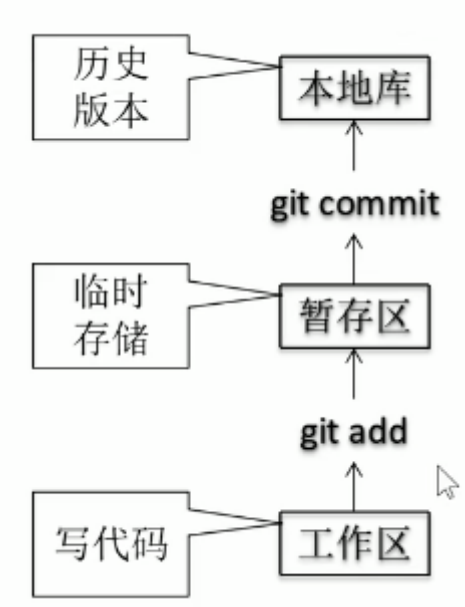
git结构

结构：



git本地使用

本地仓库初始化

```
命令：git init
功能：创建.git 目录
```

设置签名

- 形式：`用户名：xxxx  email地址：xxxx`
- 作用：标识开发人员的身份
- 签名与登录github的账号密码无关
- 优先级：同时存在时，项目级别>系统用户级别。二者至少存在一个。

```
命令：
    项目级别：  git config
        git config user.name 用户名
        git config user.email 邮箱
        配置文件在.git/config
    系统用户级别：  git config --global
        git config --global user.name 用户名
        git config --global user.email 邮箱
        配置文件在~/.gitconfig    (家目录中的.gitconfig文件)
```

git本地命令

```
git status:查看工作区，缓存区状态
git add [filename]：加入暂存区
git commit [filename]：提交到本地库，默认进入vim编辑器模式添加注释
    不进入vim编辑器：git commit -m "注释内容" [filename]
```

git查看历史记录

```
git log
    对每次修改以一行的形式简洁显示：git log --pretty=oneline
    git log --online（只显示当前版本之前的版本）
    git reflog(显示移动步数)，常用
```

版本前进后退的三种操作

- **基于索引值操作（推荐）**

  通过索引值来移动head指针

  `git reset --hard 索引值`

  索引值可在`git reflog`中查看一小段（已经够用了），或者用git log查看一大段

- 使用^符号（只能后退）

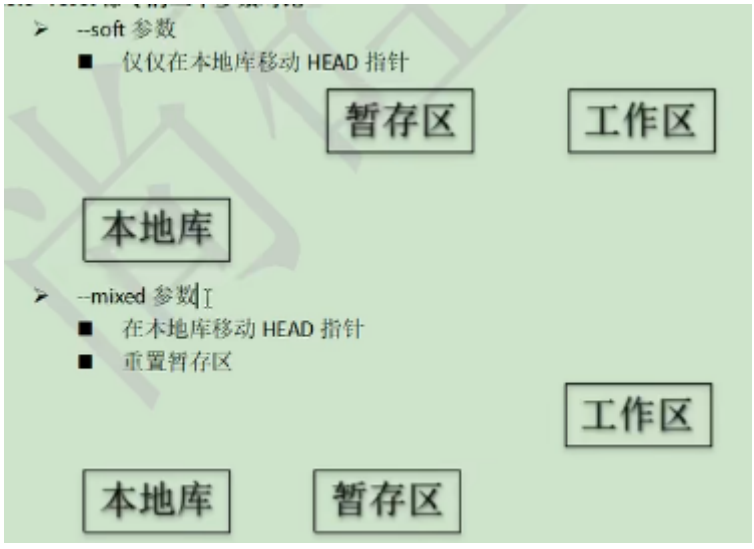
  `git reset --hard HEAD^`

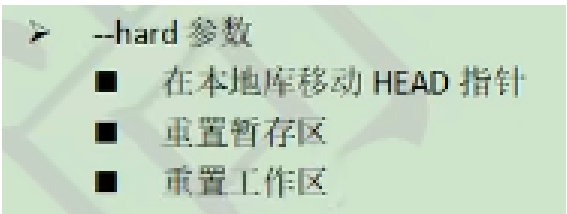
  几个异或符号就往回退几个版本

- 使用~符号（只能后退）

  或者用`git reset --hard HEAD~n`，n是几就回退几步

三个参数 --soft --mixed --hard





soft之后，status显示绿色（代表缓存区有变化但未提交到本地库），因为本地库回退了一个，相当于暂存区和工作区往前进了一个，即相当于执行了`git add`命令

mixed之后，status显示红色，因为本地库、暂存区都向后退了一个，则相当于只有工作区往前进了一个，即相当于对文件进行修改后未执行`git add`

hard则保持一致

删除

```
文件删除后，先git add再git commit
对于删除后已经添加到缓存区内但是未commit的文件进行找回，直接执行命令git reset --hard HEAD
```

