

# Git教程笔记

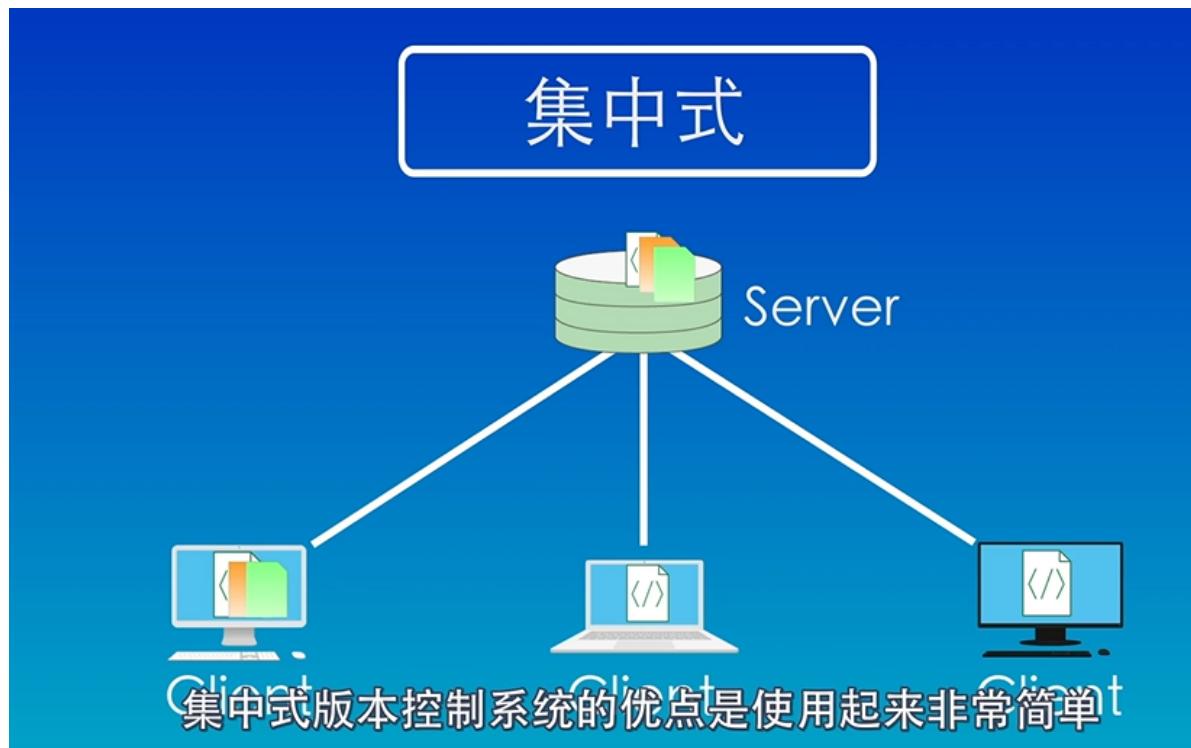
【GeekHour】一小时Git教程 [https://www.bilibili.com/video/BV1HM411377j/?p=6&share\\_source=copy\\_web&vd\\_source=603c1ecf4a2efddfb14a032be6befff](https://www.bilibili.com/video/BV1HM411377j/?p=6&share_source=copy_web&vd_source=603c1ecf4a2efddfb14a032be6befff)

## 1 课程简介

git是一种分布式版本控制系统，可以跟踪每个项目的变化。

### 1.1 集中式和分布式版本控制系统

集中式 SVN：中央处理器的单点故障会引起巨大损失



分布式 Git



## 2 安装和初始化配置

### 2.1 git的使用方式

(1) 命令行 (2) 图形化界面(GUI) (3) IDE插件/扩展

### 2.2 姓名邮箱等初始化

```
~/directory > main git -v  
git version 2.40.0  
~/directory > main git config --global user.name "Jasper Yang"
```

