Git

> Author: Dipsy Wong (dipsywong98)

1. What is Git

Git is a "Google drive" for coders with version control, it can help the collaboration between coders. Github is just one of the "cloud service provider" and we are going to use it. ~~So if you know how to use Google Drive, you know how to use git.~~

~~You can also use git locally without a cloud (actually it is called a remote) for your own version control.~~

2. How it works

The most important concept is "commit". A commit is actually a record of all your codes. It can act as a rollback point when you do something sucks. All the versions are just all the commits. Every time you call commit, git takes a photo of you code, and records it as a newer version.

The other important concept is "push" and "pull", which push is uploading your local new version (new commit) to github, and pull is downloading the new version in github to your computer. The details will be explained later.

3. Getting Started

- Create an account in github https://github.com/
  - You may also get a student pack for your account so that you can have infinite private repo (PS: use @ust.hk to register)
    https://education.github.com/pack
- Install git https://git-scm.com/downloads (remember to check git bash here, extremely convenient to use)
- Good link for visualization when learning git:
  - https://learngitbranching.js.org/?NODEMO
  - https://onlywei.github.io/explain-git-with-d3/#freeplay
- You can also google some good software which can visualize your git repo

4. Set your Account for Git (Only need to do it once)

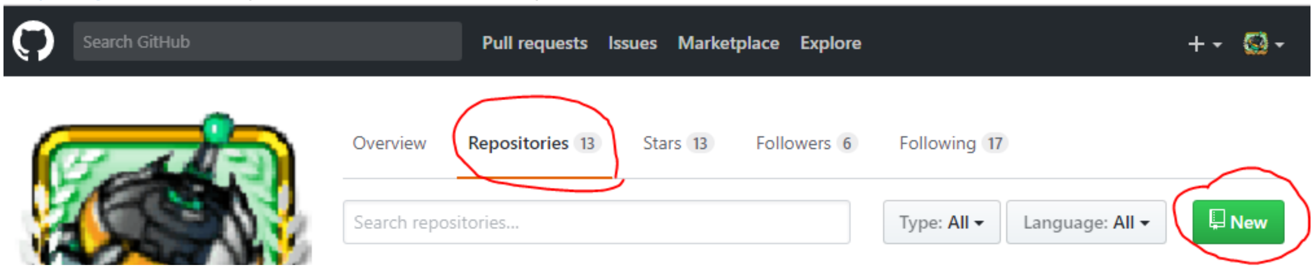
Open git bash, and execute:

```
git config --global user.email "your@email.com"
git config --global user.name "Your Name"
```

5. Create Repo on GitHub

A repo(repository) is a programming project organized by git, and stored on some git server, which is GitHub.

To create a repo, login GitHub and click me. Alternatively, you can go to your profile page https://github.com/{your username}?tab=repositories and click the "new" button.



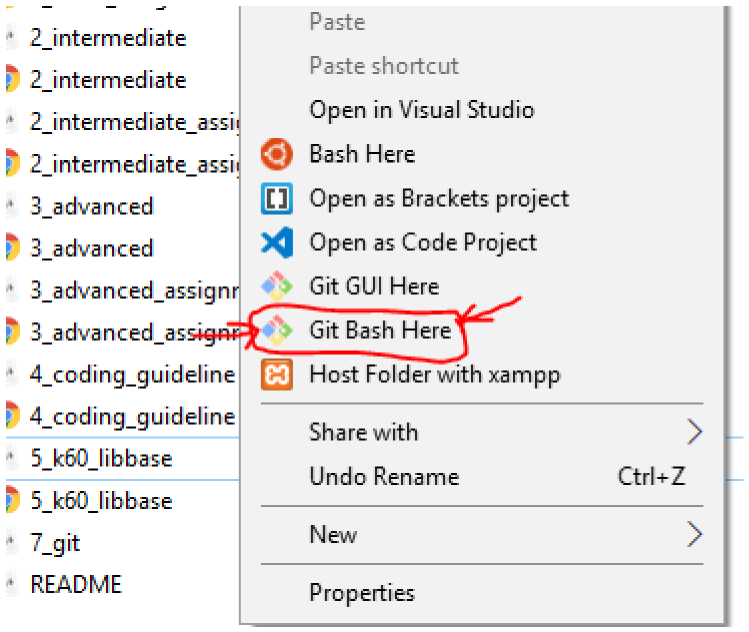
Enter you repo name and then click Create repository.

