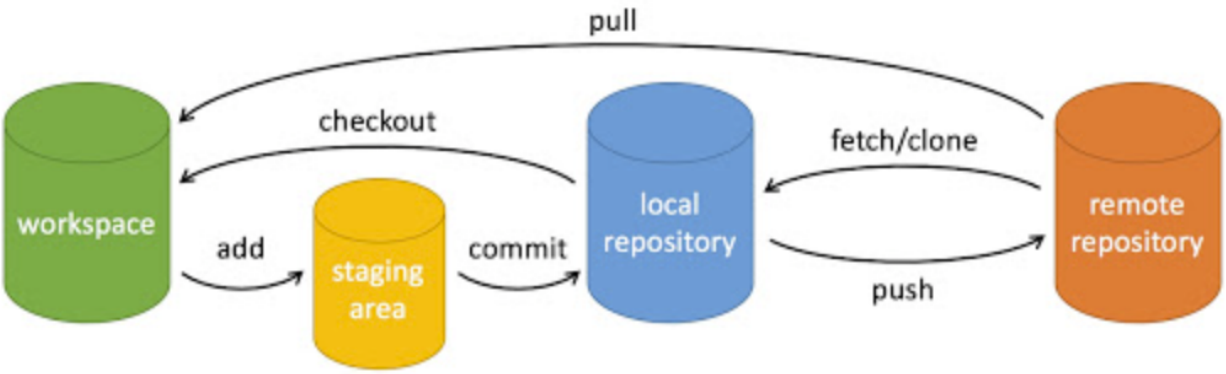
How to use Git(After Installing Git)?

Reference

> chatgpt (openai.com)
>
> Git 基本操作 | 菜鸟教程 (runoob.com)
>
> git reset 使用及回滚*git reset 目录*AI小兵的博客-CSDN博客

Git working process diagram



workspace: where you work

staging area: where the files store temporarily

local repository: the repository in your computer

remote repository: the repository in the server

Pay attention to the versions(denotes V):

$$V(workspace) \geq V(staging\ area) \geq V(local\ repo) \geq V(remote\ repo)$$

Get a repository

1. create a new repository(create a `.git` in the specified place)

   Method 1:

   - **open Command line or PowerShell to `cd` into the right place** or **(Get into the directory you want to create a repository, then right click => `use powershell to open` or right click => `more options` => `Git Bash Here`)**
   - ```
     # input the command
     git init
     ```

   Method 2:

   - **open Command line or PowerShell or Git Bash**
     ```

- ```
  # input the command
  git init <directory>
  # <directory> is the the directory you want to create a new repository
  ```

2. When you have a repository, clone from it.

   Method 1:

   - When you are in the **directory** you want, and you know the **url** of a repository
   - **open Command line or PowerShell or Git Bash**
   - ```
     git clone <url>
     ```

   Method 2:

   - When you know the **url** of a repository
   - **open Command line or PowerShell or Git Bash**
   - ```
     git clone <url> <directory>
     ```

3. check whether you successfully create a `.git` in a repository

   - **open Command line**
   - ```
     dir /a .git
     ```
   - **or open PowerShell**
   - ```
     get-childitem -force
     ```

Between workspace and staging area

1. adding something

   In **command line or powershell or git bash**

   ```
   # add files
   git add <file1> <file2>
   # use wildcards, take all the pdf as an example
   git add *.pdf
   # add directories
   git add <directory1> <directory1>
   ```

   ```
   # add all the files in the current directory
   git add .

   # problem: .git is also added, how to avoid?
   # solution: create a .gitignore, specify the file you want to "miss" the
   add operation
   # useful grammar in .gitignore
   # remark that each command take one line in .gitignore
   ```
   ```

```
# ignore 1.txt
1.txt
# ignore all .a file in the current directory
*.a
# treat foo.a as an exception
!foo.a
# ignore a directory called build
build/
# ignore all the .a file recursively
**/*.log

# to create a file in the current directory in powershell
new-item -itemtype file -name ".gitignore"
# rewrite the content
set-content -path ".gitignore" -value "some contents"
# add contents(remark that `n means newline)
add-content -path ".gitignore" -value "*.txt`n!2.txt"
# or you right click in the directory and create a new .txt, after
modifying it, update the name to ".gitignore"
```

```
# commonly used .gitignore
.gitnore
.git/
```

