## ভার্সন কন্ট্রোল সিস্টেম – গিট (git)

gitter join chat



theazharul 18



howtocode-com-bd 11



masudiuc 11



nuhil 8

ভার্সন কন্ট্রোল (Version Control) : ভার্সন কন্ট্রোল হচ্ছে এমন একটি পদ্ধতি যা আপনার প্রজেক্টের(project) বিভিন্ন সময়ের পরিবর্তনগুলো সংরক্ষণ করে রাখে। ভার্সন কন্ট্রোল সিস্টেমের মাধ্যমে আপনি আপনার প্রজেক্টের পূর্বের যে কোন সময়ের স্থিতিশীল অবস্থায় ফিরে যেতে পারবেন।

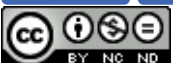
গিট(git): গিট হচ্ছে একটি ওপেনসোর্স(open source) ভার্সন কন্ট্রোল সিস্টেম। এর মাধ্যমে একজন ব্যবহারকারী যতবার তার পরিবর্তনগুলো কমিট(commit) করবে ততবার গিট তার সম্পূর্ণ ফাইল সংরক্ষণ করে রাখবে। গিট এর একটি বড় সুবিধা হচ্ছে একটি প্রজেক্ট নিয়ে অসংখ্য ডেভেলপার(developer) একই সময় কাজ করতে পারে। আপনি চাইলে ইন্টারনেট সংযোগ ছাড়াও কাজ করতে পারবেন।

### ওপেন সোর্স

এই বইটি মূলত স্বেচ্ছাশ্রমে লেখা এবং বইটি সম্পূর্ণ ওপেন সোর্স। এখানে তাই আপনিও অবদান রাখতে পারেন লেখক হিসেবে। আপনার কন্ট্রিবিউশান গৃহীত হলে অবদানকারীদের তালিকায় আপনার নাম যোগ করে দেওয়া হবে।

এটি মূলত একটি [গিটহাব রিপোজিটরি](#) যেখানে এই বইয়ের আর্টিকেল গুলো মার্কডাউন ফরম্যাটে লেখা হচ্ছে। রিপোজিটরিটি ফর্ক করে পুল রিকুয়েস্ট পাঠানোর মাধ্যমে আপনারাও অবদান রাখতে পারেন।

Like 90 Share



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).



# ভার্সন কন্ট্রোল সিস্টেম – গিট (git)

## ভার্সন কন্ট্রোল (Version Control) :

ভার্সন কন্ট্রোল হচ্ছে এমন একটি পদ্ধতি যা আপনার প্রজেক্টের(project) বিভিন্ন সময়ের পরিবর্তনগুলো সংরক্ষণ করে রাখে। ভার্সন কন্ট্রোল সিস্টেমের মাধ্যমে আপনি আপনার প্রজেক্টের পূর্বের যে কোন সময়ের স্থিতিশীল অবস্থায় ফিরে যেতে পারবেন।

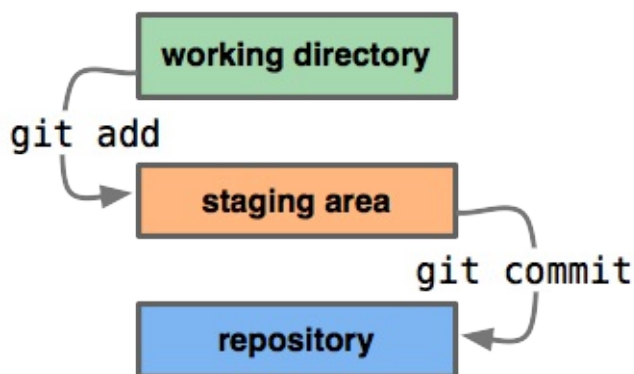
## গিট(git):

গিট হচ্ছে একটি ওপেনসোর্স(open source) ভার্সন কন্ট্রোল সিস্টেম। এর মাধ্যমে একজন ব্যবহারকারী যতবার তার পরিবর্তনগুলো কমিট(commit) করবে ততবার গিট তার সম্পূর্ণ ফাইল সংরক্ষণ করে রাখবে। গিট এর একটি বড় সুবিধা হচ্ছে একটি প্রজেক্ট নিয়ে অসংখ্য ডেভেলপার(developer) একই সময় কাজ করতে পারে। আপনি চাইলে ইন্টারনেট সংযোগ ছাড়াও কাজ করতে পারবেন।

আসুন দেখে নেই গিট এর বহুল ব্যবহৃত কমান্ডগুলো(command)-

- git init -একটি নতুন রিপোজিটরি(repository) তৈরি করার জন্য।
- git clone – পূর্ব থেকে বিদ্যমান কোন রিপোজিটরির সম্পূর্ণ তথ্য ডাউনলোড করার জন্য
- git commit – অফলাইন(offline) রিপোজিটরিতে স্থায়ীভাবে কাজ সংযুক্ত করার জন্য
- git pull – রিমোট(remote) রিপোজিটরি থেকে ফাইল ডাউনলোড করে অফলাইন রিপোজিটরির সাথে merge করার জন্য
- git push – অফলাইন রিপোজিটরি থেকে ফাইল রিমোট রিপোজিটরিতে আপলোড করার জন্য

## কাজের ধাপসমূহ:



আমরা যখন লোকাল(local) রিপোজিটরিতে কোন পরিবর্তন করি তখন আমরা working directory-তে 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।



# গিট : ইন্সটল ও কনফিগার করা

গিট নিয়ে কাজ শুরু করার জন্য আপনার কম্পিউটারকে গিট ব্যবহার উপযোগী করে নিতে হবে। সেজন্য আপনার কম্পিউটারে গিট ইন্সটল(install) করতে হবে। চলুন দেখে নেয়া যাক, কিভাবে গিট ইন্সটল এবং কনফিগার(configure) করে নিতে হয়।

## Windows:

১. গিট প্যাকেজটি ডাউনলোড করার জন্য এখানে [ক্লিক](#) করুন।
২. ডাউনলোড করা ফাইলটি ক্লিক করে ইন্সটল করে নিন।

## Linux:

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
```

- Gentoo

```
# emerge --ask --verbose dev-vcs/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
```

## গিট কনফিগার

গিট কনফিগার করার মূল উদ্দেশ্য হচ্ছে, আপনি যখন গিট এর মাধ্যমে কমিট(commit) করবেন তখন কমিটের সাথে সে আপনার তথ্য সংরক্ষণ করে রাখবে। কনফিগারেশনের সময় আপনাকে শুধু আপনার user name এবং email address বলে দিতে হবে। Windows ব্যবহারকারীরা গিট ইন্সটল করার পর কম্পিউটার ডেস্কটপে gitBash নামে একটি শর্টকাট ফাইল তৈরি হবে। সেটি খুলে তাতে নিচের কমান্ডগুলো লিখুন। Linux ব্যবহারকারীরা terminal-এ কমান্ডগুলো লিখতে পারবেন।

```
git config --global user.name "Your Name Here"
```

Your Name Here এর জায়গায় আপনার নাম লিখুন।