6. Change Working Directory

AKA let git know where you need git, and where to set up your repo

Option1:

Right click at your working directory, click git bash here



Option2:

Open git bash program, type

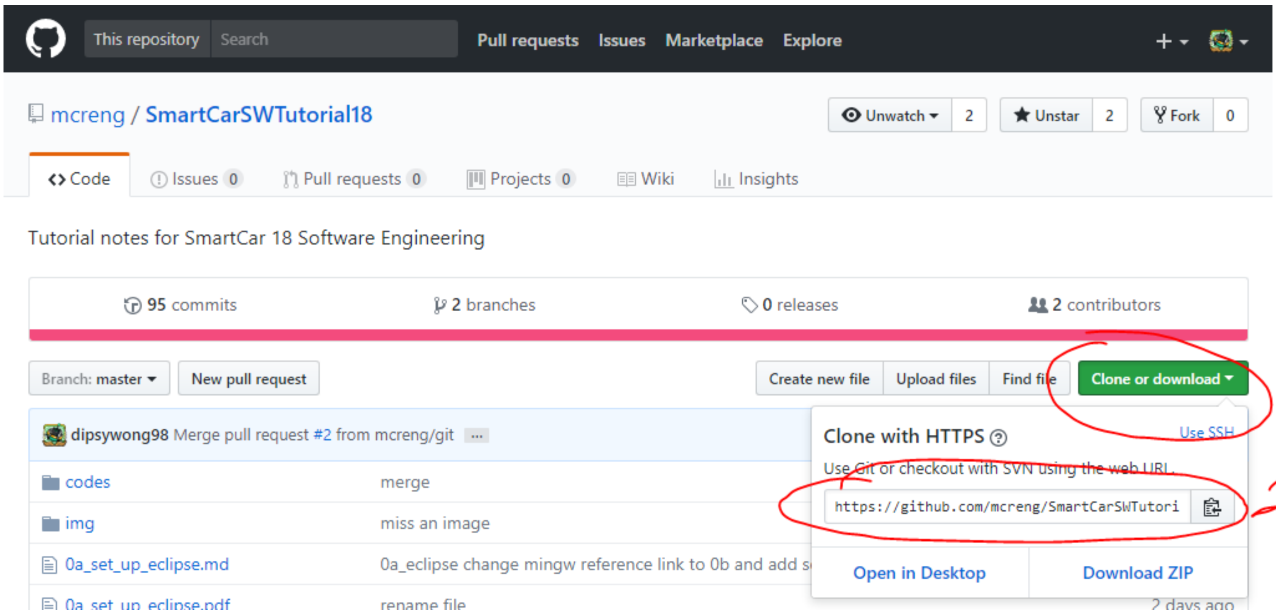
```
cd "you/dir/name"
```

PS: please use option1

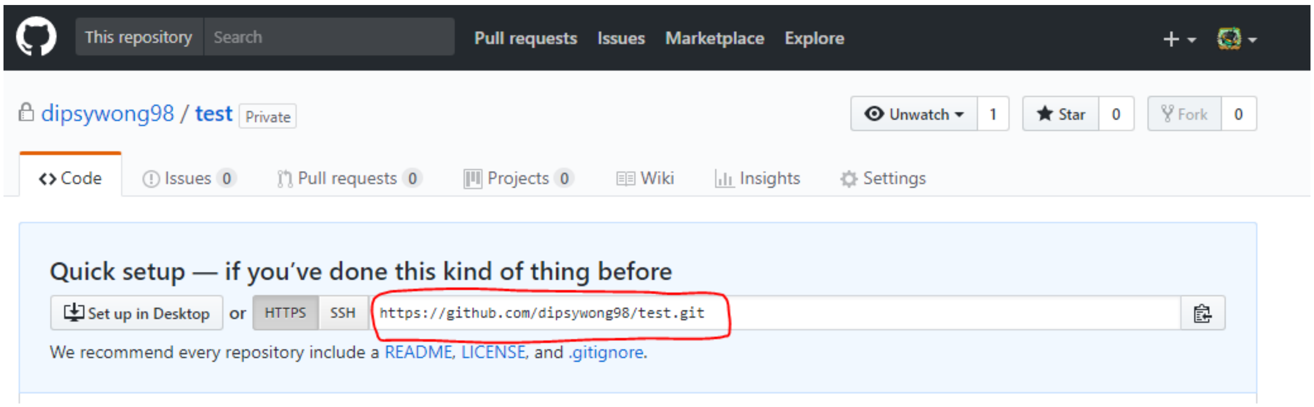
7. Get the Repository git URL

Git URL for GitHub is `https://github.com/{your username}/{your repo name}.git`. But for lazy guys, you can copy and paste:

For non-empty repo, you can get the git URL here



New repo


This repository
Search
Pull requests
Issues
Marketplace
Explore
dipsywong98 / test Private
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Quick setup — if you've done this kind of thing before
Set up in Desktop
or
HTTPS
SSH
https://github.com/dipsywong98/test.git
We recommend every repository include a README, LICENSE, and .gitignore.