终端的相关插件和配置吧

省略 (Local) : 本地配置, 只对本地仓库有效

--global : 全局配置, 所有仓库生效

--system : 系统配置, 对所有用户生效

那我们使用最多的也就是这个 --global 参数



## 3 新建仓库( *git init* )

版本库/仓库 (Repository简称Repo)

```
git init  
git clone
```

```
yiny@Ares ~/learn-git > git clone https://github.com/geekhall-l  
aoyang/remote-repo.git  
Cloning into 'remote-repo'...  
warning: You appear to have cloned an empty repository.
```

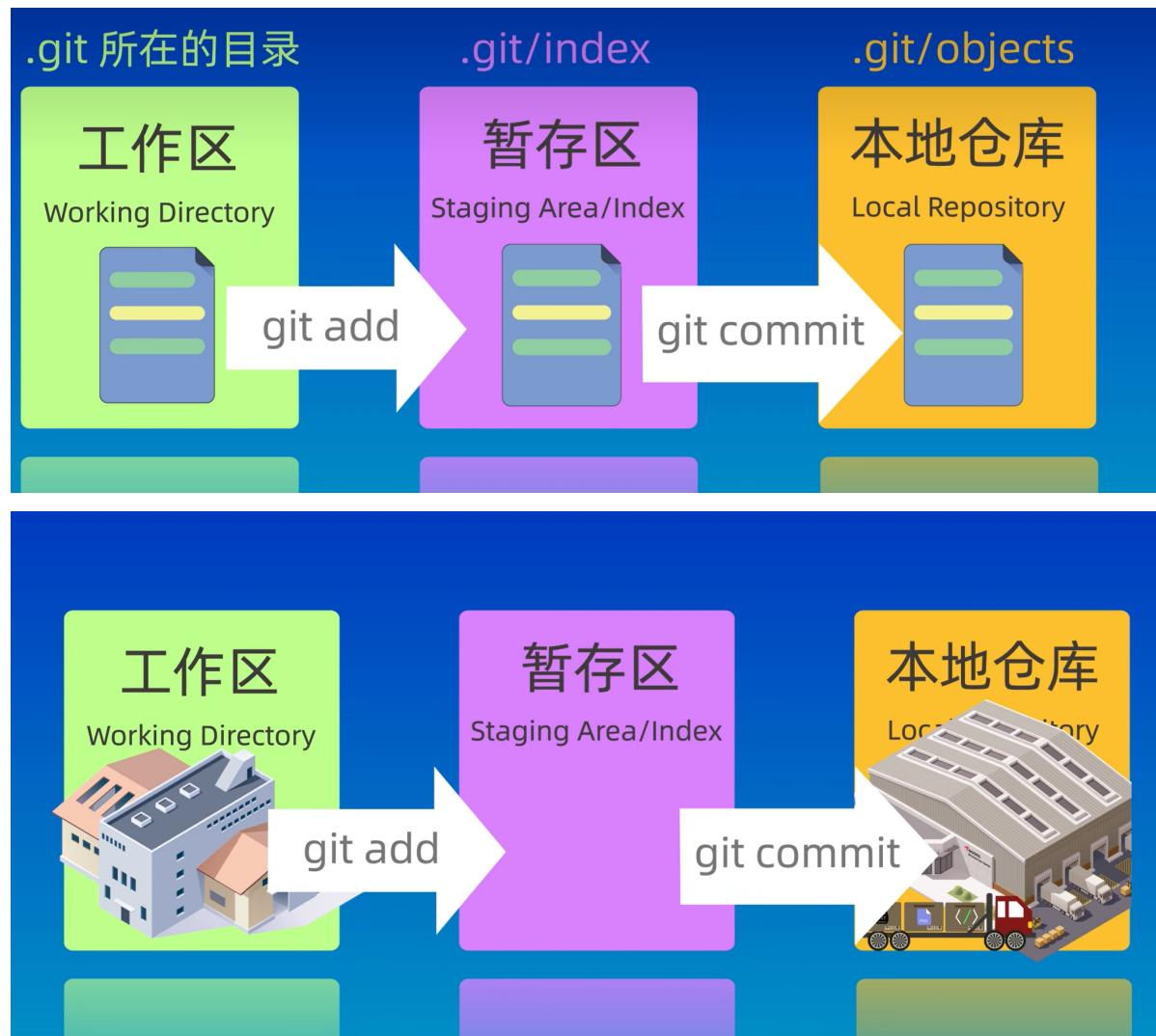
## 4 Git的工作区域和文件状态

Git的本地数据管理分为三个区域

**工作区 (Working Directory)** : 自己工作的目录, 本地资源管理器看到的目录

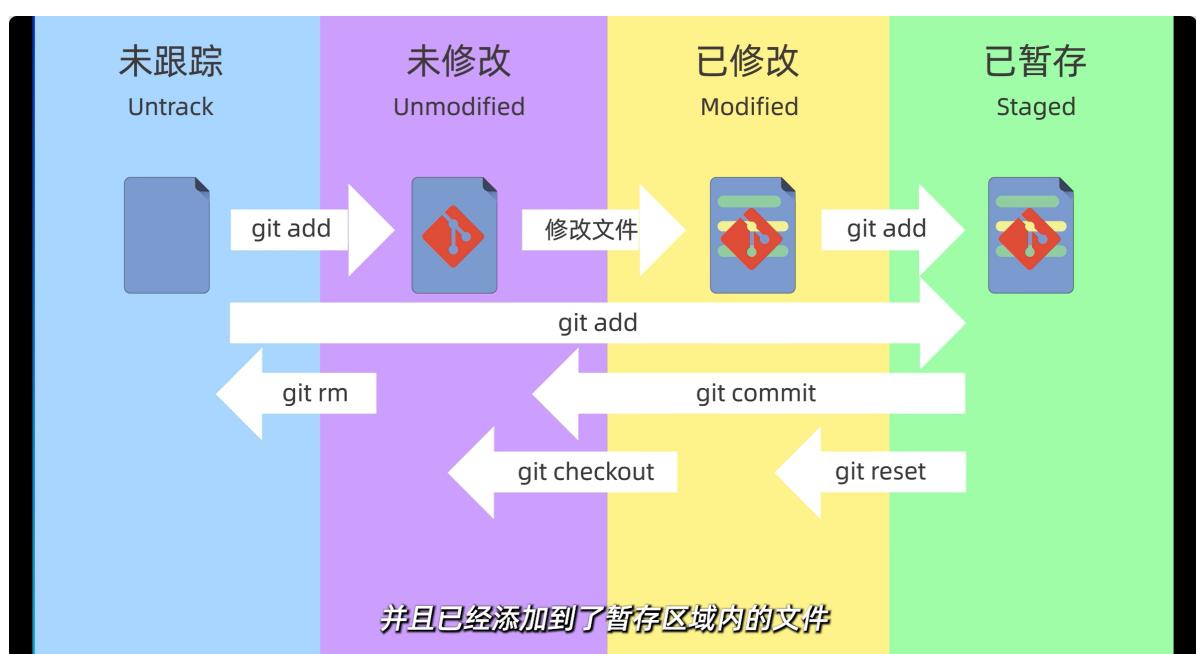
**暂存区 (Staging Area/Index)** : 临时存储区域, 用于保存即将提交到Git仓库的修改内容

**本地仓库 (Local Repository)** : 我们用 *git init* 创建的目录, 是Git存储代码和版本信息的主要位置



车间不需要每次生产都提交到本地仓库，可以先放到暂存区。

## 5 添加和提交文件( *git add & git commit* )



## 总结

|            |  |
|------------|--|
| git status | 查看仓库的状态  |
| git add    | 添加到暂存区<br>可以使用通配符，例如：git add *.txt<br>也可以使用目录，例如：git add . |
| git commit | 提交<br>只提交暂存区中的内容，不会提交工作区中的内容                               |
| git log    | 查看仓库提交历史记录<br>可以使用 --oneline 参数来查看简洁的提交记录                  |