```
git config --global user.email "your_email@youremail.com"
```

এখানে যে email address দিবেন তা অবশ্যই আপনার সার্ভার অ্যাকাউন্টের email address এর সাথে মিল থাকতে হবে। এখন আপনার কম্পিউটারটি গিট ব্যবহার উপযোগি হয়েছে। এখন থেকে আপনি আপনার কম্পিউটারে গিটের কমান্ডগুলো কাজে লাগাতে পারবেন। আমরা আমাদের সকল কমান্ড gitBash অথবা terminal-এ লিখবো।

~/ .gitconfig ফাইল এ আপনার গ্লোবাল(global) কনফিগারেশন লেখা থাকে। আপনি যখন কোন একটি রিপোজিটরিতে git init কমান্ড দিবেন তখন .git নামের একটি ফোল্ডার তৈরি হবে। এই .git ফোল্ডারের ভিতরেই গিট সব তথ্য সংরক্ষিত রাখে।

আপনি কোন একটি রিপজিটরির জন্য গিটের গ্লোবাল কনফিগারেশনকে পরিবর্তন অথবা ওভাররাইট করতে পারেন। আপনি যদি কোন একটি রিপজিটরির জন্য name এবং email পরিবর্তন করতে চান তাহলে terminal দিয়ে রিপজিটরিতে গিয়ে নিচের কমান্ডগুলো দেন।

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

এখনে আপনি যে নাম এবং ইমেইল দিবেন তা শুধুমাত্র এই রিপোজিটরির জন্য কনফিগার হবে। আপনার লোকাল কনফিগার .git/config ফাইল এ সংরক্ষিত হয়। Linux/Mac ব্যবহারকারীরা terminal-এ নিচের কমান্ড দিয়ে লোকাল কনফিগারেশন এর ফাইলটি দেখতে পারেন।

```
nano .git/config
```

ফাইল এর শেষে যদি নিচের লাইনগুলো যোগ হয় তাহলে আপনার লোকাল কনফিগার সঠিকভাবে overwrite হয়েছে।

```
[user]
  name = <your-name>
  email = <your-email>
```

আপনি নিচের কমান্ড দিয়ে আপনার গিট কনফিগারেশন একসাথে দেখতে পারবেন।

```
git config --list
```



# গিট : নতুন অফলাইন রিপোজিটরি নিয়ে কাজ করা

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 লিখুন। এতে করে আপনার রিপোজিটরিটি গিট ব্যবহারের উপযোগি হবে এবং .git নামে একটি লুকানো ফোল্ডার তৈরি হবে। আপনার রিপোজিটরিতে .git ফোল্ডারটি তৈরি না থাকলে আপনি গিটের কমান্ডগুলো কাজে লাগাতে পারবেন না।

3. আপনার রিপোজিটরিটি এখন কাজের উপযুক্ত হয়েছে। আপনি এখন এতে যে কোন ধরনে ফাইল, ফোল্ডার তৈরি, সম্পাদনা ও মুছে ফেলতে পারবেন।

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

```
git add *
```

কমান্ড 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
```

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

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

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



## গিট : বিদ্যমান রিমোট রিপোজিটরি নিয়ে কাজ করা

অনলাইনে গিট রিপোজিটরি সংরক্ষণ করে রাখার জন্য কিছু সার্ভার রয়েছে। যাদের মধ্যে [GitHub](#), [BitBucket](#), [GitLab](#) অন্যতম। এই সব সার্ভারে সংরক্ষিত রিপোজিটরিগুলোকে রিমোট রিপোজিটরি বলা হয়। আপনি সাইটগুলোতে প্রবেশ করে আপনার নিজস্ব রিপোজিটরি তৈরি করতে পারবেন। একটি রিমোট রিপোজিটরি তৈরি করার পর আপনাকে সাধারণত আপনার লোকাল কম্পিউটার থেকেই সেগুলোকে সম্পাদনা করতে হবে। সম্পাদনা শেষে আবার রিমোট রিপোজিটরিতে আপলোড করে দিতে হবে। চলুন দেখা যাক, গিট ব্যবহার করে কিভাবে আমরা একটি রিমোট রিপোজিটরিকে সম্পাদনা করতে পারি।

(আমরা এখানে bitbucket ও ওয়েব ডেভেলপমেন্ট এর রিপোজিটরি নিয়ে আলোচনা করবো)

1. প্রথমেই আপনার রিমোট রিপোজিটরিকে আপনার নিজস্ব কম্পিউটারে নিয়ে আসতে হবে। এজন্য আপনাকে `cd` কমান্ড ব্যবহার করে আপনার কম্পিউটারের `localhost` এ যেতে হবে। `localhost` এর জন্য আপনাকে `htdocs` অথবা `www` ফোল্ডারে প্রবেশ করতে হবে।
2. ব্রাউজারের মাধ্যমে আপনার রিমোট রিপোজিটরিতে প্রবেশ করুন। সেখানে `clone` বাটনে ক্লিক করার পর সেখানে একটি কমান্ড দেখতে পাবেন। `clone` কমান্ডের দুইটি অপশন থাকে। একটি হচ্ছে `https` এবং অন্যটি `ssh`। `ssh` কমান্ড নির্বাচন করলে আপনার কম্পিউটারে একটি `ssh key` তৈরি করতে হবে। পরে সেটি আপনার সার্ভারে সংরক্ষণ করতে হবে। কিন্তু `https` নির্বাচন করলে কোন `key` তৈরি করতে হবে না। চলুন দেখে নেয়া যাক, কিভাবে `ssh key` তৈরি করতে হয়-
  - terminal এ `ssh-keygen` কমান্ডটি লিখুন। এতে করে আপনার কম্পিউটারের home ফোল্ডারে লুকানো একটি `.ssh` ফোল্ডার তৈরি হবে এবং একটি `id_rsa.pub` ফাইল তৈরি হবে। `id_rsa.pub` ফাইলটি খুলুন এবং এর ভিতরের লেখাগুলো copy করুন।
  - আপনার সার্ভার account এ প্রবেশ করুন। সেখান থেকে `manage account` লিংকে(link) প্রবেশ করুন। এরপর সেখান থেকে `ssh key` লিংকে প্রবেশ করে আপনার key টি paste করুন। এরপর সংরক্ষণ করে বের হয়ে আসুন।
3. সার্ভার একাউন্ট(account) থেকে `clone` কমান্ডটি copy করে terminal এ paste করুন।
4. আপনার `clone` কমান্ডটি যদি `https` এর হয় তাহলে আপনার কাছে আপনার একাউন্ট এর user name এবং password চাইবে। সঠিক তথ্য দিলে আপনার সার্ভারের রিপোজিটরিটি আপনার নিজস্ব কম্পিউটারে ক্লোন(clone) হয়ে যাবে। আপনার `localhost` এ প্রবেশ করলে সার্ভারের রিপোজিটরির নামে একটি ফোল্ডার দেখবেন। এটাই আপনার নিজস্ব কম্পিউটারে রিমোট রিপোজিটরির ক্লোন।
5. এখন অফলাইনে থেকেই আপনার রিপোজিটরিতে কাজ করুন। এরপর আপনি যখন আপনার রিমোট রিপোজিটরিতে আপনার কোড(code) সংযুক্ত করতে চাইবেন তখন আপনাকে কিছু কমান্ড ব্যবহার করতে হবে। সেগুলো নিচে দেয়া হলো।
  - আপনার রিপোজিটরি প্রত্যেকবার স্থিতিশীল(stable) অবস্থায় পৌছলে সেগুলো Staging Area তে পাঠাতে হবে। Staging area তে পাঠানোর মানে হচ্ছে আপনি যে যে ফাইলগুলোকে নিশ্চিতভাবে

