Chapter 1 (Git setting)

git configure is the working environment

1. set the name and email `$ git config --global user.name "yourname" $ git config --global user.email your email`
2. set the default branch name `$ git config --global init.defaultBranch name`

what is snapshot?

to save a snapshot of the project when you commit is to save the whole project content, including all the files, at that point; git will also compare each file with the former version to see if it can save some storage. Comparing to other VCS, git will not save the changes between each version of files.

Chapter 2 (Git basic)

get a repository(repo)

1. init a repo in an exiting dir

- used for not under control
- `$ cd C:/Users/dell/project_name` and `git init`(this will create a new subdir named .git containing all repo files)
- to control exiting file

```
$ git add *.c
$ git add LICENSE
$ git commit -m 'Initial project vers'
```

2. clone an existing repo

```
$ git clone [url] dir_name(optional)
```

- the dir_name is the name you want the file to be
- open your vpn to all

recording change to the repo

files of 4 kinds: untracked,{ unmodified, modified, staged}(tracked)

- modified files need to be staged(use add) to be the stage file
- untracked files can be added to be the staged file,
- staged files need to be committed to be an unmodified file
- a tracked file being modified can turn into an modified file

1. to know the stage of the dir and all the files

```
$ git status
```

```
$ git status -s
```

simplified version 2. short cut

- aren't tracked '??'
- new file added to staging 'A'
- modified file 'M'
- staged and then modified again 'MM'

3. .gitignore place where you tell git to ignore the kinds of files you do not want to track, see the example in .gitignore or the net
4. check the difference of the content

```
$ git diff
```

check the difference between modified and staged

```
$ git diff -cached
```

check the difference between staged and unmodified 5. skip the stage step

```
$ git commit -a
```

this will automatically stage all the modified files 6. removing files

```
$ rm file
```

this will remove from the working dir but not staged

```
$ git rm file
```

this will remove both from the working dir and the staged place(that is the removal will be staged to commit)

```
$ git rm --cached file
```

this will only remove the file from the staged place, not the working place -> remember to add \ before the file extension, such as

```
$ git rm \*.log
```

7. move the file

```
$ git mv file_1 file_2
```

this will rename the file, meaning that it will move the first file to the second file, remove the first file and add the second file

view the log

```
$ git log
```

optios:

1. -2: show two commit
2. --stat: show the number of lines and files edited
3. --patch: show the diff
4. --pretty=value:
5. --oneline: print in a line
6. --author:choose the author
7. --grep: search for key words
8. --since/-until: choose the time
9. -S: take a string and show the commit that change the string
10. -- path/to/file: the commit that change the specified files
11. --no-merges: ignore the merge commit

undoing thing

```
$ git commit --amend
```

this will rewrite the previous commit

```
$ git reset HEAD <file>
```

---

this will unstage the staged file

```
$ git checkout -- <file>
```

this will replace the modified file with the previous staged one

```
$ git restore --staged <file>
```

the same with reset

```
$ git restore <file>
```

the same with checkout

working with remotes

```
$ git remote -v
```

show the remote handle

```
$ git remote add <shortname> <url>
```

this will explicitly add a remote

```
$ git fetch <remote short name>
```

get the data from remote project

Tag

tag works like a pointer to the commit version you have made, use a tag to specify a version

1. add an annotates tag(with detailed info) `$ git tag -a v1.0 -m "version 1.0"`
2. add a lightweight tag `$ git tag v1.0-lw`
3. add tag to previous commit `$ git tag -a v0.9 <checksum> -m "version 0.9"`
4. show a version `$ git show v1.0/v1.-lt`
5. push your tag `$ git push <remote> --tag` push all the tag `$ git push <remote> v1.0-lt` push a specific version
6. go to the previous version `$ git checkout v1.0`

7. make a new branch from the previous version `$ git checkout -b <new_branch_name> v1.0` this will switch to a new branch

alias

make your shortcut of the git subcommand `$ git config --global alias.shortcut command_name`

This is where we encounter the conflict.

Chapter 3 (Git branching)

We will put this part in the git branching file.