常用：`git commit -am "一些说明"`。

## 6 回退到指定版本(*git reset*)

回退到指定版本

### git reset的三种模式

- git reset --soft
- git reset --hard
- git reset --mixed



那么工作区的内容会被保留



其中 `git reset --soft`、`git reset --mixed` 比较常用。

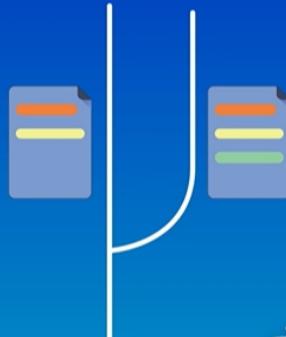
用于撤销某几次提交，当认为多次提交没必要时把多次提交改为一次性提交；

而 `git reset --hard` 不常用，使用 `git reflog` 找到提交版本号，再使用 `git reset --hard 版本号` 回溯已经commit的内容。

## 7 查看不同版本或分支之间的差异(*git diff*)

# git diff

查看工作区、暂存区、本地仓库之间的差异



查看不同版本之间的差异



因为有的时候我们需要在一些没有图形化工具的服务器上来使用Git



什么都不加的话默认是工作区和暂存区的区别: `git diff`

暂存区和版本库: `git diff HEAD/git diff --cached`

不同版本之间: `git diff 版本1提交ID 版本二2提交ID`

提交ID可用HEAD替代HEAD表示当前, HEAD^或HEAD~表示前一个版本, HEAD~n表示前n个版本

还可以在语句后加指定文件名指定对比特定文件: `git diff 版本1提交ID 版本二2提交ID 文件名`

```
~/learn-git/repo          main    git diff 5af90b8 b270efb
~/learn-git/repo          main    git diff b270efb HEAD
~/learn-git/repo          main    git diff HEAD~ HEAD
~/learn-git/repo          main    git diff HEAD^ HEAD
~/learn-git/repo          main    git diff HEAD~2 HEAD
~/learn-git/repo          main    git diff HEAD~3 HEAD
~/learn-git/repo          main    git diff HEAD~3 HEAD file3.txt
```

## 总结

`git diff`

工作区

VS

暂存区

`git diff HEAD`

工作区

+

暂存区

VS

本地仓库

`git diff --cached /  
git diff --staged`

暂存区

VS

本地仓库

`git diff <commit_hash> <commit_hash> /` 比较提交之间的差异  
`git diff HEAD~ HEAD`

`git diff <branch_name> <branch_name> /` 比较分支之间的差异



## 8 删除文件( `git rm` )

直接使用 `rm` 只能删除工作区的文件，需要 `git add + git commit` 才能删除暂存区和本地仓库的东西

使用 `git ls-files` 查看暂存区内容

用 `git rm` 可以删除和工作区和暂存区，然后要再用 `git commit` 更新

## 9 忽略文件( `.gitignore` )



在文件中列出需要忽略的文件

### .gitignore文件的匹配规则

- 空行或者以#开头的行会被Git忽略。一般空行用于可读性的分隔, #一般用作注释
- 使用标准的Blob模式匹配, 例如:
  - 星号 \* 通配任意个字符
  - 问号 ? 匹配单个字符
  - 中括号 [ ] 表示匹配列表中的单个字符, 比如: [abc] 表示a/b/c
- 两个星号 \*\* 表示匹配任意的中间目录
- 中括号可以使用短中线连接, 比如:
  - [0-9] 表示任意一位数字, [a-z]表示任意一位小写字母
- 感叹号 ! 表示取反