8. Set up repo on your computer

Option1: Clone Repo (This will create new folder containing the repo)

Execute:

```
git clone https://github.com/{your username}/{your repo name}.git
```

Option2: Add remote (make current directory into a repo)

This is a bit advance, but more convenient than git clone if you know how to use

A remote is a git server as mentioned in "What is Git", and conventionally we name the main remote as `origin`. BTW, it is possible for one repository to have multiple origin.

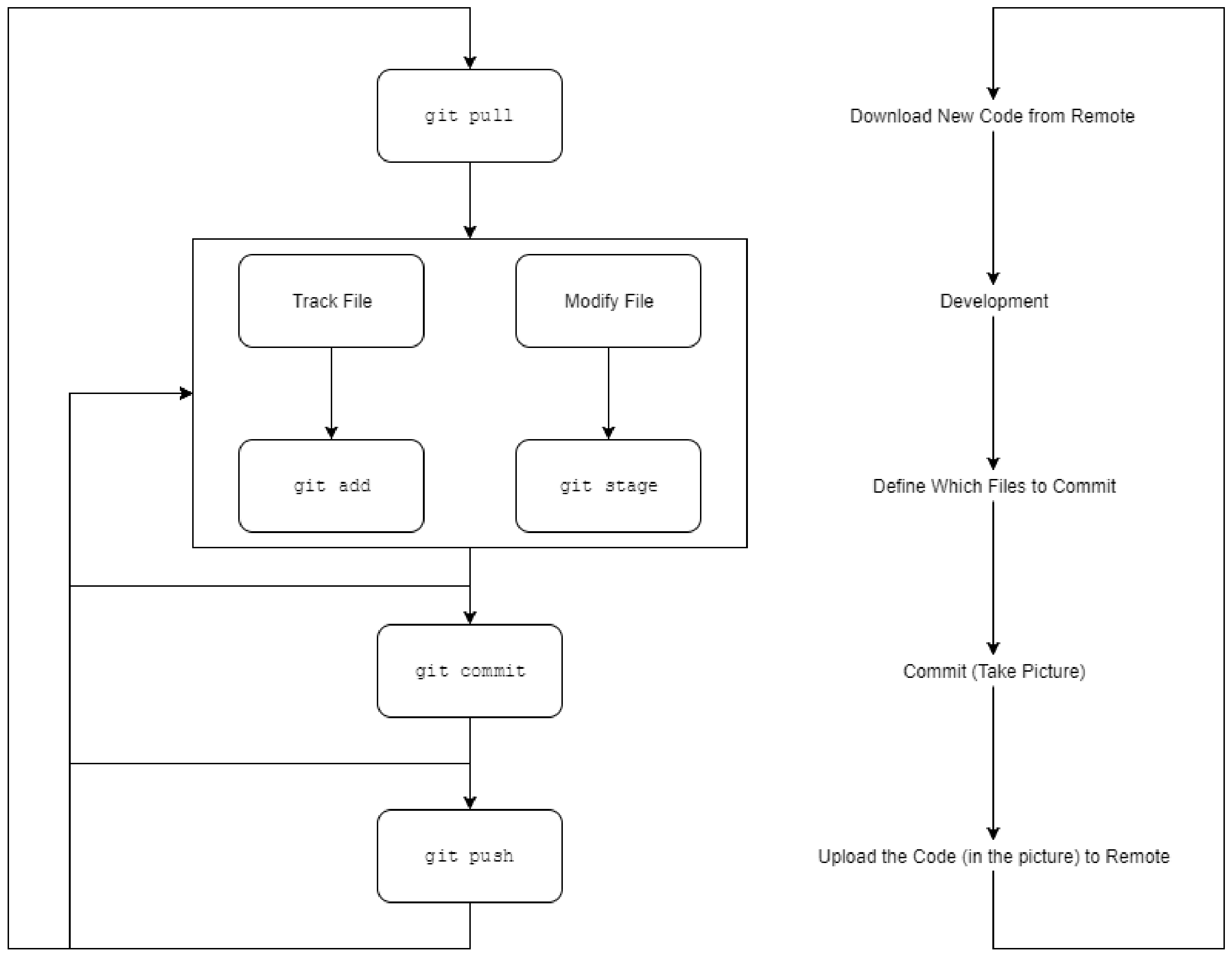
```
#initialize this directory as a git repo, AKA create a hidden /.git directory
git init

#add remote
git remote add origin https://github.com/{your username}/{your repo name}.git

#if the remote (GitHub repo) have existing code, pull it (download it)
#this line will be explain in next session
git pull origin master
```

This is useful when you have created an eclipse project, and you want to turn this project into a git repository.

