```
################################################################################
#
# 2018/12
# reference:
https://www.youtube.com/playlist?list=PLg7s6cbtAD15G8lNyoaYDuKZSKyJrgwB-
################################################################################
#

# Git Bash
$ ssh-keygen -t rsa -b 1024 -N "" -C <my_email_address>


$ git init <my_proj>
$ open .git    # MAC


$ git remote add origin <https://my_URL>
$ cat .git/config


$ git config --global user.name <my_name>
$ git config --global user.email <my_email_address>
$ git config --global core.autocrlf true    # line endings
$ git config --global color.ui auto
$ git config --local user.name <my_name>
$ git config --list


$ git add <my_file.txt>    # move file to "staging" area
$ git commit -m "my_comments"    # determine which files in staging area become a
part of which commit
$ git commit --amend    # re-write previous commit descriptions


# push local existing code to remote
$ cd <my_existing_project>
$ git init
$ git add --all
$ git commit -m "my_initial_commit"
$ git remote add origin <https://my_URL>
$ git push -u origin master

# push local tracked code to remote
$ cd <my_existing_project>
$ git remote set-url origin <https://my_URL>
$ git push -u origin master


# 1. working tree, 2. staging area, 3. head (most recent commit)
$ git diff    # changes made have not yet staged (difference between working tree
```

```
and staging area)
$ git diff --staged    # changes between staged files and most recent commit in
history
$ git diff HEAD    # compare working tree to head commit (most recent commit in
history)

$ git diff --color-words
$ git diff --word-diff
$ git diff --stat

$ git diff --name-status <my_branch_1>...<my_branch_2>


$ git log
$ git log --oneline
$ git log --stat
$ git log --stat -- <my_folder/my_file.txt>    # log stops after files are moved
$ git log --stat -M --follow -- <my_folder/my_file.txt>    # log follows file across
moves
$ git log --patch    # show difference between commits
$ git log --graph --all --decorate --oneline
$ git log --oneline --decorate --graph --all -10    # show branch info
$ gitk --follow <my_file.csv>    # track history of changes
$ gitk <my_file.txt>    # check log of a file
$ git show <my_commit_reference_id>    # changes from a commit


$ git rm <my_file.txt>    # delete file from file system
$ git rm --cached <my_file.txt>    # remove files from "staging" area without
deleting it in local (make it untracked file)
$ git add -u .    # stage all deleted files (. tells git to start at current
directory and recurse all the way down as far as it can go)
$ git rest    # "undo" git add . and remove file from "staging" area


$ git update-index --chmod=+x <my_script.py>    # change file permission on Windows


$ git mv <my_old_destination/my_file.txt> <my_new_destination/my_file.txt>
$ git add -A .    # handle situation where lots of files are moved


$ touch .gitignore    # create .gitignore at root of project
$ git ls-files --others --ignored --exclude-standard    # show ignored files

#----- .gitignore (example) -----
myfile.txt
*.log
# comments inside .gitignore
tmp_folder/    # ignore files in folder
```

```
# Created by https://www.gitignore.io/api/python,pycharm
#----- .gitignore -----


$ git branch    # check which branch I'm on
$ git branch -r    # r for remote
$ git branch <my_branch>    # create a new branch
$ git branch -d <my_branch>    # delete branch and info won't be shown in log
$ git branch -a    # list both remote-tracking and local branches


# rename a branch (at the branch which will be renamed)
$ git branch -m <my_new_branch_name>
$ git push origin :<my_old_branch_name> <my_new_branch_name>    # no need?