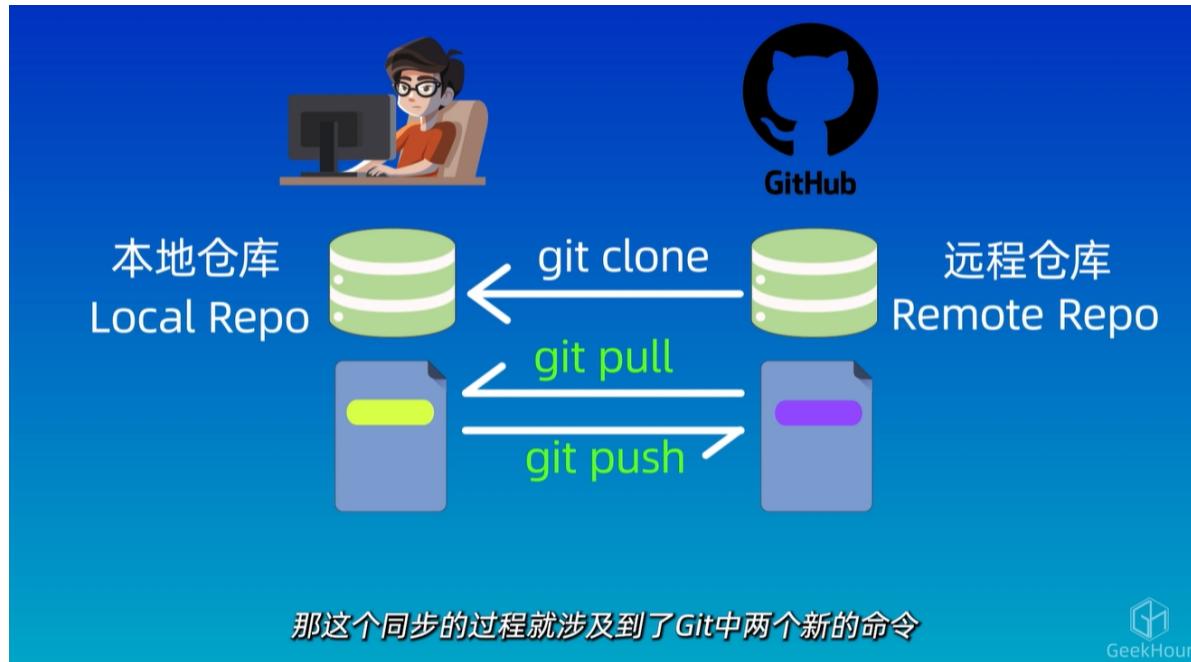
```
# 忽略所有的 .a 文件  
*.a  
  
# 但跟踪所有的 lib.a, 即便你在前面忽略了 .a 文件  
!lib.a  
  
# 只忽略当前目录下的 TODO 文件, 而不忽略 subdir/TODO  
/TODO  
  
# 忽略任何目录下名为 build 的文件夹  
build/  
  
# 忽略 doc/notes.txt, 但不忽略 doc/server/arch.txt  
doc/*.txt  
  
# 忽略 doc/ 目录及其所有子目录下的 .pdf 文件  
".gitignore" 17L但是不会忽略 doc 下面的子目录下面的 txt 文件
```



github上.gitignore预设

[github/gitignore: A collection of useful .gitignore templates](#)

## 10&11 Github账号创建 & SSH配置和克隆仓库( *git clone* )



## 总结

|           |  |
|-----------|--|
| 生成SSH Key | <code>ssh-keygen -t rsa -b 4096</code><br>私钥文件: id_rsa<br>公钥文件: id_rsa.pub |
| 克隆仓库      | <code>git clone repo-address</code>  |
| 推送更新内容    | <code>git push &lt;remote&gt; &lt;branch&gt;</code>                        |
| 拉取更新内容    | <code>git pull &lt;remote&gt;</code>                                       |



## 12 关联本地仓库和远程仓库 (*git remote* & *git pull* & *git push*)

### 总结

|           |   |
|-----------|---|
| 添加远程仓库:   | <code>git remote add &lt;远程仓库别名&gt; &lt;远程仓库地址&gt;</code>                                     |
|           | <code>Step 1</code>   |
| 查看远程仓库:   | <code>git remote -v</code>  |
| 拉取远程仓库内容: | <code>git pull &lt;远程仓库名&gt; &lt;远程分支名&gt;:&lt;本地分支名&gt;</code><br><small>相同可省略冒号后面部分</small> |
|           | <code>Step 2</code>   |

