Git

> learning: https://learngitbranching.js.org/?locale=zh_CN

Basic concept

Three areas

- working directory 工作区

  实实在在的目录与文件，直接操作的地方就是工作区

  - 对应操作：edit
- staging area (alias. stage or index) 暂存区

  修改后需要**add**才能进入暂存区，暂存区内的东西才能**commit**

  - 对应操作：add

  > 这里的add不是用来track文件的

- git directory (repository) or commit history 版本库

  被记录的版本

  - 对应操作：commit

`git status`中工作区的文件显示为红色，暂存区的为绿色

> 奇怪的是我修改后可以直接commit，另外IDEA图形界面的commit是附带add的

reset

command	areas
`git reset` & `git reset --mixed`	重置暂存区的文件与上一次的提交(commit)保持一致，工作区文件内容保持不变
`git reset --hard`	将暂存区与工作区都回到上一次版本，并删除之前的所有信息提交
`git reset --soft`	保留工作区与缓存区，但是把版本之间的差异存放在缓存区

diff

command	areas
`git diff`	对比工作区和暂存区
`git diff HEAD`	对比工作区和版本库
`git diff --cached`	对比暂存区和版本库

Unstaged & Untracked

Files unstaged and untracked need to be deleted or commited before merging.

Rollback

```
git checkout -- <file> # rollback modification in work directory
```

Stash

> senario:
>
> 当我们开发一个新功能时会先从master拉出一个分支dev，然后在这个dev分支下吭哧吭哧的开始写代码开发新功能。就在此时，线上版本master出现了bug，我们应该放下手头上新功能的开发工作先将master上的bug修复，这个时候dev分支下的改动怎么处理？
>
> 首先我们新功能的代码还没开发完成，其次新功能这里还有一些bug没解决，就这样把有问题的代码提交到dev分支中，虽然可以解决目

> 前我们的处境但不是很妥；但是第二种方案，直接切换，明显更不妥。怎么办？我们好像陷入了困境……
>
> Git提供了一个**git stash命令**恰好可以完美解决该问题, 其将当前未提交的修改(即，工作区的修改和暂存区的修改)先暂时储藏起来，这样工作区干净了后，就可以切换切换到master分支下拉一个fix分支。在完成线上bug的修复工作后，重新切换到dev分支下通过**git stash pop**命令将之前储藏的修改取出来，继续进行新功能的开发工作

stash是stack

```
# stash所有工作区和暂存区的修改
git stash
git stash save # the same as `git stash`
git stash save "comments" # add message
# stash后，被stash的修改会被删除
```

```
# check stash list
git stash list
# like: stash@{index}: WIP on [分支名]: [最近一次的commitID] [最近一次的提交信息]
```

```
# 取出和丢弃
git stash pop # 取出最近一次stash的文件到工作区
git stash apply stash@{index} # 取出指定index的文件
git stash drop stash@{index} # 丢弃index的stash
```

Branch

```
git branch -m [old name] [new name] // rename
```

Merge

conflict

`scene`: After pulling, there are conflicts so merge is suspended

`solution`: modify the conflict files and `all` files tracked need to commit. If want to abort merge, use `git merge --abort`

```
# suppose it is on `feature` branch
git merge master # feature merge into master, it is still in feature branch
```

Rebase

> 为原始分支中的每个提交创建全新的提交来重写项目历史记录，原始分支的提交历史改变，且无法知道什么时候合并的

> https://zhuanlan.zhihu.com/p/57872388

```
# suppose it is on `feature` branch
git rebase main # feature will rebase on main
git rebase main feauture # the same as above
```

Local and Remote

Credential

> Https

`git config --global credential.helper store`: create a cache to store your account and password, after next time you type account and password, you don't have to type them again.

User

```
#local 只对某个具体的仓库生效
git config --local
```

```
#global 对当前用户所有的仓库生效
git config --global

#system 对系统所有登录的用户生效
git config --system
```

```
git config --local user.name 'name'
git config --local user.email 'email'
```

⭐ local user's email should be the same as remote user's email, otherwise a new remote user will be created.

Remote

`git remote`: check remote name

`git remote -v`: check remote name and url

`git remote add [name] [url]`: add a remote

`git remote remove [name]`

Pull & Push

`pull`

`git pull [remote host] [remote branch]:[local branch]`

if remote branch and local branch are the same name and remote host is default, `[remote branch]:[local branch]` can be omitted.

Reset

`git reset [target ID] --hard`: reset to target commit.