রিপোজিটরিতে অগ্ৰভুক্ত করবেন তার তালিকা তৈরি করা। সে জন্য আপনাকে `git add` কমান্ড ব্যবহার করতে হবে। যদি আপনি বর্তমান সবগুলো ফাইলকে Staging area তে সংযুক্ত করতে চান তবে আপনাকে

```
git add *
```

কমান্ড 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
```

- আপনার staging area তে সংযুক্ত করা ফাইলগুলোকে মূল রিপোজিটরির সাথে নিশ্চিতভাবে সংযুক্ত করা। সে জন্য আপনাকে `git commit` কমান্ডটি ব্যবহার করতে হবে। আপনাকে প্রত্যেক commit এর সাথে একটি বার্তা সংযুক্ত করে দিতে হবে। তখন আপনার সম্পূর্ণ কমান্ডটি হবে-

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

- এখন কমিট(commit) করা ফাইলগুলোকে রিমোট রিপোজিটরিতে সংযুক্ত করতে আপনাকে `git push` কমান্ডটি ব্যবহার করতে হবে। আপনি কোন ব্রাঞ্চ(branch (branch সম্পর্কে অন্য টিউটরিয়ালে আলোচনা করা হবে))-এ ফাইল সংযুক্ত তাও বলে দিতে হবে। আমরা আমাদের ফাইল `master` branch এ সংযুক্ত করবো। সম্পূর্ণ কমান্ডটি নিচে দেয়া হলো।

```
git push origin master
```

এখানে origin হচ্ছে রিমোট রিপোজিটরির নাম। যে রিমোট রিপোজিটরি থেকে ফাইল ক্লোন করা হয় তার নাম সাধারণত origin দেয়া থাকে।

## গিট : একই রিপোজিটরিতে একাধিক ডেভেলপার

সাধারণত আমরা যখন কোন প্রজেক্ট বা রিপোজিটরি নিয়ে একটি টিম(team) কাজ করি তখন প্রজেক্ট ব্যবস্থাপনাটা একটু জটিল হয়ে যায়। বিশেষ করে একই ফাইলে একাধিক ডেভেলপার কাজ করা। এই সমস্যার সমাধানের জন্য গিট যে পদ্ধতি ব্যবহার করেছে তা এক কথায় অসাধারণ। কোন কাজ অথবা কোড না হারিয়েই একই ফাইলে কাজ করতে পারবেন অসংখ্য ডেভেলপার। একটি রিপোজিটরি নিয়ে অসংখ্য ডেভেলপার বিচ্ছিন্নভাবে কাজ করলেও সবাই সংযুক্ত থাকতে পারবেন গিট এর মাধ্যমে। কথাটা শুনে খাপছাড়া মনে হলেও এটাই সত্যি। তাহলে দেরি না করে চলুন দেখা যাক কিভাবে একই রিপোজিটরিতে অসংখ্য ডেভেলপার কাজ করতে পারবেন। আমরা এখানে যে পদ্ধতি নিয়ে আলোচনা করবো তা অধিকাংশ ওপেনসোর্স(opensource) রিপোজিটরিতে ব্যবহার করা হয়।

- প্রথমেই আমাদেরকে এমন একটি রিমোট রিপোজিটরি তৈরি করতে হবে যা কেন্দ্রীয় রিপোজিটরি হিসেবে ব্যবহার করা হবে। আপনি যদি বিনামূল্যে github(একটি গিট রিপোজিটরি সার্ভার) ব্যবহার করেন তাহলে যে কেউ আপনার রিপোজিটরিতে read access পাবে। কিন্তু bitbucket ব্যবহার করলে আপনি যদি রিপোজিটরি public না করে দেন তাহলে কেউ দেখতে পাবে না। চাইলে আপনি private রিপোজিটরিতেও আপনার পছন্দনীয় ব্যক্তিকে read, write অথবা admin access দিতে পারবেন। তাই আপনি যে সকল ডেভেলপারকে আপনার রিপোজিটরিতে সংযুক্ত করবেন তাদেরকে read access দিন।
- ফোর্ক(fork): admin ব্যতীত যে সকল ডেভেলপার একই রিপোজিটরিতে কাজ করবেন তাদেরকে কেন্দ্রীয় বা মূল রিপোজিটরিতে Fork করতে হবে। ফোর্ক মানে হচ্ছে ডেভেলপারের সার্ভার একাউন্টে(account) মূল রিপোজিটরির মত একটি স্বতন্ত্র রিপোজিটরি তৈরি করা। ফোর্ক করার পর ডেভেলপার যে রিপোজিটরি তৈরি করবেন তা একান্তই তার নিজস্ব। এতে মূল রিপোজিটরির সাথে দৃশ্যত কোন সম্পর্ক থাকবে না। পরবর্তীতে আমরা চাইলে মূল রিপোজিটরির সাথে সংযোগ দিতে পারবো। একটি রিপোজিটরি ফোর্ক করা খুবই সহজ-
  - কেন্দ্রীয় রিপোজিটরিতে প্রবেশ করুন – Fork বাটনে(button) ক্লিক করুন – প্রয়োজনীয় তথ্য পূরণ করে submit করুন ডেভেলপার এখন একটি নিজস্ব রিপোজিটরির মালিক। :)
  - এখন ডেভেলপার তার নিজস্ব রিপোজিটরিতে নতুন যে কোন feature নিয়ে কাজ করতে পারবেন। করতে পারবেন যে কোন ধরনের সম্পাদনা। কাজতো হলো কিন্তু এবার এত পরিশ্রমের কাজ কিভাবে মূল রিপোজিটরির সাথে সংযুক্ত করবো? খুবই সহজ–
  - Compare: কোন কোন রিমোট রিপোজিটরি সার্ভারে আপনার করা কাজকে compare করার সুযোগ রাখা হয়েছে। চাইলে আপনি দেখে নিতে পারেন মূল রিপোজিটরির সাথে আপনি কি সংযোজন-বিয়োজন করেছেন। compare করার জন্য ডেভেলপার তার রিপোজিটরি পাতায় compare বাটনে ক্লিক করুন। দেখে নিন আপনার কাজের বর্ণনা।
    - Pull Request: আপনার কাজগুলো মূল রিপোজিটরিতে অন্তর্ভুক্ত করার জন্য মূল রিপোজিটরিতে একটি Pull Request পাঠাতে হবে। এতে করে মূল রিপোজিটরিতে ডেভেলপারের কাজ সংযুক্ত করার জন্য একটি বার্তা প্রেরণ করা হবে। admin চাইলে ডেভেলপারের কাজ গ্রহন অথবা বাতিল করতে পারবেন। ডেভেলপার শুধু তার কাজ গ্রহন করা হয়েছে কি বাতিল করা হয়েছে এই মর্মে একটি বার্তা পাবেন। Pull Request পাঠানোর জন্য ডেভেলপার তার রিপোজিটরি পাতায় pull request বাটনে ক্লিক করুন।

- এখন এডমিন তার মূল রিপোজিটরিতে একটি নতুন Pull Request পাবেন। কাজ পর্যবেক্ষণের পর তা গ্রহন করার মত হলে মূল রিপোজিটরির সাথে Merge করে নেবেন। উপরের যে ধাপগুলো বর্ণনা করা হয়েছে সেগুলো সম্পূর্ণ সার্ভারভিত্তিক। আপনি চাইলে সার্ভারের ওয়েব সাইটে প্রবেশ না করে গিট কমান্ড এর মাধ্যমে সরাসরি আপনার নিজস্ব কম্পিউটার থেকে ডেভেলপারদের কাজ পর্যবেক্ষণ করতে পারবেন। সেজন্য আপনাকে জানতে হবে কিভাবে আপনি একাধিক রিমোট রিপোজিটরির সাথে সংযুক্ত হবেন। আমরা আগামী টিউটরিয়ালে এ বিষয়ে আলোচনা করবো, ইনশাআল্লাহ।

## গিট : একাধিক রিমোট রিপোজিটরির সাথে সংযোগ

একই রিপোজিটরিতে একাধিক ডেভেলপার কাজ করলে একজন ডেভেলপার অন্য একজন ডেভেলপারের কাজ pull করার প্রয়োজন হতে পারে। এছাড়াও admin যখন কোন ডেভেলপারের কাজ পর্যবেক্ষণ করবেন তখন ঐ ডেভেলপারের রিপোজিটরির সাথে admin এর রিপোজিটরির সংযোগ দিতে হবে। চলুন দেখে নেয়া যাক কিভাবে রিমোট রিপোজিটরির সাথে সংযোগ দেয়া যায়। রিমোট রিপোজিটরির সাথে সংযোগ দেয়ার জন্য আমাদেরকে git remote কমান্ডটি ব্যবহার করতে হবে। সম্পূর্ণ কমান্ডটি নিচে দেয়া হলো-