## 13 Gitee和GitLab

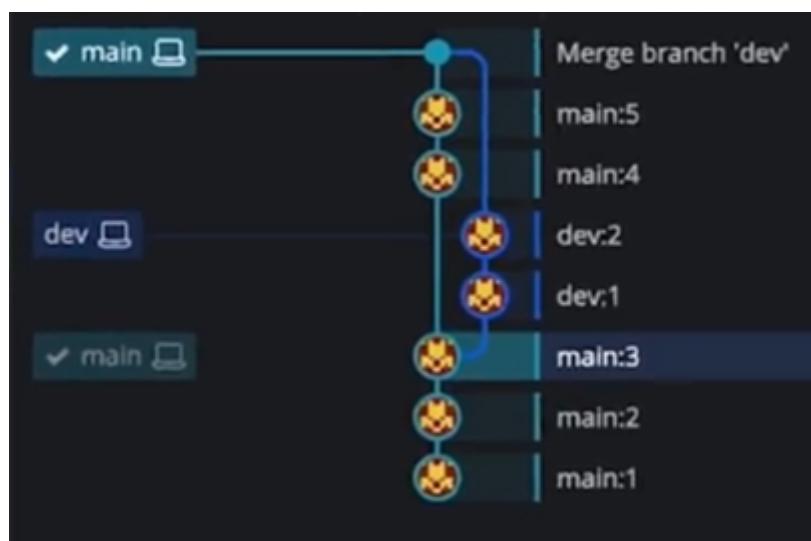
可以把本地仓库同时关联GitHub/Gitee/GitLab的远程仓库

## 14 GUI工具

## 15 VSCode中使用Git

## 16 分支简介和基本操作

## 分支的使用场景



## 总结

查看分支列表:

\$ git branch

创建分支:

\$ git branch branch-name

切换分支:

\$ git checkout branch-name

【推荐】 \$ git switch branch-name

合并分支:

\$ git merge branch-name

删除分支:

【已合并】 \$ git branch -d branch-name

【未合并】 \$ git branch -D branch-name

## 17 解决合并冲突( *git merge* )

A file conflict was found when attempting to merge into main

merge-  
dc29b3  
o8c06e  
**224f84** yesterday  
Merge branch 'dev'  
commit: 224f84  
author: Jasper Yang authored 2023/4/9 @ 23:11  
parent: b4d139,244  
+ 2 added  
Path Tree View all files  
+ dev1.txt  
+ dev2.txt

272

main  
feat  
main

main:6  
feat:1  
Merge branch 'dev'  
main:5  
main:4  
dev:2  
dev:1  
main:3  
main:2  
main:1

Terminal

```
~/learn-git/branch-demo ➜ main ± git commit -am "main:6"  
[main dc29b3a] main:6  
1 file changed, 1 insertion(+)  
~/learn-git/branch-demo ➜ main ➜ git merge feat  
Auto-merging main1.txt  
CONFLICT (content): Merge conflict in main1.txt  
Automatic merge failed; fix conflicts and then commit the result.  
✖ ➜ ~/learn-git/branch-demo ➜ main ± >M< |
```

可以使用 `git diff` 查看，然后手动合并

```

both modified: main1.txt

no changes added to commit (use "git add" and/or "git commit -a")
~/learn-git/branch-demo  ↴ main ±+ >M< git diff
~/learn-git/branch-demo  ↴ main ±+ >M< vi main.txt
~/learn-git/branch-demo  ↴ main ±+ >M< vi main1.txt
~/learn-git/branch-demo  ↴ main ±+ >M< git add .
~/learn-git/branch-demo  ↴ main + >M< git commit -m "merge conflict"
[main 2a07e55] merge conflict
~/learn-git/branch-demo ↴ main