9. Basic Flow of Git



git pull
Track File
Modify File
git add
git stage
git commit
git push
Download New Code from Remote
Development
Define Which Files to Commit
Commit (Take Picture)
Upload the Code (in the picture) to Remote

9.0 Overview

if you just want to upload all of your code, do this:

```
git pull    #download new version

#coding...

git stage *      #define this version is changes of all codes
git commit -a -m "commit message"    #name the version
git push         #upload new version
```

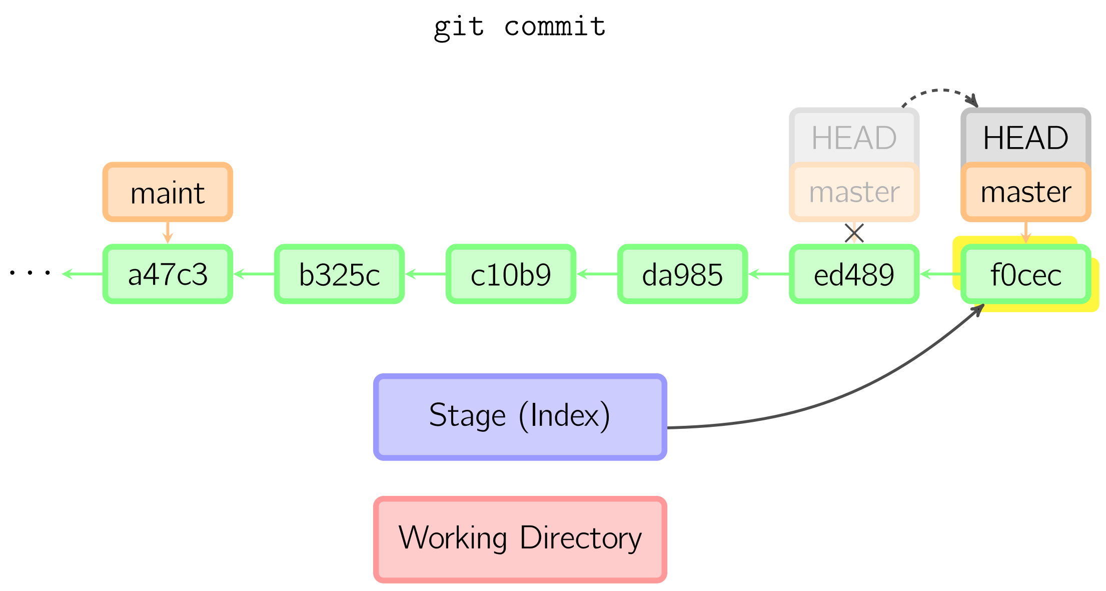
9.1 Some New Concepts

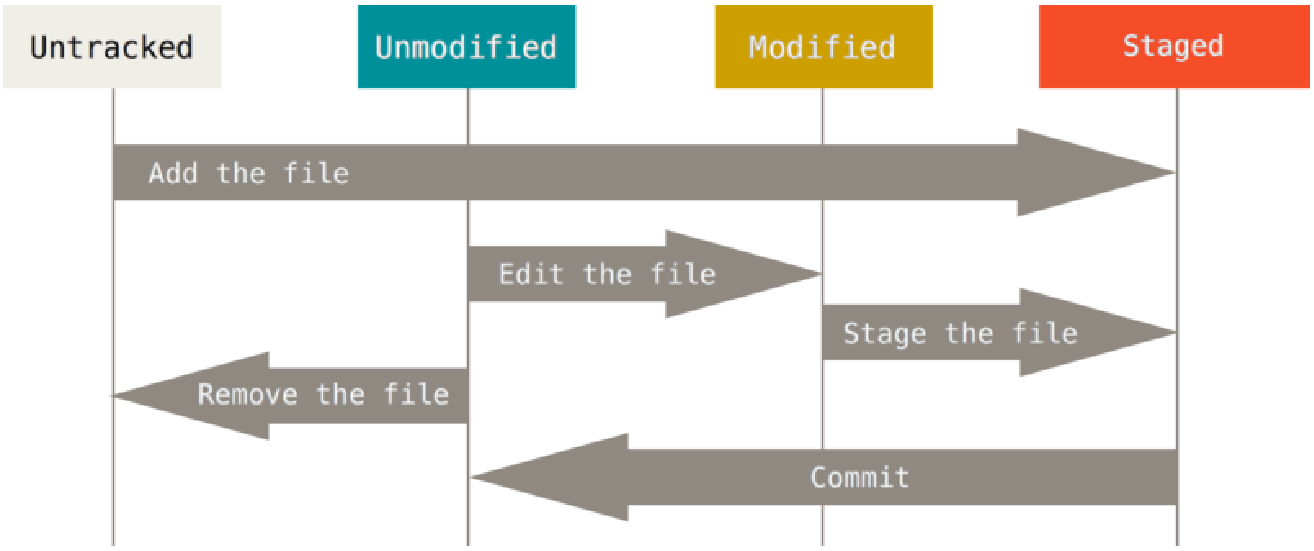
Before this, you may know that a repo have 2 versions (ignoring other contributors):

HEAD version and local version. (HEAD is a cursor to browse your version history. At this stage we are assuming HEAD version is your latest commit version)

Actually, there is an intermediate version by git, which is called stage.