```
$ git remote add <name> <remote-repo-url>
```

এখানে name এর জায়গায় আপনি রিমোট রিপোজিটরিকে যে নামে ডাকতে চান সে নাম হবে। remote-repo-url এর জায়গায় রিমোট রিপোজিটরির HTTPS/SSH clone address দিতে হবে। ধরুন আমরা একটি কেন্দ্রীয় রিপোজিটরি তৈরি করলাম যার নাম হচ্ছে rms এবং রিপোজিটরির HTTPS clone address হচ্ছে <https://bitbucket.org/precursortechnology/rms.git>। এখন এতে যতজন ডেভেলপার কাজ করবে সবাই এই রিপোজিটরিটি fork করতে হবে। এখন ডেভেলপাররা যদি তাদের রিপোজিটরিকে কেন্দ্রীয় রিপোজিটরির (<https://bitbucket.org/precursortechnology/rms.git>) সাথে সংযোগ দিতে চান তাহলে তাদেরকে লিখতে হবে

```
$ git remote add upstream https://bitbucket.org/precursortechnology/rms.git
```

এখানে আমরা কেন্দ্রীয় রিপোজিটরিকে চেনার জন্য *upstream* নাম ব্যবহার করেছি। আপনি চাইলে অন্য যে কোন নাম ব্যবহার করতে পারেন। এখন নিশ্চিত হওয়ার জন্য *upstream* এর মান দেখতে পারেন।

```
$ git remote show upstream
```

একটি রিপোজিটরি চাইলে একাধিক রিমোট রিপোজিটরির সাথে সংযুক্ত হতে পারবে। সবগুলো remote এর মান একসাথে দেখার কমান্ড -

```
$ git remote -v
```

এখন কোন ডেভেলপার যদি কেন্দ্রীয় রিপোজিটরির ( *upstream* ) তথ্য pull করতে চান তাহলে তাকে লিখতে হবে -

```
$ git pull upstream master
```

pull করা মানে হচ্ছে ঐ রিপোজিটরির সর্বশেষ তথ্যগুলো আপনার রিপোজিটরির সাথে একীভূত করে নেয়া। pull কমান্ড দিয়ে মূলত দুটি কাজ করা হয়। প্রথমে রিমোট রিপোজিটরির তথ্যগুলো fetch করে নেয়া হয় এবং পরে সেগুলো লোকাল রিপোজিটরির সাথে merge করা হয়। উপরের কমান্ড দিয়ে আমরা upstream নামক রিমোট

রিপোজিটরির `master` branch এর সব তথ্যগুলো/commits লোকাল repository এর বর্তমান ব্রাঞ্চ এ merge করে নিলাম।

এখন আপনার নিজের রিপোজিটরি (*origin*) তে *upstream* এর সব তথ্যগুলো পাঠাতে হলে লিখতে হবে -

```
$ git push origin HEAD    # এখানে HEAD = বর্তমান লোকাল ব্রাঞ্চ এর নাম
```

যেকোন রিমোট remove করার কমান্ড `git remote rm <remote-name>` . সুতরাং, `upstream` রিমোট remove করতে হলে কমান্ড হবে -

```
$ git remote rm upstream
```

কোন রিমোট rename করার কমান্ড `git remote rename <present-name> <new-name>` . সুতরাং, `upstream` রিমোট কে `origin2` করতে হলে কমান্ড হবে -

```
$ git remote rename upstream origin2
```



## গিট-ফ্লুও পরিচিতি(Git-flow introduction)

গিট-ফ্লুও আসলে একটা শাখা(branching) মডেল, যেটা Vincent Driessen নামের একজন লোক সর্বপ্রথম বানিয়েছেন। এটা মলুত গিটের উপর নির্ভর করে বানানো হয়েছে। এর সবচেয়ে বড় সুবিধা হল একাধিক ডেভেলপার একসাথে একটা বড় প্রোজেক্ট নির্ভেজাল ভাবে কাজ করতে পারে।

কিন্তু, গিট-ফ্লুও নিয়ে কাজ করতে হলে আগে, গিটের উপর ভাল দখল থাকতে হবে। গিটের শাখা (branching) কিভাবে কাজ করে সেটা ভাল করে জানতে হবে। আর এই ক্ষেত্রে, কমান্ড লাইনে (command line) এ কাজ করা সবার আগে আবশ্যিক।

# গিট-ফ্লুও ইন্সটলেশন এবং কনফিগারেশন

## গিট-ফ্লুও ইন্সটলেশন

- ডেবিয়ান নির্ভরশীল লিনাক্স সিস্টেমে গিট-ফ্লুও ( `git-flow` ) ইন্সটল করার জন্য নিচের কমান্ডটি রান করুন

```
$ sudo apt-get install git-flow
```

- ফেডরা নির্ভরশীল লিনাক্স সিস্টেমে গিট-ফ্লুও ( `git-flow` ) ইন্সটল করার জন্য নিচের কমান্ডটি রান করুন

```
$ yum install gitflow
```

- অ্যাপল নির্ভরশীল ম্যাক সিস্টেমে গিট-ফ্লুও ইন্সটল করার জন্য নিচের কমান্ড রান করুন

```
$ brew install git-flow
```

- উইন্ডোজ সিস্টেমে গিট-ফ্লুও ইন্সটল করার জন্য সবার আগে `cygwin` ইন্সটল করুন। এটা একটা উইন্ডোজ সিস্টেমে, লিনাক্স এনভায়রোমেন্ট তৈরি করে। এরপর নিচের কমান্ডগুলো রান করুনঃ

```
$ wget -q -O - --no-check-certificate  
https://github.com/nvie/gitflow/raw/develop/contrib/gitflow-installer.sh | bash
```

- যদি এরপর `git-flow init` রান করার পর কেউ "flags: FATAL unable to determine getopt version" এই error দেখতে পায়, তাহলে `util-linux package` ইন্সটল করতে হবে `Cygwin` দিয়ে।

আর বিস্তারিত জানতে উক্ত লিঙ্কে ক্লিক করুনঃ [উইন্ডোজ সিস্টেমে ইন্সটল গিট-ফ্লুও](#)

## গিট-ফ্লুও কনফিগার

গিট-ফ্লুও কনফিগারেশন করার জন্য সবার আগে একটা গিট রিপোজিটরি থাকতে হবে। আমরা যখন কোন একটা প্রোজেক্টে `git init` করি, তখন সমস্ত কাজ গুলো একটা ডিফল্ট শাখা "মাস্টার(master)" এ জমা হতে থাকে।

বিরক্ত লাগছে... চলুন আমরা এবার নিজে করিঃ

১। আগে আমরা একটা ফোল্ডার বানাবঃ