```

可以看到提交之后就自动完成了我们合并的过程

## 总结

两个分支未修改同一个文件的同一处位置：Git 自动合并

两个分支修改了同一个文件的同一处位置：产生冲突

解决方法：

Step1 - 手工修改冲突文件，合并冲突内容

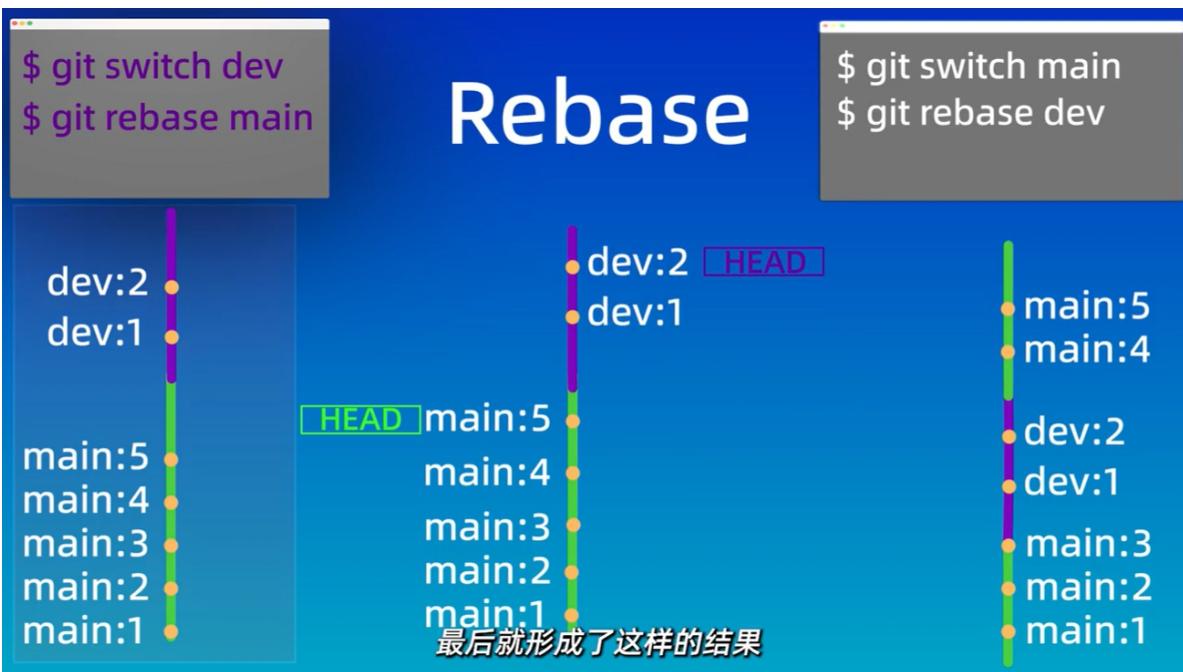
Step2 - 添加暂存区 \$ git add file

Step3 - 提交修改 \$ git commit -m "message"

