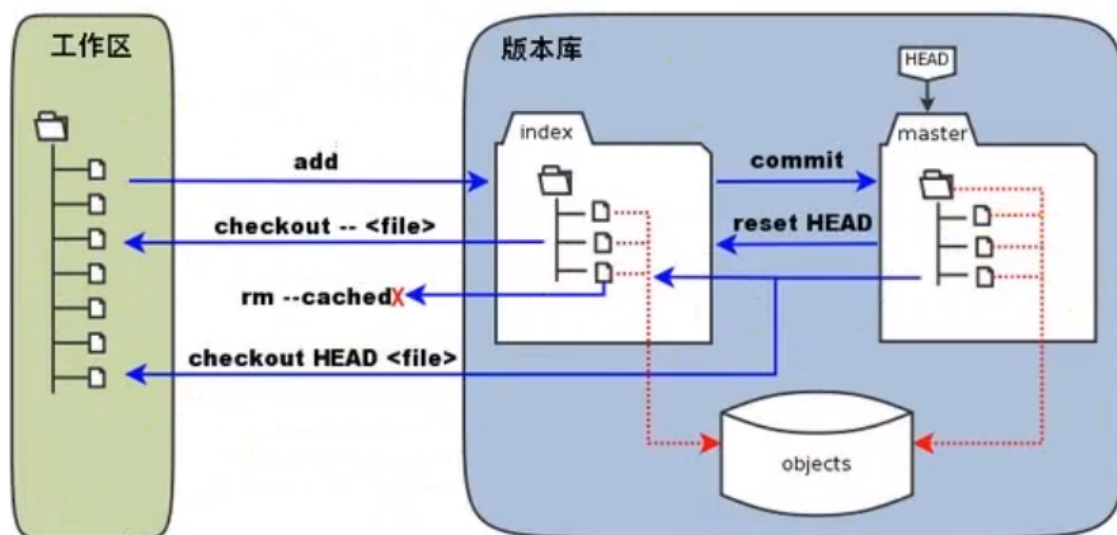


基本概念

- git工作区：工作区就是写代码的地方
- 暂存区：存放将要提交的文件（也叫索引区）
- 本地库：存放本地版本的地方（文件做好准备，向远程的仓库提交）



git和代码托管中心

代码托管中心的任务：维护远程仓库

局域网内的托管中心：gitlab

互联网上的托管中心：github、码云

跨团队协作

git支持跨团队协作，协作方式为第三方若想参与开发，则fork一份代码并clone到本地仓库中进行开发，当觉得代码ok之后发送请求合并到本项目中，若通过申请，则第三方的代码就贡献到了项目中。

git常用命令

git的常用命令可以支持建库，提交，克隆等操作。

创建本地库

```
1 | git init
```

使用此命令初始化一个git本地仓库，很多git命令都需要在本地库中进行，因此这个命令是git的第一个命令。

使用此命令后，git会在此目录上生成一个隐藏文件 ".git"，此隐藏文件就代表了这已经是一个本地库了，并且默认有一个主分支（master）

设置签名

签名：用于标示不同的开发人员身份，

格式：

用户名：root

email（邮箱）：123121@qq.com

这里的签名和远程代码托管中心无任何关系

签名级别

项目级别：仅在当前本地仓库有效

系统级别：登录当前操作系统的用户都有效

一般设置为系统级别

签名命令

```
1 | git config
```

项目级别的签名命令

```
1 | git config user.name puyinzhen
2 | git config user.email 12323@qq.com
```

系统级别的签名命令

```
1 | git config --global user.name puyinzhen
2 | git config --global user.email 12312@qq.com
```

开发过程中一般设置系统级别签名即可

查看工作区和暂存区状态

```
1 | git status
```

此命令用于显示工作目录和暂存区的状态，可以看到哪些修改操作被暂存了，哪些还没有，哪些文件没有被追踪。

将工作区文件写入暂存区

```
1 | git add <file>
```

此命令可以将工作区中的文件写入暂存区

将文件从暂存区中删除

```
1 | git rm --cached <file>
```

此命令可以将放入暂存区的文件删除，注意此时是暂存区的文件被删除了，但是工作区的文件还在，可以重新提交一次即可。但是如果使用

```
1 | git rm <file>
```