```
mkdir git-flow-test
```

২। এবার আমি একটা ফাইল বানাব `git-flow-test` ফোল্ডারে ডুকে।

```
cd git-flow-test
```

```
touch helloworld.txt
```

৩। এবার আমরা সবার আগে এখানে গিট (`git`) রিপোজিটোরি বানাব। এজন্য আমরা এখানে নিচের কমান্ড চালবঃ

```
git init
```

৪। এখন আমরা আমাদের চেঞ্জগুলো গিটে যোগ করে একটা কমিট করব। তার জন্য নিচের কমান্ডগুলো চালাবঃ

```
git add .    git commit -m "initali commit with first file"
```

৫। এখন আমাদের সবগুলো পরিবর্তন গিটের প্রাথমিক ব্রাঞ্চ `master` এ আছে। এখন আমাদের গিট রিপোজিটরি `git-flow` বানানোর জন্য তৈরি। গিট-ফ্লুও বানানোর জন্য এখন আমরা নিছের কমান্ডগুলো রান করবঃ

# গিট-ফ্লো ফিচার

গিট-ফ্লো ফিচার শুরু করার আগে আমরা কিছু জিনিস বানাব। যেমন আমরা কোন প্রোজেক্ট করার আগে কোন একটা প্রোজেক্ট ম্যানেজমেন্ট সিস্টেমে, প্রোজেক্ট রিলেটেড সব ফিচার (feature), বাগ (bug) ইত্যাদি আমরা বিস্তারিত লেখে রাখি। এরপর আমরা বিভিন্ন ডেভেলপারকে আমরা তা assign করে দেই।

ধরি আমাদের প্রোজেক্টের নামে হেলো "গিট-ফ্লো (Hello Git-Flow)" আমাদের কাছে নিম্নলিখিত ফিচারগুলো আছেঃ

- HGT-01: Bootstrap Hello Git-Flow project
- HGT-02: Create login with Username and Password
- HGT-03: Create user signup
- HGT-04: Create forget Password
- HGT-05: Homepage with login and Signup option

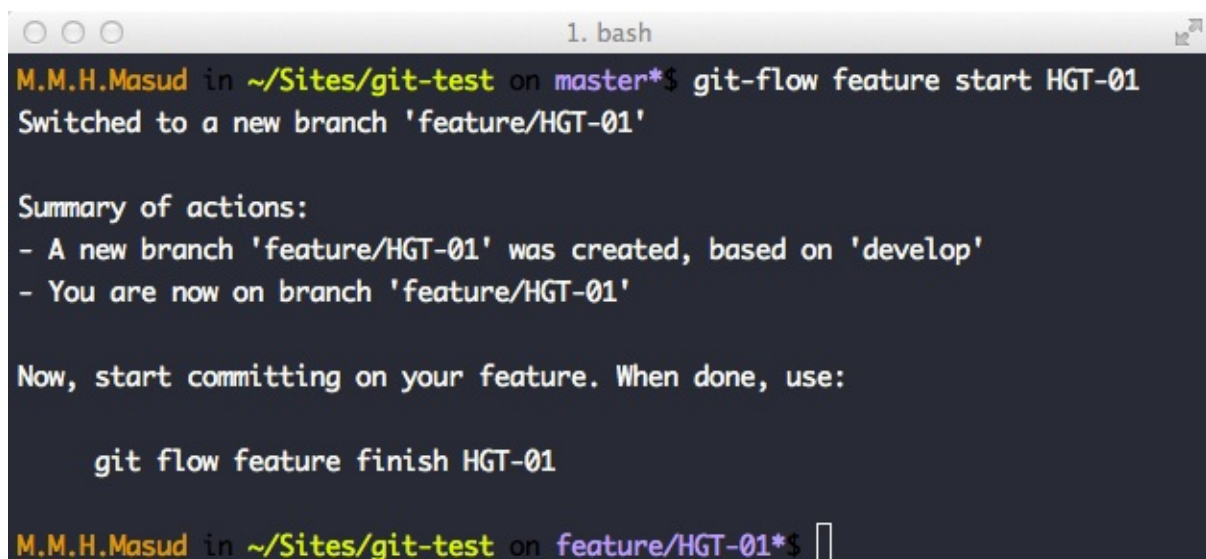
এখন, আমরা একটা একটা করে ফিচারে কাজ শুরু করব, গিট-ফ্লো (git-flow) ফিচার ব্যবহার করে।

ধাপ-০১

সবার আগে আমরা HGT-01 ফিচার নিয়ে কাজ শুরু করব। এজন্য আমরা কমান্ড লিখব

```
git-flow feature start HGT-01
```

এরপর আমরা নিচের ছবির মত একটা মেসেজ দেখতে পাব,



```
1. bash
M.M.H.Masud in ~/Sites/git-test on master*$ git-flow feature start HGT-01
Switched to a new branch 'feature/HGT-01'

Summary of actions:
- A new branch 'feature/HGT-01' was created, based on 'develop'
- You are now on branch 'feature/HGT-01'

Now, start committing on your feature. When done, use:

    git flow feature finish HGT-01

M.M.H.Masud in ~/Sites/git-test on feature/HGT-01*$
```

এখানে আমরা যেটা দেখতে পারছি, তা হোল, সবার আগে develop নামে যে ব্রাঞ্চ (branch) আছে, তার উপর নির্ভর করে একটা নতুন ব্রাঞ্চ (branch) feature/HGT-01 তৈরি হয়েছে। এরপর স্বয়ংক্রিয় ভাবে, ওই ব্রাঞ্চ (branch) পয়েন্ট করে ফেলেছে।

ধাপ-০২

এখন আমরা ফিচার ডেভেলপমেন্টের কাজ করব এবং ক্রমান্বয়ে কমিট করব, যেভাবে আমরা গিটে (git) কমিট করতাম। আমরা যত পরিবর্তন করব, সবই আমাদের বর্তমান ফিচার (feature) ব্রাঞ্চ এ যোগ হতে থাকবে। ধরি, একটা নতুন ফাইল বানাব কারেন্ট ফিচারের জন্য

```
touch project-config.php
```

এবার এই ফাইলে কিছু পরিবর্তন করব, মানে কিছু লেখা যোগকরব

```
nano project-config.php
```

এবার আমরা ফাইলটা গিটে যোগ করব এবং গিটে কমিট করব

```
git add project-config.php  
git commit -m "project configuration file is added"
```

ধাপ-০৩

এবার ধরে নিলাম আমাদের ফিচার feature ডেভেলপমেন্টের কাজ শেষ। এখন আমরা গিট-ফ্লুও দিয়ে আমাদের ফিচার কমপ্লিট করব।

```
git-flow feature finish HGT-01
```

এই কমান্ড চালানোর পর আমরা নিচের ছবির মত একটা মেসেজ দেখতে পারব

```

1. bash
bash bash
M.M.H.Masud in ~/Sites/git-test on feature/HGT-01*$ touch project-config.php
M.M.H.Masud in ~/Sites/git-test on feature/HGT-01*$ nano project-config.php
M.M.H.Masud in ~/Sites/git-test on feature/HGT-01*$ git add project-config.php
M.M.H.Masud in ~/Sites/git-test on feature/HGT-01*$ git commit -m "project config is added"
[feature/HGT-01 a3c83fc] project config is added
1 file changed, 1 insertion(+)
create mode 100644 project-config.php
M.M.H.Masud in ~/Sites/git-test on feature/HGT-01*$ git-flow feature finish HGT-01
Switched to branch 'develop'
Your branch is up-to-date with 'origin/develop'.
Updating 96ce5bc..a3c83fc
Fast-forward
 project-config.php | 1 +
1 file changed, 1 insertion(+)
create mode 100644 project-config.php
Deleted branch feature/HGT-01 (was a3c83fc).