中止合并：当不想继续执行合并操作时可以使用下面的命令来中止合并过程：

\$ git merge --abort

## 18 回退和变基( *git reset & git rebase* )



### git merge 和 git rebase的区别

前者方便查看提交历史和回滚，后者形成线性历史

## 19 分支管理和工作流模型

Git Flow模型、GitHub Flow模型

**分支命名**

推荐使用带有意义的描述性名称来命名分支

版本发布分支/Tag示例： v1.0.0

功能分支示例： feature-login-page

修复分支示例： hotfix-#issueid-desc

**分支管理**

定期合并已经成功验证的分支, 及时删除已经合并的分支

保持合适的分支数量

为分支设置合适的管理权限

好了 本节课内容就到这里

GeekHour

## 附 Git CheetSheet by GeekHour

| 初始化                                     | Git Cheat Sheet by GeekHour | 添加和提交                                     | 合并分支  | 暂存                       | 重命名远程仓库   |
|---|-----------------------------|---|---|--------------------------|---|
| 初始化设置用户名和邮箱                             |                             | 添加一个文件到仓库                                 | 合并分支<br>合并分支后会生成一个新分支，如果使用`git merge`命令，会把所有修改都合并进当前分支，而如果使用`git merge --no-ff`模式，合并后的历史会变成一条直线 | 移动一个文件到新的位置              | Stash操作可以把当前工作目录“暂存”起来，等以后恢复使用                    |
|   |                             | git add <file>                            | git merge --no-ff -m "message" <branch-name>  | git mv <file> <new-file> | git remote rename <old-name> <new-name>           |
| 添加所有文件到仓库                               |                             |   | 从工作区和暂存区中删除一个文件，然后暂存区继续操作   |                          | 从远程仓库拉取代码   |
| 创建仓库                                    | git add .                   |   | git merge <file> <commit-id>  | git rm <file>            | git pull <remote-name> <branch-name>              |
| 创建一个新的本地仓库（省略`<project-name>`则当前目录为仓库）  | git init <project-name>     | 提交所有暂存区的文件到仓库                             | 只从暂存区中删除一个文件，工作区中的文件没有变化  | git stash list           | 将本地修改的代码push到远程仓库（origin）当分支的代码，然后合并到本地分支         |
| 下载一个远程仓库                                | git clone <url>             | git commit -am "message"                  | 恢复一个文件到之前的版本  | git stash pop            | 将本地修改的代码push到远程仓库（origin）当分支的代码，然后合并到一个子分支，继续修改文件 |
| Git的四个区域                                |                             | git merge --ff -m "message" <branch-name> | 恢复最近一次stash   | git pull --rebase        |   |
| 分支                                      |                             | git branch                                | 创建一个新分支，用来撤销指定的提交，后者的所有变化都将被移除掉，并且应用到当前分支   | git stash apply          |   |
| · 工作区 (Working directory)：就是你在电脑面前看到的目录 |                             | git branch <branch-name>                  | 重置当前分支的HEAD到之前的某个提交，并删除所有之后的提交。`--hard`参数表示直接修改工作区和暂存区，`--soft`参数表示重置暂存区，`--mixed`参数表示重置工作区     | git stash drop stash@{2} | 将本地修改的代码push到远程仓库（origin）当分支的代码，然后合并到本地分支         |
| · 预提交 (Pre-commit)：将对文件的修改在提交前进行校验      |                             | git checkout <branch-name>                | 撤销暂存区的文件，重新放进工作区 (git add的逆操作)  | git stash clear          | git fetch <remote-name>                           |
| · 提交 (Commit)：把修改好的文件提交到本地仓库            |                             | git checkout -b <branch-name>             | git restore --staged <file>   |                          | git branch -r                                     |
| Git的三种状态                                |                             | git branch -D <branch-name>               | git restore --staged <file>   |                          | fetch某一个特定的远端分支                                   |
| 基本概念                                    |                             | git branch -m <branch-name>               | 重置当前分支的HEAD到之前的某个提交，并删除所有之后的提交。`--hard`参数表示直接修改工作区和暂存区，`--soft`参数表示重置暂存区，`--mixed`参数表示重置工作区     |                          | git fetch <remote-name> <branch-name>             |
| · master：默认分支                           |                             | git branch -D <branch-name>               | git stash drop stash@{2}  |                          |   |
| · origin：默认远程仓库                         |                             | git branch -m <branch-name>               | 删除所有的stash  |                          |   |
| · HEAD：指向当前分支的指针                        |                             | git branch -M <branch-name>               |   |                          |   |
| · HEAD@{n}：上一个版本                        |                             | git branch -M <branch-name>               |   |                          |   |
| · HEAD@{n+1}：上一个版本                      |                             | git tag <tag-name>                        |   |                          |   |
| 特殊文件                                    |                             |   |   |                          |   |
| · .gitignore：忽略的文件和目录                   |                             |   |   |                          |   |
| · .gitignore：忽略文件，不需要提交到仓库的文件           |                             |   |   |                          |   |
| · .gitattributes：指定文件的属性，比如执行待          |                             |   |   |                          |   |
| · .gitkeep：让文件保持不变                      |                             |   |   |                          |   |
| · .gitconfig：记录全局的配置信息                  |                             |   |   |                          |   |