After adding, we want to see what we have added to the staging area

```
git status -s
```

we can also see the **difference** bewteen **staging area** and the **working space**

```
git diff
```

2. **Give up** all the modification in the workplace **after your last** `add`, reset it to the current stage as staging area.

```
# It's the opposite of the add command
# give up all the modification in a file
git checkout <filename1>
# give up all the modification in a directory
git checkout <dir1>
# give up all the modification in every .txt file
git checkout *.txt
# give up all the modification in everything
git checkout .
```

DON'T DO THIS UNLESS YOU ARE SURE WHAT YOU'RE DOING ACTUALLY

```
git rm <file_name> -f
# delete the file both in the workplace and staging area, that is, your local file will be deleted.
```

Between staging area and local repository

1. `commit` the changes from the staging area to local repository

```
# check the difference between staging area and repository
git diff --cached
# commit all files
git commit -m "<commit_message>"
# commit specified files or directory
git commit -m "<commit_message>" <file1> <file2>
git commit -m "<commit_message>" <dir1> <dir2>
```

2. `reset` the versions

```
# basic syntax 1:
git reset [--soft|--mixed|--hard] <commit_id> <stuff>
# stuff: can be files(file1.pdf), directories(dir1), files with wildcards(*.txt) or everything(.)
# commit_id: indicate the version you want to reset
# if you want to check all the commits(including ids, messages...) in the reverse order of time
git log
# --soft: reset the <stuff> in the local repo to the <commit_id> version, keeping the workspace and staging area unchanged
# --mixed: reset the <stuff> in the local repo to the <commit_id> version and so does the staging area, keeping the workspace unchanged
# --hard: reset the <stuff> in the local repo to the <commit_id> version and so does the staging area and workspace
# BE CAREFUL TO USE --hard

# basic syntax 2:
git reset [--soft|--mixed|--hard] <head_para> <stuff>
# <head_para> denote the version you want to reset
# the first of version of the countdown(倒数第一个版本)
git reset [head~1|head^]
# then the second, the third... can infered:
git reset [head~2|head^^]
git reset [head~3|head^^^]
...
```

Branch Management

Purpose: When we are updating a project, we may udating parts in parallel.In this way, we can update the commits in the different branched in the local repository and so does the remote repository

1. list all the existing branch

```
git branch
# the branch with a * is the current branch
# the default branch create by git is master
```

2. create a new branch

```
git branch <branch_name>
```

3. rename the current branch

```
git branch -M <new_name>
```

4. switch to another branch

```
git checkout <branch_name>
```

5. merge commits from another branch to the current branch

```
git merge <branch_name>
# Merge conflicts usually occur when two branches modify the same part of the same file(different part of same file will be merged successfully, and those changes cannot be merged automatically.

# solution
# 1. Use the git status command to view merge conflict files and which parts have conflicts.
# 2. edit the file and make sure you cover all the conflict parts
# 3. use git add and git commit to commit the current file to the current branch(which you want to be the final branch)
```

6. delete a branch forever(after you merge the commits, a branch will be useless)

```
git branch -d <branch_name>
```

Between local repository and remote respository

1. Push somthing to remote repo

   Link a existing remote repository

```
git remote add <remote_repo> <url>
# <remote_repo> is the name of your remote repository
# <url> the url of your remote repository
# For the first time, you may need to identify your access right
```

push it to the remote repository

```
git push <repo_name> <branch_name>
# <remote_repo> is the name of your remote repository
# <branch_name> is the a branch name in your remote and local repository, if the branch doesn't exist, it will be created.
# if branch names in remote and local repository differs
git push <repo_name> <local_branch_name>:<remote_branch_name>

# if the log in the local repository contradict the log in remote repository, it may failed. You can cover the remote log with the local log:
git push -- force <repo_name> <branch_name>
```

2. delete a remote branch

```
git push <repo_name> --delete <branch_name>
```

3. download from remote repo to a new workspace
    - init a new local repo
    - linked the remote repository
    - fetch remote repository(from remote to stagging area)

      ```
      git fetch <remote_repository_name>
      ```

    - switch to the local branch you want(indispensable steps)

      ```
      git checkout <local_branch_name>
      ```

    - merge the changes to workspace

      ```
      git merge <remote_repository_name>/<remote_branch_name>
      ```