Summary of actions:
- The feature branch 'feature/HGT-01' was merged into 'develop'
- Feature branch 'feature/HGT-01' has been removed
- You are now on branch 'develop'

M.M.H.Masud in ~/Sites/git-test on develop*$ 

```

গিট-ফ্লো ফিচার পারলিশ

আমরা যদি মনে করি, কন একটা ফিচার ব্রাঞ্চে অন্য কেউ কাজ করবে, তাহলে আমরা চাইলে গিটে আপ করে দিতে পারি। এজন্য আমরা নিচের কমান্ডটা রান করবঃ

```
git-flow feature publish HGT-01
```

এই কমান্ড চালানোর পর নিচের মত একটা স্ক্রীন দেখা যাবে।

```
1. bash
bash
M.M.H.Masud in ~/Sites/git-test on feature/HGT-01*$ git-flow feature publish HGT-01
Saving password to keychain failed
Identity added: /Users/Masud/.ssh/id_rsa (/Users/Masud/.ssh/id_rsa)
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 312 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@bitbucket.org:masudiiuc/git-test.git
 * [new branch]      feature/HGT-01 -> feature/HGT-01
Already on 'feature/HGT-01'
Your branch is up-to-date with 'origin/feature/HGT-01'.

Summary of actions:
- A new remote branch 'feature/HGT-01' was created
- The local branch 'feature/HGT-01' was configured to track the remote branch
- You are now on branch 'feature/HGT-01'

M.M.H.Masud in ~/Sites/git-test on feature/HGT-01*$
```

## গিট-ফ্লুও রিলিস

গিট-ফ্লুও এর আরেকটি প্রয়োজনীয় কমান্ড হোল গিট-ফ্লুও রিলিস। এটা দিয়ে আমরা মূলত একটা development branch থেকে production সার্ভারে দেয়ার মত একটা রিলিস তৈরি করি। এটা দিয়ে আমরা খুব সহজে, একটা রিলিস এর নাম্বার দিতে পারি, যেন খুব সহজে আমরা আবার পরবর্তী কোন রিলিএসে ব্যাক করতে পারি।

আমরা যদি গিট-ফ্লুও রিলিসের সিনট্যাক্সজানতে চাই, তাহলে আমরা কমান্ড লাইনে গিয়ে নিচের কমান্ড টাইপ করলে হবেঃ

```
Git-flow release (press enter)
```

এবার আমরা নিচের মত একটা স্ক্রীন দেখতে পাব।

```

1. bash
bash bash
M.M.H.Masud in ~/Sites/git-test on develop*$ git-flow release
No release branches exist.

You can start a new release branch:

git flow release start <name> [<base>]
M.M.H.Masud in ~/Sites/git-test on develop*$ 

```

এখানে তিনটা টার্ম আছে, যা জানতে হবেঃ

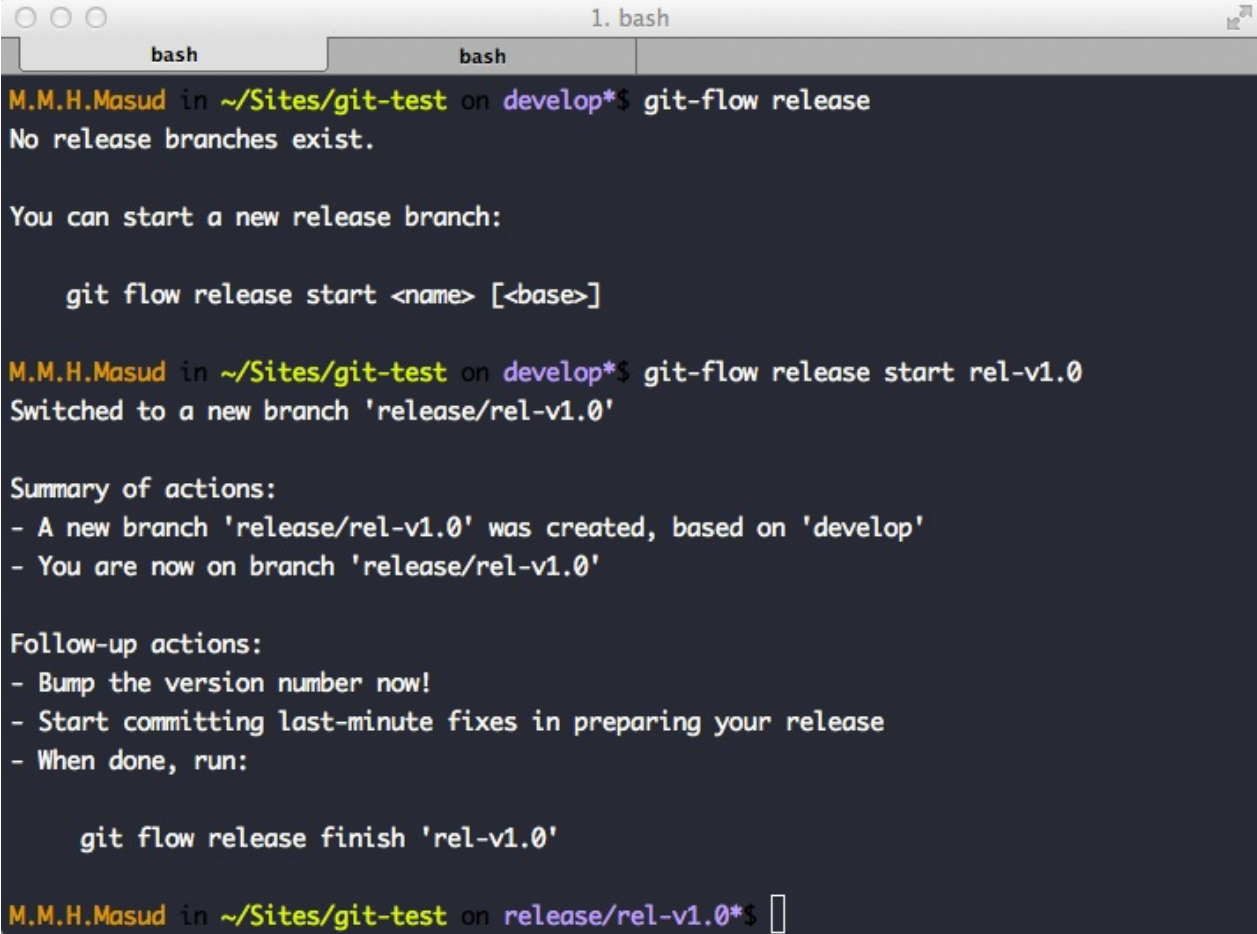
1. <name> : রিলিসের নাম। যেমন: rel-v1.0
2. [<base>]: এটা একটা ব্রাঞ্চের নাম। মানে আমরা যদি development branch থেকে ব্রাঞ্চ তৈরি না করে, অন্য কোন:

এবার আমরা একটা রিলিস তৈরি করব। আমরা ধরে নিচ্ছি, আমাদের ব্রাঞ্চ development। তাহলে আর দেরি না করে নিচের কমান্ডটা রান করি ៖



```
git-flow release start rel-v1.0
```

এখন আমরা নিচের মত একটা স্ক্রীন দেখতে পাবোঃ



```

1. bash
bash
M.M.H.Masud in ~/Sites/git-test on develop*$ git-flow release
No release branches exist.

