Git使用场景

场景：仓库中的临时文件

我们编译出来了大量临时文件或很大的二进制文件，如 .o, .lib文件，这些文件不想上传。

提出问题： 想要在本文件夹中做版本控制，但需要忽略某些特定的文件

解决方案： 使用.gitignore文件来标记不想要进行版本控制的临时文件。

.gitignore 文件的用法:

.gitignore文件是由我们自己创建, 并默认放置在仓库的根目录。
Git 默认会忽略`.gitignore`中的文件名的大小写, 不过我们可以通过`git config core.ignorecase false`，来设置为不忽略大小写。

1. 文件内容格式

```
vim .gitignore

# 忽略.lib为后缀的文件
*.lib

# libmysql.lib 这个文件不忽略
!libmysql.lib

# 忽略所有的bin文件夹
bin/

# 忽略根目录下的bin文件夹
/bin/
```

2. 已经被忽略的文件如何添加到暂存区: `git add -f <filename>`
3. 已经添加到暂存区中的文件如何忽略: `git rm --cached <filename>`

场景: 不小心提交了一个临时文件

我们对这个临时文件不想做版本跟踪，但是在`.gitignore`文件中添加该文件，这个文件仍然会被追踪。

提出问题: 如何忽略一个已经被追踪的文件?

解决方案:

1. `git rm filename`直接从仓库中删除该文件，并把该改动commit后，随后在`.gitignore`中添加该文件为忽略。
2. `git update-index --assume-unchanged <filename>`, 这个操作不会删除该文件，也不用提交，但命令太长

场景: 需要标记一个特定的版本

当我们的代码进入到比较稳定，或者开发出了一个功能，需要标记一个commit来作为稳定版本的基准。

提出问题： 如何为commit添加标记和备注信息

解决方案： 使用`git tag`为版本打标签

```
git tag -a vx.x.x -m "message"
git push origin tags    ///< 推送到远端
```

场景： 开发一个功能

某项功能可能开发时间较久，但又想把未完成的代码上传到远端版本库，来实现多台电脑同步。

例如：在开发随车通信需求时，在`linux`上编译`dis`，而我开发的环境在`windows`上, 当我在本地`window`开发的临时代码，想要放到`linux`机器上，这时我们需要分支来对代码的同步。

提出问题： 怎样才能在不影响远端仓库的代码的情况下，在远端备份开发过程代码?

解决方法： 使用 git branch
brunch 介绍:
brunch意味着你可以从主分支中，分叉出来一个分支来提交代码而不影响主分支的代码。

```
/// 创建并切换到分支
git checkout -b <branch_name>

/// 做相应的提交，修改
git add .
git commit -m"some comment"

/// 把本地分支上传到远端
git push origin <branch_name>

/// 切换到另一台电脑上，拉去自己的分支
git pull origin <branch_name>
```

合并分支:

```
git checkout develop    ///< 当前处于develop分支下
git merge master   ///< 把master的东西合入到当前分支，方便在自己开发的分支上处理冲突
git checkout master    ///< 切换到master分支
git merge develop    ///< 把develop合并到当前分支
git branch -d future    ///< 把合并过的分支删除
```

场景：临时切换分支

我们会遇到临时切换回主分支的情况。
例如： 当我在开发随车通信功能开发一半时，雷总让我在仓库中提交一个文档。如果我在自己的开发分支上上传该文档，那么在`master`分支上会没有这个文档，其他人也获取不到，所以只能切换回`master`分支上进行上传。

提出问题： 快速切换分支，做完提交，切换回开发分支时，工作区应跟切换分支前一样。

解决方案：
那么现在分为两种情况：

1. 我们工作区没有未被commit的文件，那么我们直接`git checkout <branch_name>`, 即可切换到相应的分支。
2. 我们工作区有很多未被commit的代码，这时我们可以选择，把工作区内代码全部commit或者选择使用`git stash`来临时把未被commit的代码给存储起来， 在我们切换回开发分支时，再把临时存