During your development, you are working on the local version. After that you can stage the new files to the intermediate version. When git compare the intermediate version with HEAD version, git will know there is some changes. When you commit, your are making the intermediate version to be the new version of the repository, and HEAD version will be updated.





9.2 Git Status

```
git status
```

This command can tell you:

- Your current working branch (branch will be discussed later)
- Files in your directory but not in your local repo
- Files that have difference in local version and intermediate version (modified but not staged)
- Files that have difference in intermediate version and HEAD version (modified and staged)
- Number of commits you are ahead/behind of master

It is suggested to run this command before other git command to let you know what is your current status. Below can give your the idea on the information that `git status` provides.

9.3 Git Pull (Download New Code)

Before start coding new stuff, remember to git pull. If there is new stuff on remote but not on your computer, and you have implement new stuff that you want to upload to remote, conflict will happen (There are other sources of conflict and will be discussed later). So always make sure your local repository is in sync with the remote before doing new stuff. To git pull:

```
git pull
```

This will update the local, intermediate and head version as the newest version on the remote

9.4 Git Add (Track New Files)

Git is a system which tracks the changes of the files, so you need to tell git which files you would like to track. To track a file:

```
#option 1: one by one
git add readme.md

#option 2: more than one
git add my_class.h my_class.cpp

#option 3: wild card, * is corresponding to any sequence of characters
git add *.txt

#option 4: by directory
git add img/
```

The file just added will be directly staged. The files in the stage area are the files to commit.

9.5 Git Stage (Stage Modified Files)

When the tracked files have changes and you want to record them, you need to stage the the files. To stage a file.

```
#you can do it in a way similar to git add
git stage your_file.name
```

If you stage a file which is originally not tracked, it will become tracked and staged, which is same as `git add`.

It is update certain file in the intermediate version as the the local version

(intermediate version = local version)

9.6 Git Commit (Record this Stage)

A commit is the unit of git to track the versioning.

It is done by recording all the changes in the stage area. After commit, you repo is move on by one version, and the committed changes will be saved, and the corresponding files will be marked as unchanged as well. (create a new version for HEAD, HEAD version = intermediate version)

Each commit will come along with a commit message (to describe what is changed, please make it short and precise) and a SHA (a unique id for the commit)

```
#commit the staged files, and prompt a vim editor for you to edit commit message
git commit

#if you dont know how to use vim, use this to enter commit message at once
git commit -m "my commit message"
```

If you accidently enter vim (console line text editor), you can press `esc` and then enter `:wq!` , you can leave. If you want to learn vim, google it.

if you want to commit the changes of all tracking files, you can use `git commit -a -m ";your commit message";` to avoid typing `git stage`. But make sure you know which files are going to be committed (always `git status` first).

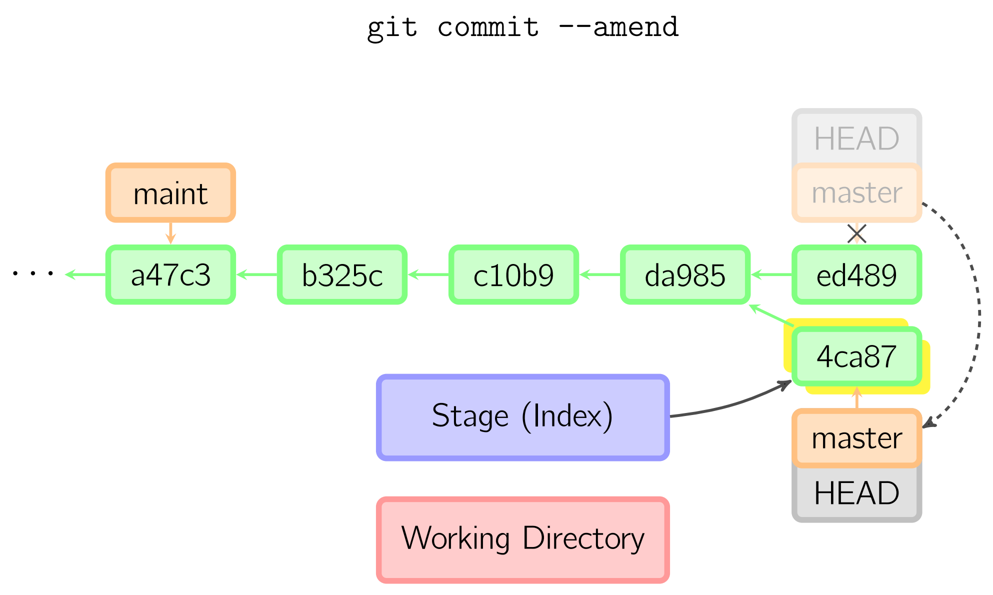
```
git commit -a -m "my commit message"

#similar with
git stage *
git commit -m "my commit message"
#except the first one will not add untracked files (important)
```