You can start a new release branch:

git flow release start <name> [<base>]

M.M.H.Masud in ~/Sites/git-test on develop*$ git-flow release start rel-v1.0
Switched to a new branch 'release/rel-v1.0'

Summary of actions:
- A new branch 'release/rel-v1.0' was created, based on 'develop'
- You are now on branch 'release/rel-v1.0'

Follow-up actions:
- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

git flow release finish 'rel-v1.0'

M.M.H.Masud in ~/Sites/git-test on release/rel-v1.0*$ 

```

আমরা এখন দেখতে পেলাম একটা নতুন ব্রাঞ্চ rel-v1.0নামের একটা নতুন ব্রাঞ্চ তৈরি হয়েছে।

আমরা এখন জেতা করতে পারি, এই ব্রাঞ্চটাকে একটা টেস্টিং সার্ভারে দিয়ে দেতে পারি। একদল QA দলকে বলতে পারি, টেস্টিং করতে। যখন, টেস্টিং শেষ হবে, আমরা রিলিসটাকে finish করে দিতে পারি।

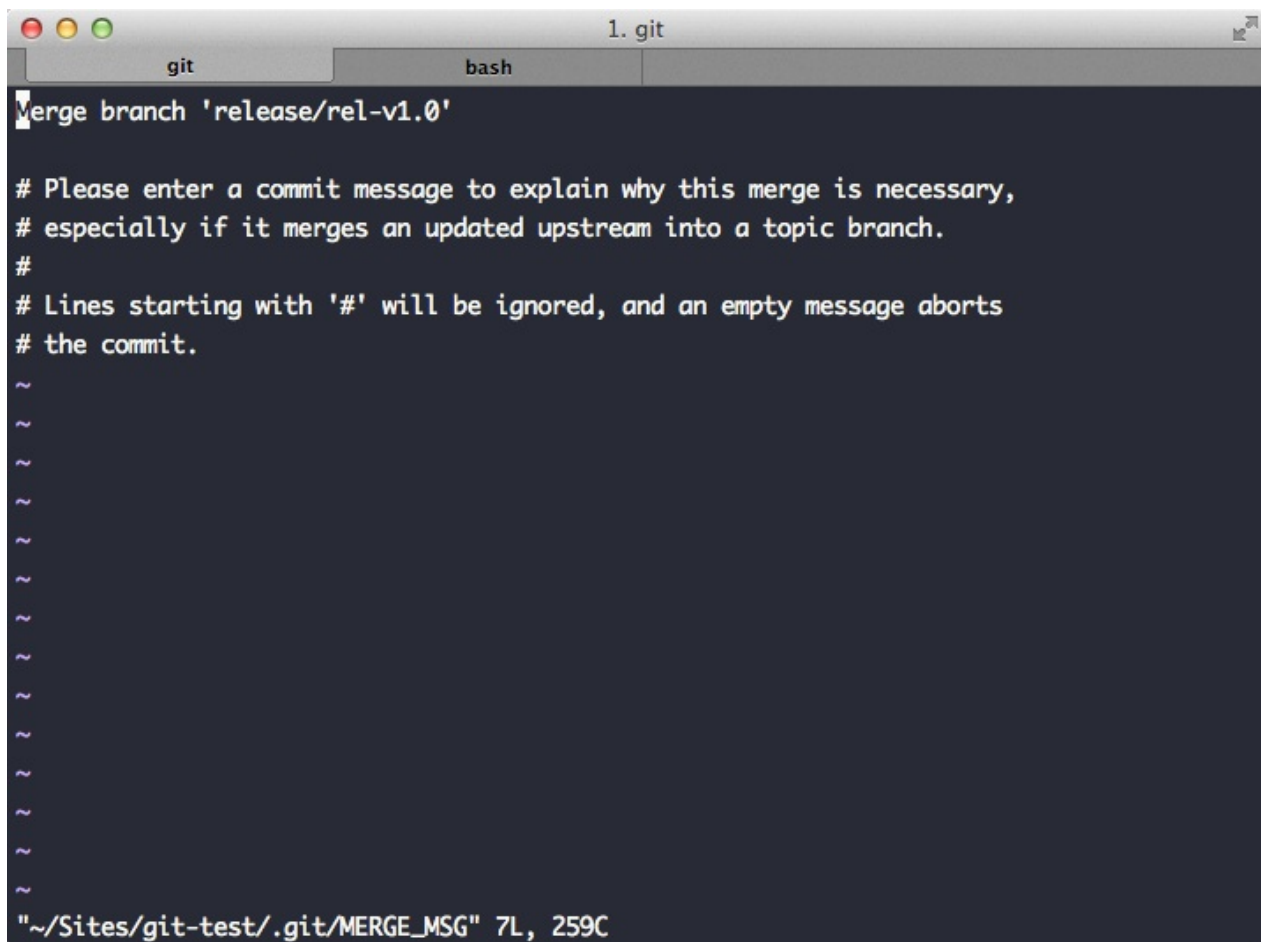
এখন জানতে হবে, রিলিস শেষ(finish) করলে কি হয় :ঃ আমরা যখনই কোন রিলিস শেষ করব, গিট-ফ্লো স্বয়ংক্রিয়ভাবে রিলিস ব্রাঞ্চের সকল কোড development branch এবং production branch (master এখানে production branch) এ merge করে দেয়।

এখন আর কথা না বাড়িয়ে, কমান্ডটা রান করে দেখিঃ