储的代码拿出来。

`stash`介绍：

```
/// 把所有未commit的文件(工作区、暂存区里的文件)都放入一个临时的分支，使工作区可以切换分支
git stash
或
git stash save "some comment"

/// 切换到其他分支，并做一些提交，并切换回自己的开发分支
git checkout master
git add .
git commit -m"some comment"
git checkout develop

/// 把临时存储的代码给拿出来，放入工作区(之前暂存区的文件在pop后的状态是到工作区)
git stash pop
```

场景： 某个commit，提交错分支了

开发过程中，突然出现了一个`BUG`需要立即修复，我们急着修复，把修复的代码放入了正在大改开发分支上。
提出问题： 我们需要怎样，把主分支上的`BUG`给修正过来
解决方案:

1. 切换到主分支，再次把刚才修改的文件，同样在主分支上进行修改，再次进行提交。
2. 切换到主分支，使用`git cherry-pick <SHA>`把特定提交给放到主分支中。

方案一存在修改的不一致，当后面需要合并分支时，需要处理冲突。
方案二快速提交，不用再次使用手动修改文件。

`git cherry-pick <SHA>`用法示例：
该操作会把特定的commit给，放入当前所在的分支，并产生一个新的提交

之前分支的情况：

```
a - b - c - d - f   Master
        \
        e - f - g Feature
```

`cherry-pick`操作

```
/// 切换到
git checkout master
/// 把提交f给提交到本分支
git cherry-pick f
```

操作后的分支情况：

```
a - b - c - d - f   Master
    \
    e - f - g       Feature
```

场景: commit的信息输入错了

提出问题： 怎么修改提交的commit信息

解决方案：

1. 修改最近一次提交的commit

```
 git commit --amend

 /// 进入到提交的文件里面，默认使用vim打开
 /// 修改好提交信息，保存后退出
```

2. 如果想要修改之前的commit

```
git rebase -i HEAD~3    ///< 回退到HEAD前面第三个commit处

/// 想要修改哪一个提交就把pick换成你想要的操作,如edit
/// 然后执行
git rebase --continue
```

场景: 开发到一半，发现修改的思路有误

我们从远端仓库拉去最新代码，修改过程中，发现修改错误了，想再从已经提交的代码上重新开始。

提出问题： 如何回退版本

解决方案：

1. 没有commit想要回退， 只是清除工作区修改的代码, 如何让当前已经修改过的代码恢复到HEAD的最新提交代码一致, 即清除工作区修改的代码

```
/// 清除所有没有被暂存的改动
git checkout .

/// 清除该文件没有被暂存的改动
git checkout filename
```

2. 想要撤销上一个commit

```
/// 删除工作区改动的代码，撤销最近一次的commit，撤销git add .
/// 注意完成这个操作后，就恢复到了上一次的commit状态。
git reset --hard HEAD^
```

- `--hard` 换成 `--soft`, 则会保留已经暂存和修改的文件
- `HEAD^` 换成 `HEAD~2` 则可以回退两个commit

场景： 想要找到某个特定业务的所有提交

假设我们的commit的信息都是采用模板来填写的，且已经有大量的 `commit` 时候，需要过滤检索一些特定提交信息的 `commit`。

提出问题： 如何使用关键字搜索提交信息

解决方案：
使用git自带的文字搜索功能 `git log --all --grep='TrainPosCall'`, 搜索提交信息中带有 `TrainPosCall` 的 `commit`。

场景： 想要确认代码的改动

想要分步提交修改库函数的文件和修改业务逻辑的文件，需要确认每个文件的改动。

提出问题： 怎么查看已修改的代码对比之前的版本

解决方案：

1. 查看尚未缓存的改动：`git diff`
2. 查看已缓存的改动： `git diff --cached`, `git diff --staged`
3. 查看已缓存的与未缓存的所有改动：`git diff HEAD`
4. 显示摘要而非整个 diff： `git diff --stat`
5. 版本号与版本号之间的差别: `git diff <SHA> <SHA>`

参考资料

https://juejin.cn/post/6844903965625155597

https://www.freecodecamp.org/news/git-cheat-sheet/

https://www.runoob.com/git/git-diff.html

https://www.atlassian.com/git/tutorials/saving-changes/git-diff

https://www.cnblogs.com/ibingshan/p/10783552.html

https://git-scm.com/docs/git-branch

https://stackoverflow.com/questions/7124914/how-to-search-a-git-repository-by-commit-message/7124949#7124949