If your previous commit is a mistake, but you haven't push, you can still amend it by

```
git commit --amend -m "my commit message"
```

This will drop the last commit, and add a new commit using your stage version



9.7 Git Push (Upload New Commits)

After you commit, remember to push it, or else if other contributor push their commit, you will suffer from conflict when you want to push your code after some while.

```
git push
```

This will update the versions on the remote.

10. Fix Bugs by Undo

10.0 Overview

```
#if you stage something wrong and want to remove it from stage area
git reset the_stupid_file.txt

#if you want to roll back to the last commit after coding some stupid things
git reset --hard

#if you commit something wrong but thanks god you haven't push it
git reset HEAD^ #go back by 1 commit, hard reset
git reset a47c3 #go back to commit which have id (SHA) a47c3, hard reset

#if you commit something wrong but you already push it
git revert HEAD^ #create a new commit, which content is commit that before HEAD
git revert a47c3 #create a new commit, which content is commit having id (SHA) a47c3
```

10.1 Git Reset (Unstage the File)

`git reset` is a way to unstage the files from the staging area, by making the file content of the staging area same as previous commit. There is a soft reset and hard reset

- soft reset: the file content of your working directory will be unchanged
  (intermediate version = HEAD version)
- hard reset: all the file content of your working directory will be rolled back to the previous commit
  (local version = intermediate version = HEAD version)

if you stage or add a wrong file, you can remove it from the stage area by typing

```
#you can do it in a way similar to git add
git reset your_file.name

#or reset anything in the stage area by
git reset
```

If you want to reset anything in your working directory to HEAD version (last commit), do hard reset

```
git reset --hard
```

You can also use `git reset` to rollback multiple commits (if you haven't push those commits)

```
git reset HEAD^ #go back by 1 commit, hard reset
git reset a47c3 #go back to commit which have id (SHA) a47c3, hard reset
```

this is done by changing the version of HEAD to different commit version, and then copy the HEAD version to stage version and local version.

10.2 Git Revert (Undo Commits)

If you want to rollback commit that have been pushed, use `git revert` to safely undo it. Git revert is done by appending a new commit to current HEAD, make the version same as the reverted state. For example:

```
git revert a47c3
```

This will commit a new version same as `a47c3`.

You may also use `git revert HEAD^` to undo last commit.

Here we are not going to use `git reset` because `git reset` will remote the commit, if other contributor have already pull that commit, they will GG.

11. Branching

11.0 Overview

When there is diverge version/ new feature/ different people working, better use branch to avoid conflict/ as a restart point

```
#create new branch
git branch my-branch

#move HEAD to a branch
git checkout my-branch

#merge branch master with feature
#(master branch takes the new changes in feature branch)
git checkout master
git merge feature -m "commit message"

#rebase branch master on feature
#(make master brach base on feature branch)
git checkout master
git rebase feature
```

11.1 Why Branch

As a restart point

When you are making diverge version/ new feature, and then you find it totally sucks and you want to go back to a normal version and do not want to see your stupid code again, you can directly throw away this new feature branch. （砍掉重煉不太痛）

Avoid conflict

Conflict occurs when there is two new version of a same file. For example there is a file called `a.txt`, have `hello world` as its content. If Peter pushed a new version of `a.txt`: `hello genius Peter`, and I want to push new version of `a.txt`: `hello boygod Leslie`, conflict will occur as git don't know which is the real version. If branches is set up to separate everyone's work, everyone's new version can be tracked at the same time, and conflict can be solved only when there is merges of branch.

11.2 Create Branch and Switch Branch

To create branch which is base on current HEAD position, type

```
git branch branch-name
```

but note that you are still on your original branch, so you need to type

```
git checkout branch-name
```

to actually switch your working branch to that branch.

You may also create and switch in one shot

```
git checkout -b branch-name
```

Note that the version of your HEAD is still on same commit, and the stage version and local version unchanged as well.