则是将工作区和暂存区的文件都删除了。

将暂存区的文件上传到本地仓库中

```
1 | git commit <file>
```

还可以使用快捷提交的方式

```
1 | git commit hello.java -m '第二次提交'
```

此命令可以将暂存区中的文件，以及日志和版本信息提交到本地仓库中。

放弃工作区的文件的修改操作

```
1 | git restore <file>
```

此命令可撤销刚刚对工作区文件的修改操作

若是撤销对文件的add操作，则使用命令

```
1 | git restore --staged <file>
```

从本地仓库退回到暂存区

```
1 | git reset head <file>
```

次命令可以将最近一次提交到本地仓库的文件退回到暂存区

查看提交日志

完整查看日志

```
1 | git log
```

在一行中查看

```
1 | git log --pretty=oneline
```

紧缩hashcode后的样式

```
1 | git log --oneline
```

查看被删除后的日志，次方法即使错误删除了当前的分支或者记录，仍然可以重新回到此版本

```
1 | git reflog
```

版本回退

```
1 | git reset --hard <版本hash值>
```

文件比较

```
1 | git diff <file>
```

默认比较工作区和暂存区的文件。

暂存区和本地库比较

```
1 | git diff head <file>
```

创建远程库别名

```
1 | git remote add <别名> <远程库地址>
```

查看远程库

```
1 | git remote -v
```

向远程库推送

```
1 | git push <远程库地址/地址别名> <分支名>
```

克隆库

```
1 | git clone <远程地址>
```

拉取远程库

第一种：直接到位

```
1 | git pull <远程库/别名> <分支名>
```

第二种：分两步

```
1 | #此操作会将远程库数据拉到本地，但是并没有合并
2 | git fetch <远程库地址/别名> <分支名>
```

```
1 | #此操作将本地和远程库的数据合并
2 | git merge <分支名>
```

实际上pull就等于 fetch+merge

分支

分支就是在开发过程中从主干中另外出一条开发线路

分支的好处：

同时并行推进多个功能开发，提高开发效率，各个分支在开发过程中，如果某一个分支开发失败，不会对其他分支造成影响。

分支操作

查看分支

```
1 | git branch -v
```

创建分支

```
1 | git branch <分支名>
```

切换分支

```
1 | git checkout <分支名>
```

此命令可以切换分支

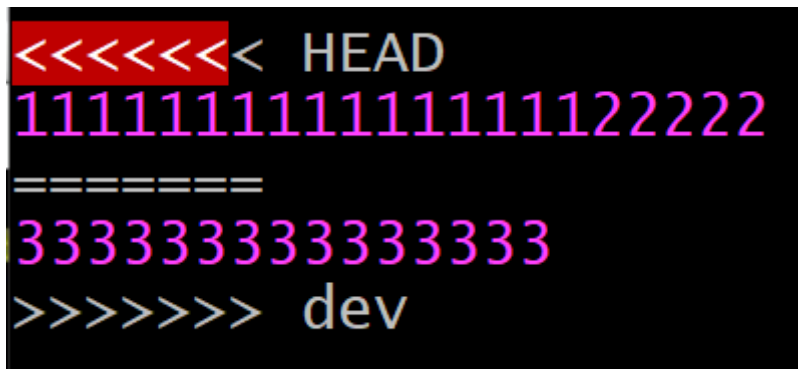
合并分支

合并分支时必须切换到接收分支的那个分支上，基本上是主分支（master）

```
1 | git merge <分支名>
```

解决冲突

当合并操作产生冲突的时候会出现以下情况



```
<<<<<<< HEAD
1111111111111111111122222
====
33333333333333333333
>>>>>>> dev
```

此时需要我们手动去解决冲突。

手动保留正确的内容并且删除多余的符号

然后使用

```
1 | git add <文件名>
```

提交到暂存区

最后使用

```
1 | git commit
```

完成冲突解决，此时不需要写文件名

远程库的冲突解决

当团队中开发时，若A和B两个人修改了同一个文件的同一个地方，那么在向远程库提交的时候，若A先提交，则A提交成功，B之后提交则提交失败，B必须先把A已经提交的数据，pull下来，解决冲突，合并完成之后再提交，则可以成功。

跨团队合作

非团队人员，登录账号之后，fork一份代码库到自己的远程库中，然后pull到本地库进行修改，如果需要合并到原来项目中，则提交合并申请，若通过，则完成了一次跨团队开发

避免代码冲突的方式

为了避免代码冲突，可以尝试再修改文件之前，先更新一下代码，保持本地库的代码和远程库中的代码一致，以这种方式就可以有效的解决代码冲突问题。