$ git push origin -u <my_new_branch_name>


# checkout is for changing branches
$ git checkout <my_branch>    # changing branches
$ git checkout <my_commit_reference>    # detached HEAD; go back to previous version
for exploring purpose
# latest node is called "head"
$ git checkout <my_commit_reference> -- <my_file.txt>    # revert a file to previous
version
$ git checkout master    # back to the latest version - master branch
$ git checkout -- <my_file.txt>    # undo/discard changes made to file
$ git stash
$ git stash pop
$ git checkout -b <my_branch>    # create a new branch and checkout at the same time
$ git checkout -b <my_branch> origin/my_remote_branch    # "download" a branch which
was created remotely


# checkout a file "my_file.py" from another branch - "my_branch_2"
$ git checkout my_branch_1
$ git checkout my_branch_2 -- my_file.py

# update "my_feature_branch" with the latest "my_development_branch"; sync
repositories
$ git checkout <my_development>
$ git pull origin <my_development>
$ git checkout <my_feature>
$ git merge <my_development>


# if there is a conflict while merging (@ master & merge feature branch to master)
<<<<<<< HEAD
"content in master" or "branch current checkout to"
=======
"content in incoming branch"
```

```
>>>>>>> "incoming branch; attempt to merge"

$ git merge --abort    # abort merge while having conflicts
$ git merge --squash "target branch"    # commit but not history


# remote tracking branches (origin/*) are the middle way between branches at local
laptop and at GitHub.com
$ git remote add origin <my_destination_URL>    # setup location where we send
information to from laptop; origin is the name of the destination
$ git remote set-url origin <my_destination_URL>    # change URL
$ git remote rm origin    # remove remote
$ git remote -v
$ git remote prune origin --dry-run    # remove local branches which are removed in
remote server
$ git remote prune origin


# fetch, pull, push operations
$ git fetch origin    # grab Github.com files to remote tracking branches
$ git pull origin    # pull = fetch + merge
$ git pull origin <my_branch>    # pull latest files from remote and ensure local
"my_branch" is up to date (check???)
$ git pull <my_development_branch> <my_feature_branch>    # merge development branch
to feature branch; keep feature branch up to date (check???)
$ git push origin    # only for all matching branches (local/remote)
$ git push origin <my_feature_branch>    # only for my_feature_branch (local/remote)
 (check???)
$ git push -u origin master    # difference?
$ git push <my_remote_origin> <my_branch_master>    # if remote doesn't have master,
then you need to specify  (check???)
$ git push origin --delete <my_remote_branch>    # delete remote branch
$ git push --set-upstream origin <my_development_1>    # setup upstream for a branch
$ git branch -vv    # find out which remote branch a local branch is tracking


# git reset is for shaping history in repo
# git reset has - mixed(default), soft(combine commits), hard(erase things)
$ git reset HEAD    # take changes out of staging area (mixed; default option)
$ git reset --soft HEAD~2    # the most recent 2 commits
$ git reset --soft HEAD~1    # use --soft if you want to keep your changes
$ git reset --hard HEAD~3    # throw the most recent 3 commits away
$ git reset --hard HEAD~2    # use --hard if you don't care about keeping the
changes you made
$ git checkout <my_commit_reference> <my_old_file.txt>    # pull a specific version
of file back to current repo

# revert: create a new commit (commit + 1 from earlier commit)

$ git revert <my_commit_reference>    # revert changes made in a commit
```

```
$ git reflog    # track commit made and discarded
$ git reset --hard <my_commit_reference>    # restore code base


# if content of a large git-lfs file cannot be seen
$ git lfs install
$ git pull


#----- change default home directory while opening git bash -----
C:\Program Files\Git\etc\bash.bashrc
# added the following two lines into bash.bashrc
if [[ $(pwd) = $HOME ]]; then cd Desktop/Shih/projects
fi
#----- change default home directory while opening git bash ------


$ git clone <http://login@url.git>    # alternative: ssh

#----- Use pcron to sync GitHub repo/Unix automatically -----
git clone <https://url.git> .

git config credential.helper 'store'

git pull <https://url.git>

pcrontab -e
* * * * * git --git-dir=<unix_path.git> --work-tree=<unix_path> pull
<https://url.git>

pcrontab -l

nohup pcron &
```