Initialize a local github repository

```
SAIG05:git_learn$ git init
```

a git project includs three parts: a working directory, a staging area and a repository.

1. working directory: do all the actual work, creating, editing and organizing files.
2. staging area: list the changes make to the working directory.
3. repository: store all the changes as different versions of the git project.

Check the changes of the working directory

```
SAIG05:git_learn$ git status
```

It will show untracked files and file changes staged for commit

Add a file to the staging area

```
SAIG05:git_learn$ git add <filename>
```

the new file tells us the file was added to the stagging area.

```
SAIG05:git_learn$ git add .
```

this command will add all the files in the staging area

Track the changes of the staged files

```
SAIG05:git_learn$ git diff <filename>
```

Permanently stores changes from the staging area inside the repository

```
SAIG05:git_learn$ git commit -m "complete the first line of dialogu
re"
```

When we want to refer back to the earlier version of a project, we can use

```
SAIG05:git_learn$ git log
```

as all the commit are stored chronologically.

The difference between git and github

1. **eg**t is a version control system which enable us to look back at previous versions and undo errors. A project is managed by git is called git repository.
2. **eg**f s` is a hosting service for git repository, enable us to store local git repository in the cloud.

there are other servce to store git repository.

Connet the local git repository to the romote github repository

```
SAIG05:git_practice$ git remote add origin https://github.com/Wnele
iGao/git_practice.git
SAIG05:git_practice$ git push -u origin master
```

Setting Username in Git (Set the user name for all the git repository)

Git Username is not the Username of github

```
SAIG05:git_practice$ git config --global user.name "WneleiGao"
```

Confirm my user name

```
SAIG05:git_practice$ git config --global user.name
```

Setting the email of git user

```
    SAIG05: git_practice$ git config --global user.email "wgao1@ual
berta.ca"
```

we can check the user email address by typing the comand in the terminal

```
    SAIG05: git_practice$ git config --global user.email
```

Use credential-osxkeychain

```
    SAIG05: git_practice$ git credential-osxkeychain
```

if it shows

```
usage: git credential-osxkeychain <get|store|erase>
```

means the credential tool is installed on the local machine.

Tell git to use osxkeychain helper

```
SAIG05:git_practice$ git config --global credential.helper osxk
eychain
```

The usage is that, only the first time of using git clone require to type username and password, after that, the git clone does not need them anymore, because they are added to the keychain helper.

Backtrack the HEAD commit

Definition of HEAD commit: the most recent commit, to see the head commit, enter

```
SAIG05: git_practice$ git show HEAD
```

recover the file in working directory to the most recent commit(discard changes in the working directory)

```
SAIG05: git_practice$ git checkout HEAD <filename>
```

unstage the file from the staging area using

```
SAIG05: git_practice$ git reset HEAD <filename>
```

if we want to reset back to any one of the previous commit, we need to use (set HEAD point to a previous commit)

```
SAIG05: git_practice$ git reset <commit_SHA>
```

should be replaced by the first 7 characters of the SHA of a previous commit, just change the commit history, not the content of the files, to change the content of files in working directory, we need to run

```
SAIG05: git_practice$ git checkout HEAD <filename>
```

git branch

Find out which branch i am working on use the command

```
SAIG05: git_practice$ git branch
```

a branch is a different version of the git repository, create a new via the command

```
SAIG05: git_practice$ git branch new_branch
```

the new branch name is "new_branch", can't contain space, usually have some meaning of this branch. At this point, the content of "new_branch" is identical to the master branch.

change the branch by using

```
SAIG05: git_practice$ git checkout new_branch
```

Once I switch the branch, I am able to commit to the new branch and keep the master branch intouch. the old commits are inherited from the master branch.

include the changes made on one branch to the master branch by the commands

first we need to switch the branch to master

```
    SAIG05: git_practice$ git checkout master
```
then do the fast-forward merge
```console
    SAIG05: git_practice$ git merge new_branch
```

Merge conflict

if the changes happened on the same line of new_branch and master branch, we will run into merge issues

```
SAIG05: git_practice$ git merge new_branch
```

we need to manually fix the conflicts and commit again.

delete branch

The new_branch is created for developing new features for a git project. After testing the new feature, we should merge it to the master branch. After merging, the new_branch can be deleted.

```
SAIG05: git_practice$ git branch -d new_branch
```

Collaborate with others on a git project requires

1. A complete replica of the project on my local machine
2. a way to keep track and review others' work
3. access to the definitive project version.

pck mrc: a shared git repository

Get own replica

```
SAIG05: $ git clone remote_location clone_name
```

remote_location tells git where to find the remote(web address or a filepath) clone_name: the name of the directory you want to put the content.

at this stage I make any changes to clone_name, remote_location will not see it. At the same, git give the remote location the name mpeg .

we can check it via

```
SAIG05: $ git remote -v
```

the remote address listed twice, one is for fetch and the other one is for push.

Check the changes made on remote and bring a local copy of these changes

```
SAIG05: $ git fetch
```

these changes will not merge to my local git repo, it will create a new branch called origin/master, put the changes in the new created branch.

bring the local git repo speed up to the remote repo, we still need to run

```
SAIG05: $ git merge origin/master
```

Typical workflow for collaberation

1. Fetch and merge changes from the remote
2. Create a branch to work on a new project feature
3. Develop the feature on your branch and commit your work
4. Fetch and merge from the remote again (in case new commits were made while you were working)
5. Push your branch up to the remote for review

Steps 1 and 4 are a safeguard against merge conflicts, very important to eliminate merge conflicts.

```
SAIG05: $ git push origin my-branch
```

import note

1. A remote is a Git repository that lives outside your Git project folder. Remotes can live on the web, on a shared network or even in a separate folder on your local computer.
2. The Git Collaborative Workflow are steps that enable smooth project development when multiple collaborators are working on the same Git project.