When you want to push the new branch to your remote, for the first time you need to type

```
git push --set-upstream origin branch-name
```

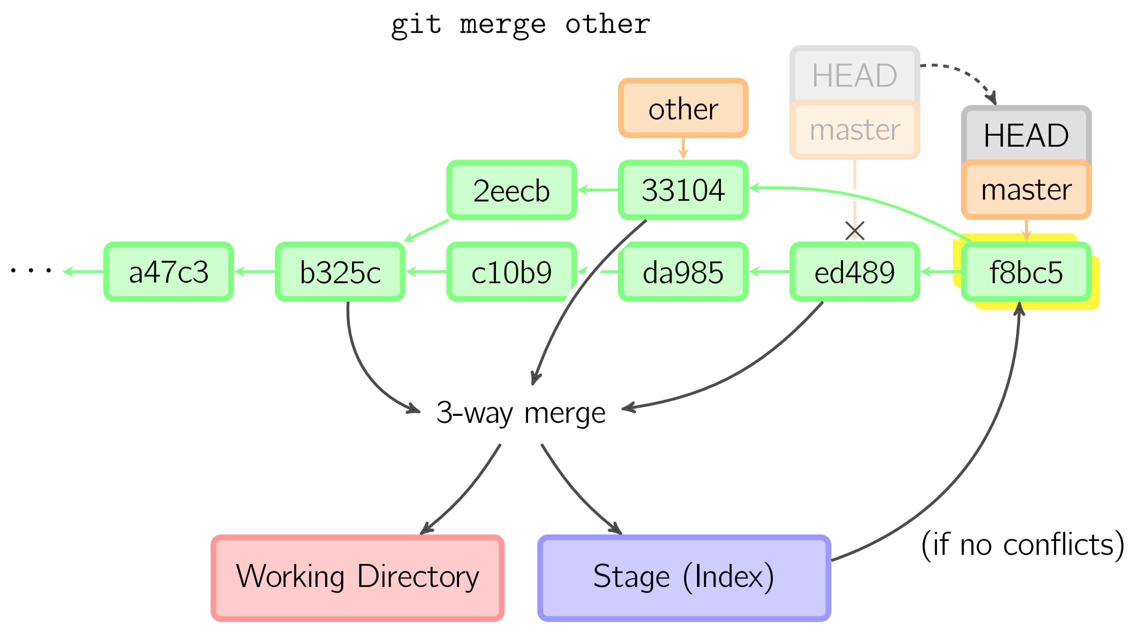
to create a new branch at your origin and push.

11.3 Merge Branch

When you done with some branch (let's call is feature branch) and want master branch to have these changes, you can first switch to master branch, and then merge the feature branch to master branch by referencing the newly created merge commit reference to both last commit at master branch and last commit at feature branch (Join two or more development histories together).

```
git checkout master
git merge feature -m "merge feature branch"
```

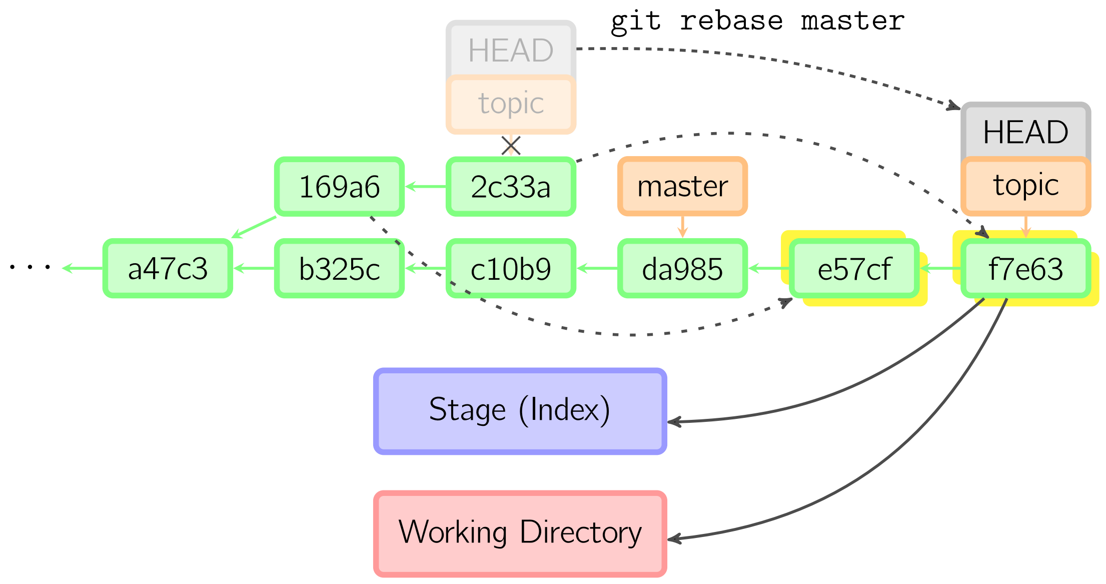
If the merge have possible conflict, it is better to solve the conflict at feature branch first, to make sure the new code is compatible with current code in master, so that code at master branch is always stable



11.4 Rebase

What you need to do with rebase is exactly same as merge, except its merging principle is different. Rebase is done by "sequentially regenerate a series of commits so they can be applied directly to the head node", which is copying the commits in feature branch, and paste them on master branch. And then the final commit history of master branch looks linear, while there is diamond shape in `git merge`.

```
git checkout master
git rebase feature
```



12 More on Git

12.1 Git Diff

This is to check difference in two commit.