删除后找回：`git reset --hard [指针位置]`

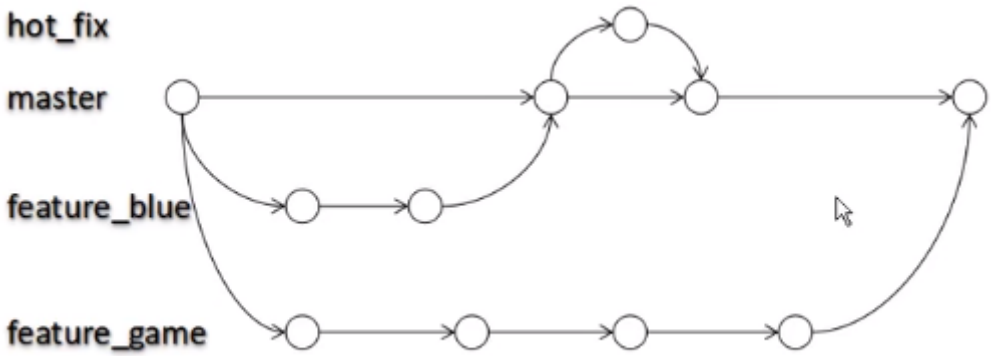
比较文件

git以行为单位进行比较

```
git diff [filename]，默认和暂存区中的进行比较
git diff HEAD/指针 [filename]，则和本地库中的某一版本进行比较
不带文件名只用git diff可以比较多个命令
```

git分支

原理：

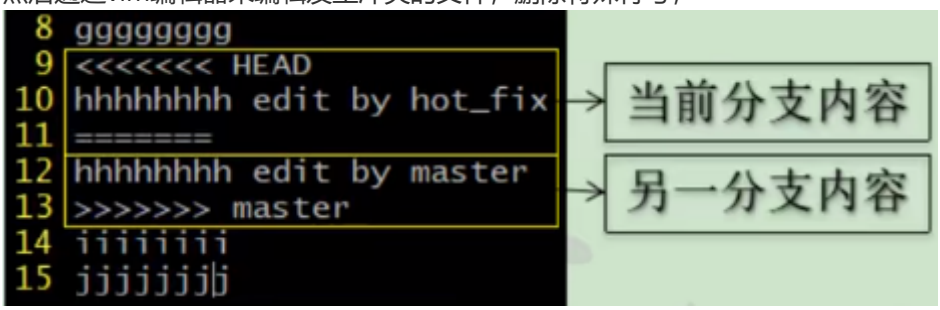



```
查看分支：git branch -v
创建分支：git branch [name]或者 git checkout -b [name] (创建并查看分支)
进入分支：git checkout [name] 或者 git switch [name](可用来更改分支，更常用)
合并分支：切换到接受修改的分支，执行git merge [分支名branchname]
```

合并时冲突

- 如果合并时发生冲突会进入MERGING模式，`hot_fix|MERGING)`

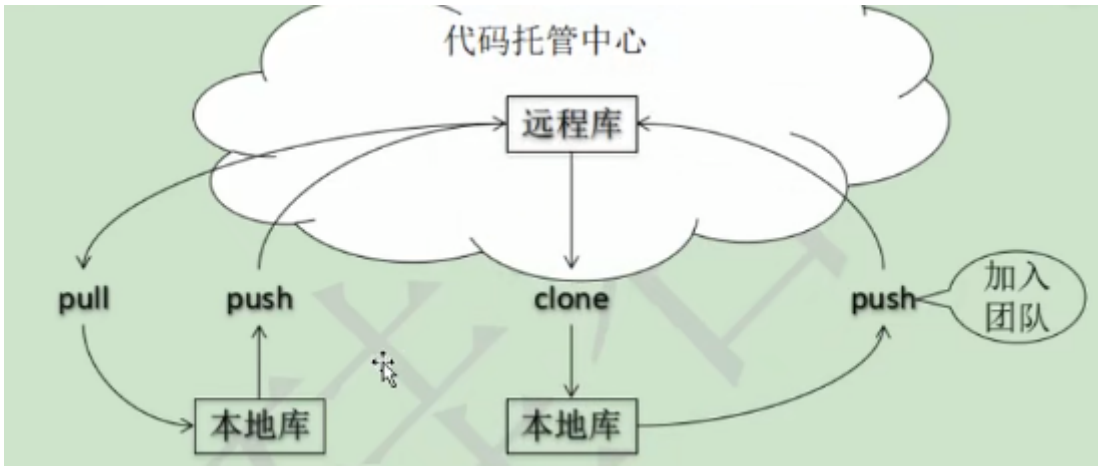
然后通过vim编辑器来编辑发生冲突的文件，删除特殊符号，



保存后，用git add [filename]和git commit （注意，没有filename，只是git commit)来退出MERGING模式。

- 也可以先pull，再合并

本地库与远程库交互




```
连接远程仓库：git remote add 远程仓库别名 github地址
推送：git push 别名 推送的分支
克隆：git clone github地址
```

克隆时，不必初始化，git会自动初始化本地库

注意克隆后再修改，然后再pull时，注意登陆账号的变化，注意远程仓库别名会发生变化，默认为origin

```
查看配置文件：cat .git/config
```

拉取操作

pull = fetch + merge

可以直接pull下来`git pull 别名 master`

如果为了保险，避免冲突，可以先fetch再merge

```
依次执行：
git fetch [别名] [master或者分支]
git checkout [别名/master]，然后cat（此处的/就是单纯的/）（此操作为单纯的看一下代码被修改后的样子）
git checkout master（进入master支）
git merge [别名/master]（此处的/就是单纯的/，代表远程仓库中的master）
```

ssh登陆

1. 设置用户名密码

```
git config --global user.name [name]
git config --global user.email [email]
```

2. 进入当前用户家目录

```
cd ~
```

3. 删除.ssh目录

```
rm -rvf .ssh
```

4. 运行命令

```
ssh-keygen -t rsa -C [email]
```

5. 查看.ssh目录下id_rsa.pub文件内容
6. 复制内容到github中的ssh管理
7. 测试连接，输入yes

   执行命令：

```
ssh -T git@github.com
```

团队协作冲突解决

https://www.bilibili.com/video/BV1pW411A7a5?p=40