```
git-flow release finish rel-v1.0
```

এখন আমরা নিচের মত একটা স্ক্রীন দেখতে পাবোঃ

গিট মারজ মেসেজ



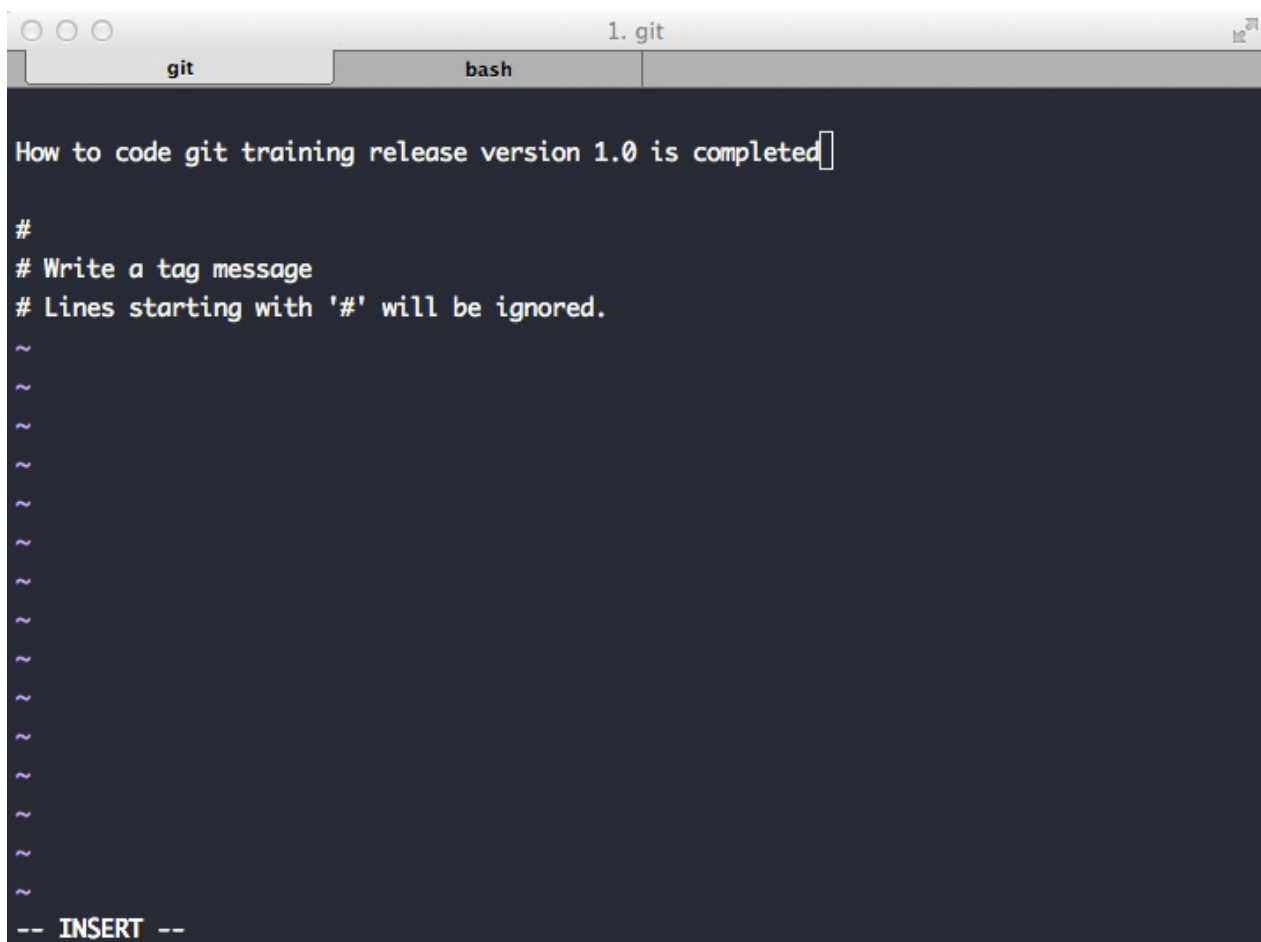
```

1. git
git bash
Merge branch 'release/rel-v1.0'

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~/Sites/git-test/.git/MERGE_MSG" 7L, 259C

```

## গিট ট্যাগ মেসেজ



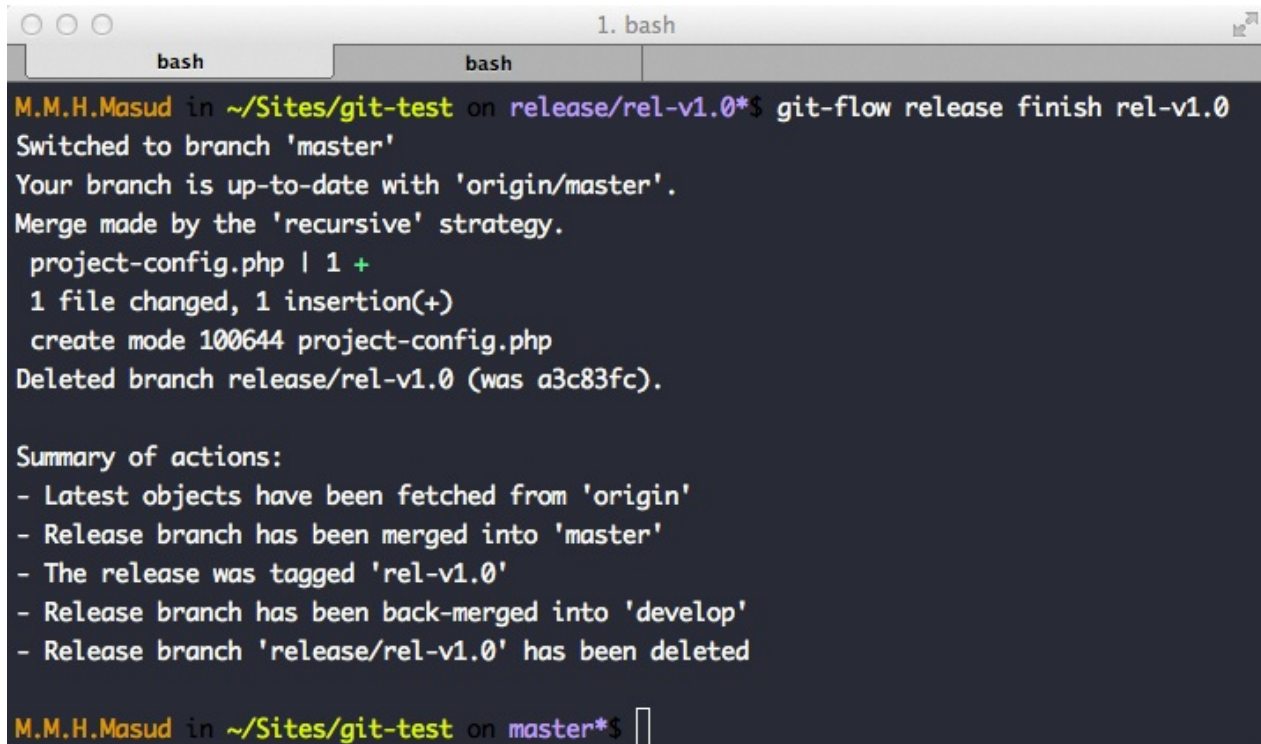
```

1. git
git bash
How to code git training release version 1.0 is completed

#
# Write a tag message
# Lines starting with '#' will be ignored.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --

```

## গিট ফিনিশ শেষ



```

M.M.H.Masud in ~/Sites/git-test on release/rel-v1.0*$ git-flow release finish rel-v1.0
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
Merge made by the 'recursive' strategy.
 project-config.php | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 project-config.php
Deleted branch release/rel-v1.0 (was a3c83fc).

Summary of actions:
- Latest objects have been fetched from 'origin'
- Release branch has been merged into 'master'
- The release was tagged 'rel-v1.0'
- Release branch has been back-merged into 'develop'
- Release branch 'release/rel-v1.0' has been deleted

M.M.H.Masud in ~/Sites/git-test on master*$ █

```

এখন আমরা একটু খেয়াল করে **summary of Actions** এর দিকে খেয়াল করে দেখি। এখানে আসলে গিট-ফ্লুও কি কি কাজ করেছে, তার একটা সারাংশ তুলে ধরেছে। এবং সব শেষে মাস্টার ব্রাঞ্চ এ শিফট করেছে। মানে, rel-v1.0 এর সকল কোড এখন (master) মাস্টার ব্রাঞ্চেও আছে এবং development ব্রাঞ্চে ও আছে।

## গিট-ফ্লও হটফিক্স