(Actually use some IDE or GitHub website is a better choice)

```
#compare local and stage
git diff

#compare HEAD and local
git diff HEAD

#compare stage and HEAD
git diff --cached

#compare two commits
git diff b325c da985

#compare branch and local
git diff branch-name
```

12.2 Git Stash

Git stash is for temporary saving. When you are working on some version, and you need to switch to other branches, normally you need to make there is no local changes first (version of local and HEAD need to be sync). However you don't want to drop your changes nor making a new commit, you can use git stash to temporary save your changes.

```
#save your changes, and then reset your local changes
git stash
#then you can checkout other branches

#apply the changes stashed
git stash apply

#and then you need to manually drop the stash
git stash drop
```

you may have multiple stashes, you can list then out

```
git stash list
```

sample output

```
stash@{0}: WIP on master: ce172ad ggg
stash@{1}: WIP on master: fe32a47 master
stash@{2}: WIP on master: aa120be master
stash@{3}: WIP on master: 4e6da87 ggg
```

and apply/ drop specific stash by adding index as extra parameter

```
git stash apply 1 #apply stash@{1}: WIP on master: fe32a47 master
git stash drop 1 #delete stash@{1}: WIP on master: fe32a47 master
```

12.3 Git Checkout

As mentioned above `git checkout` can be used for switching working branch, it can also switch HEAD version to different commit. Note that detached HEAD will occur when you commit here.

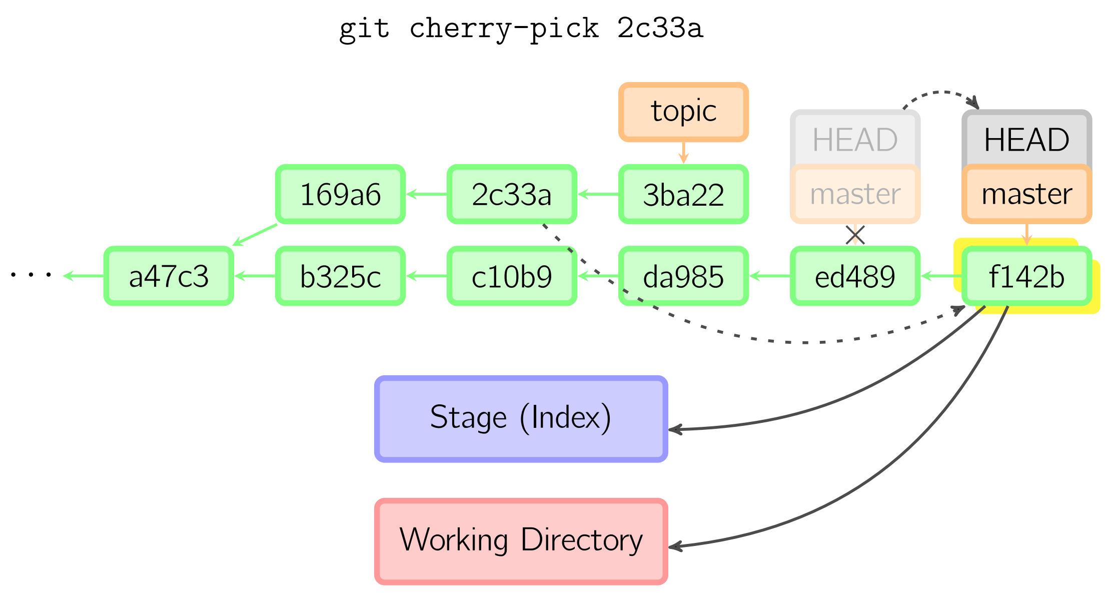
```
#set HEAD, stage, and local version as b325c
git checkout b325c
```

After coding on `b325c`, you may commit, this will lead to a detached HEAD, which mean a commit don't have branch label to track. If you checkout to other version, you may forever lost this version (unless you can remember its SHA). If you want to keep it, you can create branch for it.

12.4 Git Cherrypick

Cherrypick is to create a commit which is identical with another commit. This is useful if you want to copy a certain commit on a branch to another branch.

```
git cherry-pick 2c33a #create a new commit, which is same as b325c
```



Reference

http://marklodato.github.io/visual-git-guide/index-en.htm

https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository