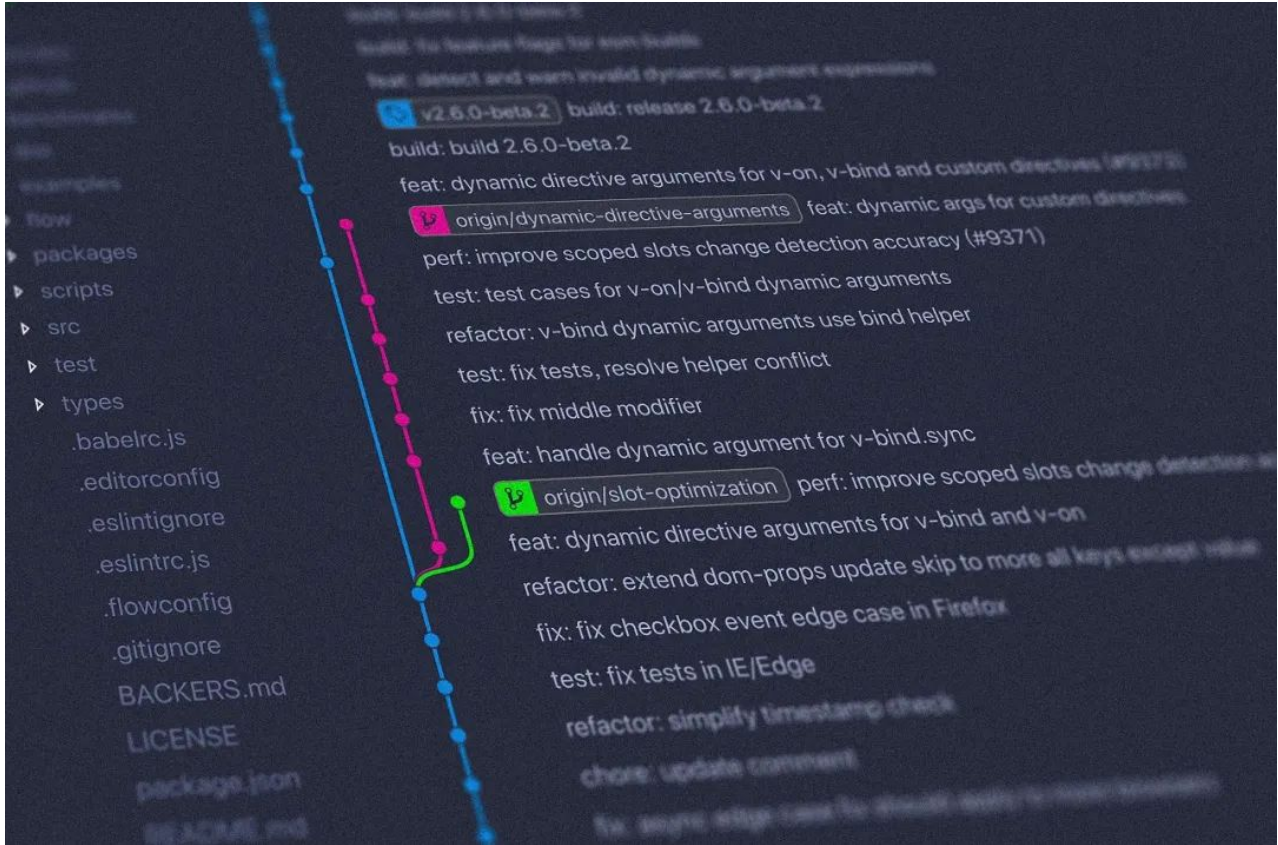


7000+字带你全面搞懂 Git 命令+原理！

mp.weixin.qq.com/s/w6aWNwL6z-dxrWTu-6wYMw

捡田螺的小男孩 [JavaGuide_2020-10-14 09:36](#)



前言

掌握Git命令是每位程序员必备的基础，之前一直是用smartGit工具，直到看到大佬们都是在用Git命令操作的，回想一下，发现有些Git命令我都忘记了，于是写了这篇博文，复习一下~

文章目录

- Git是什么?
- Git的相关理论基础
- 日常开发中，Git的基本常用命令
- Git进阶之分支处理
- Git进阶之处理冲突
- Git进阶之撤销与回退
- Git进阶之标签tag
- Git其他一些经典命令

Git是什么

在回忆Git是什么的话，我们先来复习这几个概念哈~

什么是版本控制？








百度百科定义是酱紫的~

□

版本控制是指对软件开发过程中各种程序代码、配置文件及说明文档等文件变更的管理，是软件配置管理的核心思想之一。

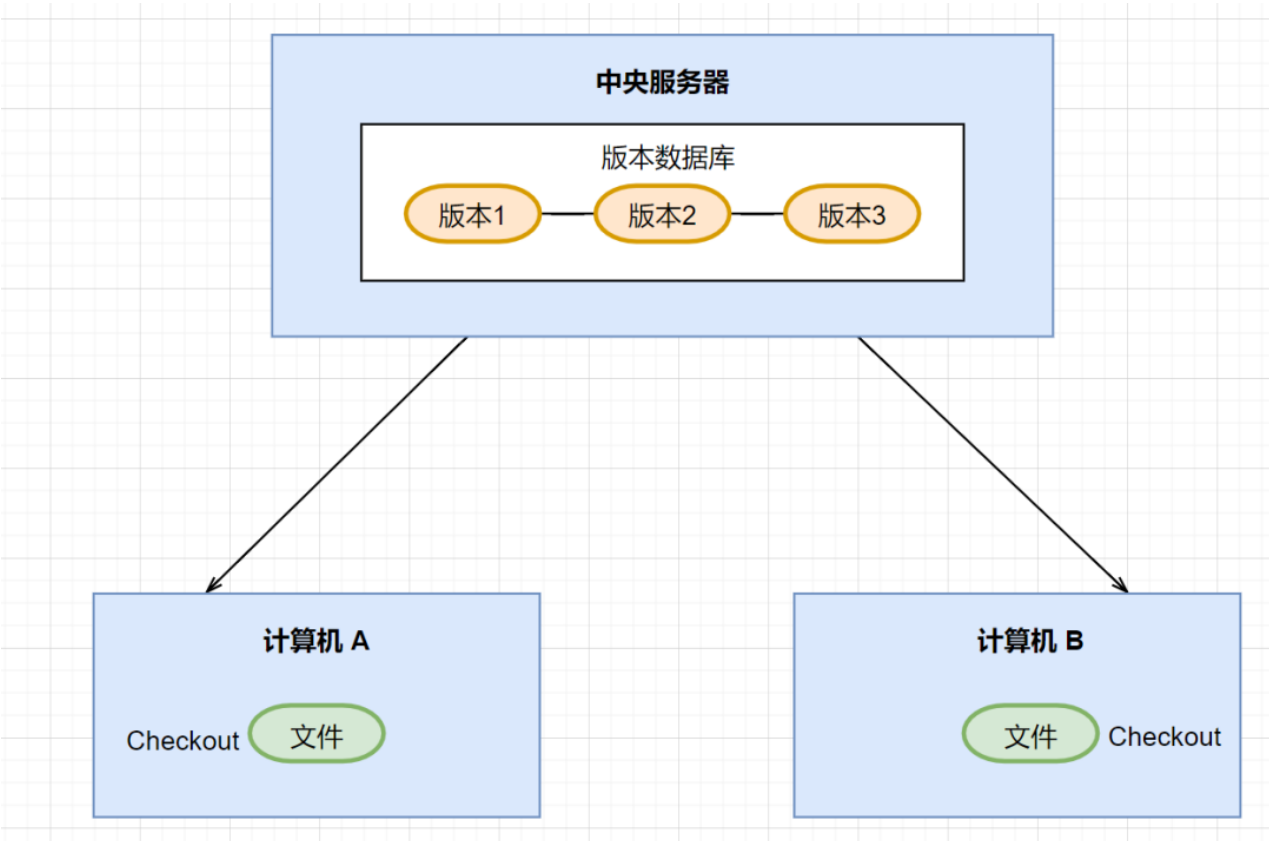
”

那些年，我们的毕业论文,其实就是版本变更的真实写照...脑洞一下，版本控制就是这些论文变更的管理~

 毕业论文.docx	2020/6/20 14:37	Microsoft Word ...	0 KB
 毕业论文不改版.docx	2020/6/20 14:37	Microsoft Word ...	0 KB
 毕业论文最终版.docx	2020/6/20 14:37	Microsoft Word ...	0 KB
 毕业论文最最终版.docx	2020/6/20 14:37	Microsoft Word ...	0 KB
 毕业论文最最终不改版.docx	2020/6/20 14:37	Microsoft Word ...	0 KB
 毕业论文最最终打死不改版.docx	2020/6/20 14:37	Microsoft Word ...	0 KB
 毕业论文最最终打死不改版2.docx	2020/6/20 14:37	Microsoft Word ...	0 KB

什么是集中化的版本控制系统？

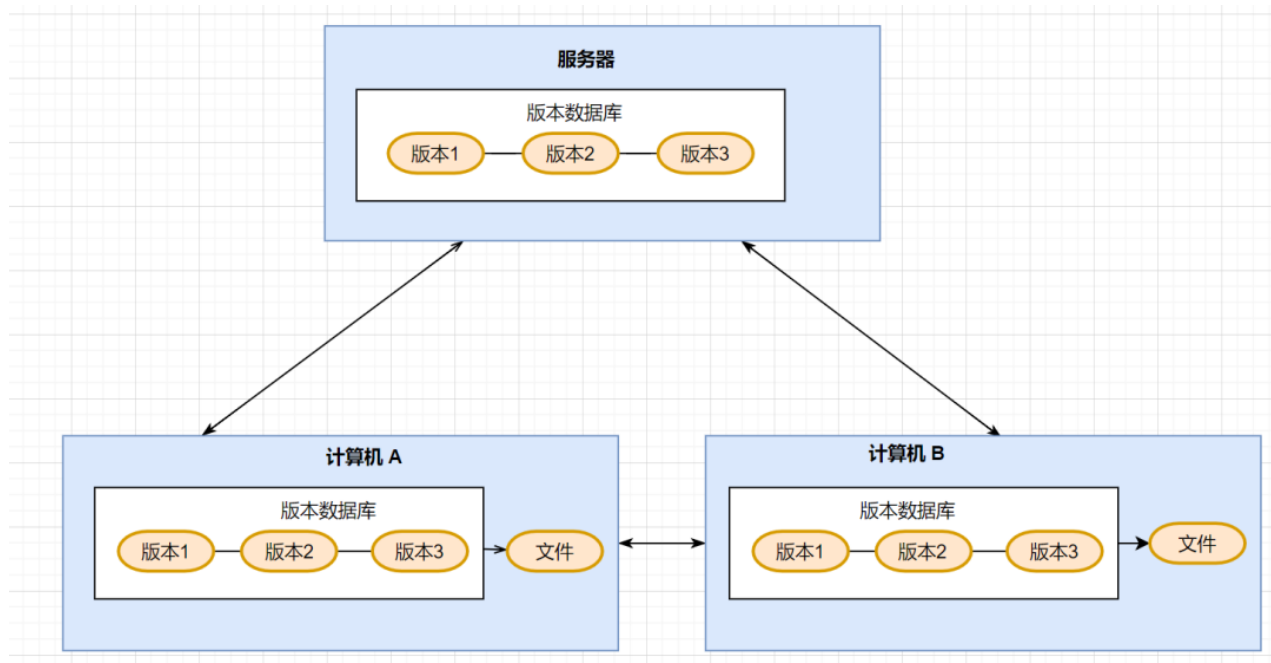
那么，集中化的版本控制系统又是什么呢，说白了，就是有一个集中管理的中央服务器，保存着所有文件的修改历史版本，而协同开发者通过客户端连接到这台服务器，从服务器上同步更新或上传自己的修改。



什么是分布式版本控制系统？

分布式版本控制系统，就是远程仓库同步所有版本信息到本地的每个用户。嘻嘻，这里分三点阐述吧：

- 用户在本地图就可以查看所有的历史版本信息，但是偶尔要从远程更新一下，因为可能别的用户有文件修改提交到远程哦。
- 用户即使离线也可以本地提交，push推送到远程服务器才需要联网。
- 每个用户都保存了历史版本，所以只要有一个用户设备没问题，就可以恢复数据啦~



什么是Git？

Git是免费、开源的**分布式版本控制系统**，可以有效、高速地处理从很小到非常大的项目版本管理。

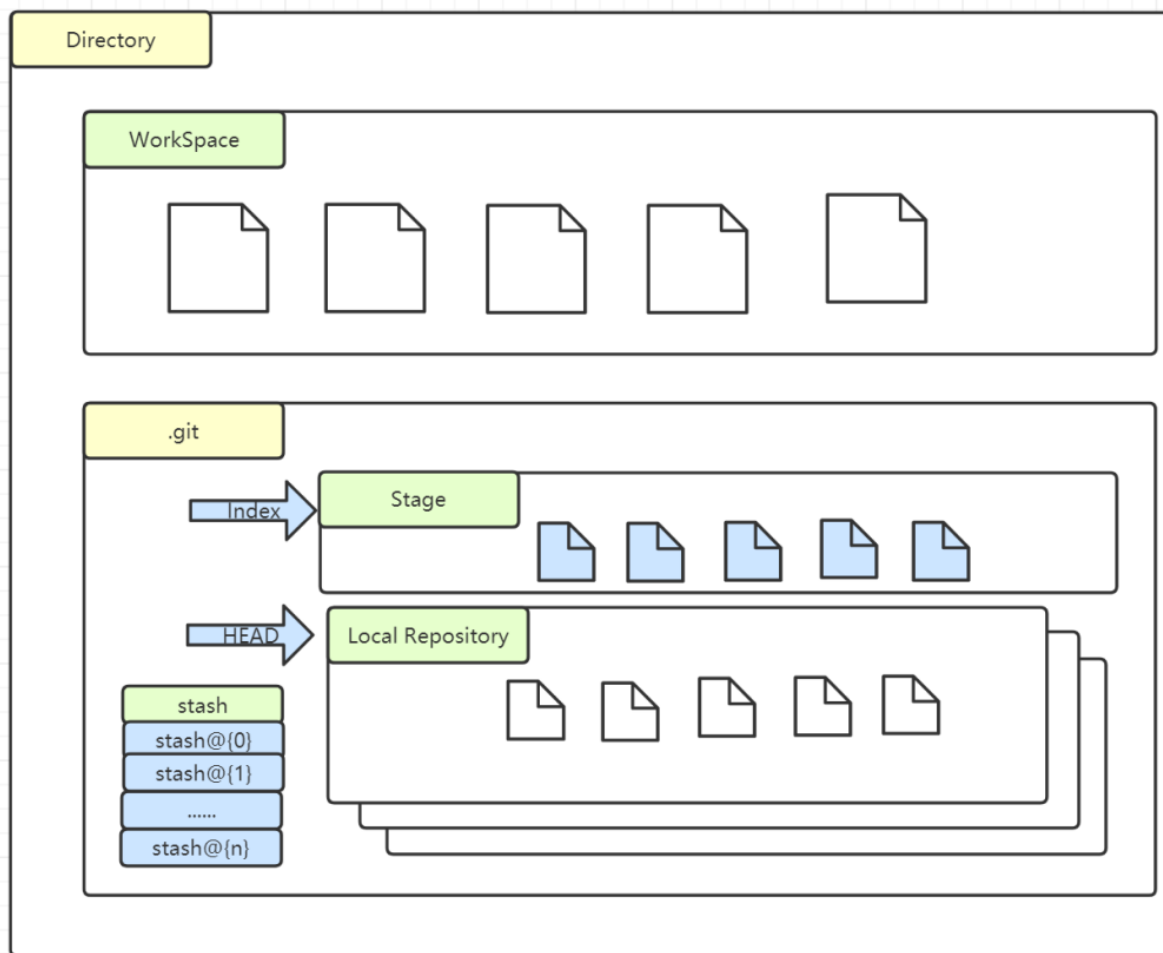


Git的相关理论基础

- Git的四大工作区域
- Git的工作流程
- Git文件的四种状态
- 一张图解释Git的工作原理

Git的四大工作区域

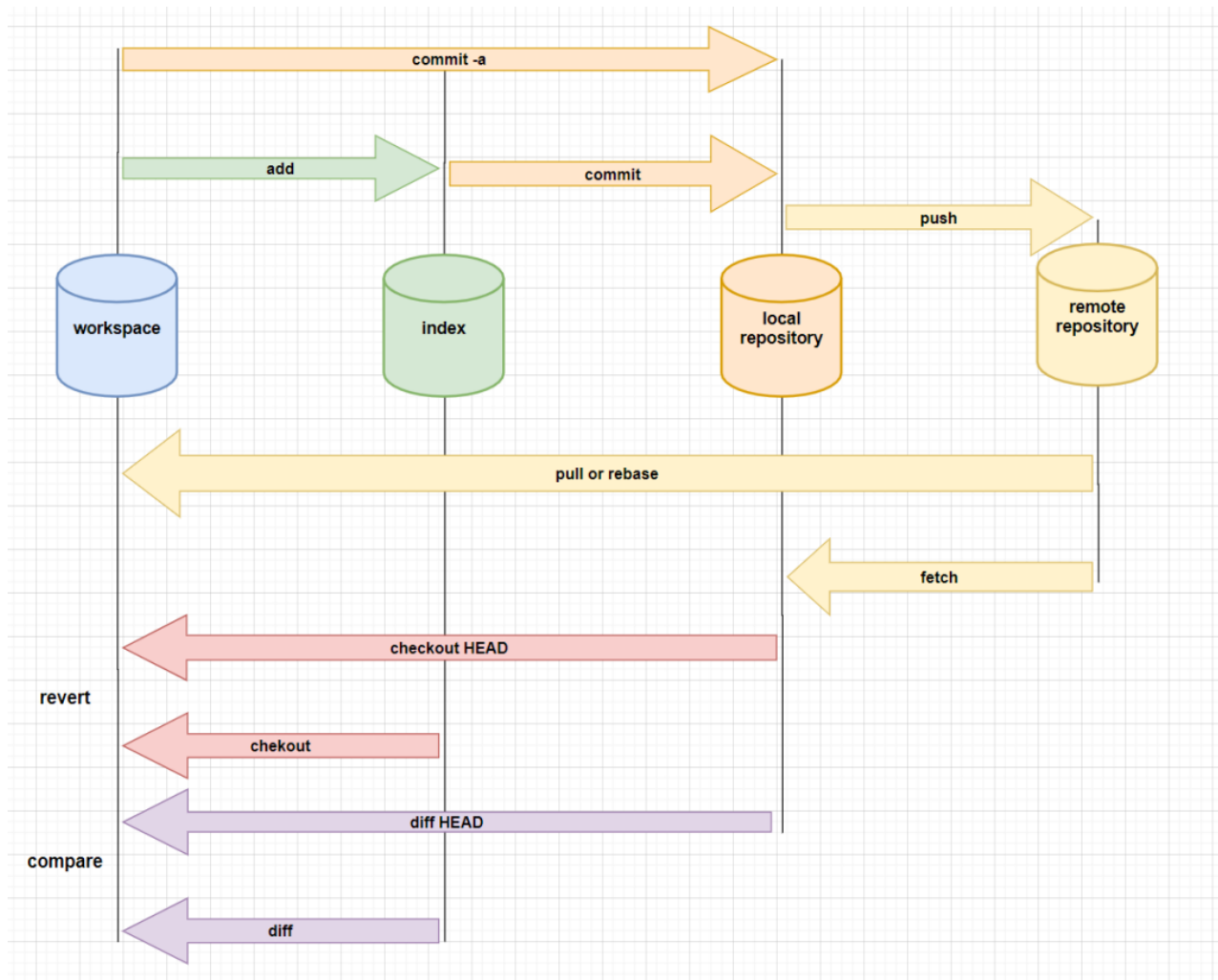
先复习Git的几个工作区域哈：



- **Workspace**：你电脑本地看到的文件和目录，在Git的版本控制下，构成了工作区。
- **Index/Stage**：暂存区，一般存放在 .git 目录下，即 .git/index, 它又叫待提交更新区，用于临时存放你未提交的改动。比如，你执行 git add，这些改动就添加到这个区域啦。
- **Repository**：本地仓库，你执行 git clone 地址，就是把远程仓库克隆到本地仓库。它是一个存放在本地的版本库，其中 **HEAD** 指向最新放入仓库的版本。当你执行 git commit，文件改动就到本地仓库来了~
- **Remote**：远程仓库，就是类似 github，码云等网站所提供的仓库，可以理解为远程数据交换的仓库~

Git的工作流程

上一小节介绍完Git的四大工作区域，这一小节呢，介绍Git的工作流程咯，把git的操作命令和几个工作区域结合起来，个人觉得更容易理解一些吧，哈哈，看图：

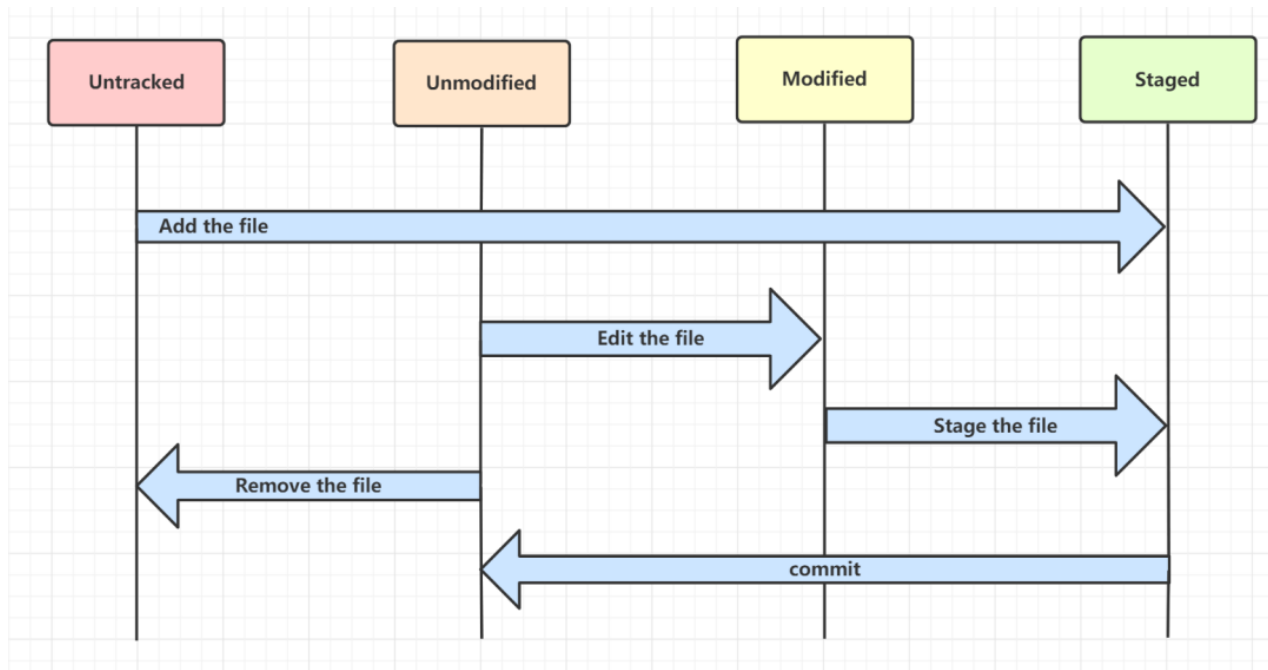


git 的正向工作流程一般就这样：

- 从远程仓库拉取文件代码回来；
- 在工作目录，增删改查文件；
- 把改动的文件放入暂存区；
- 将暂存区的文件提交本地仓库；
- 将本地仓库的文件推送到远程仓库；

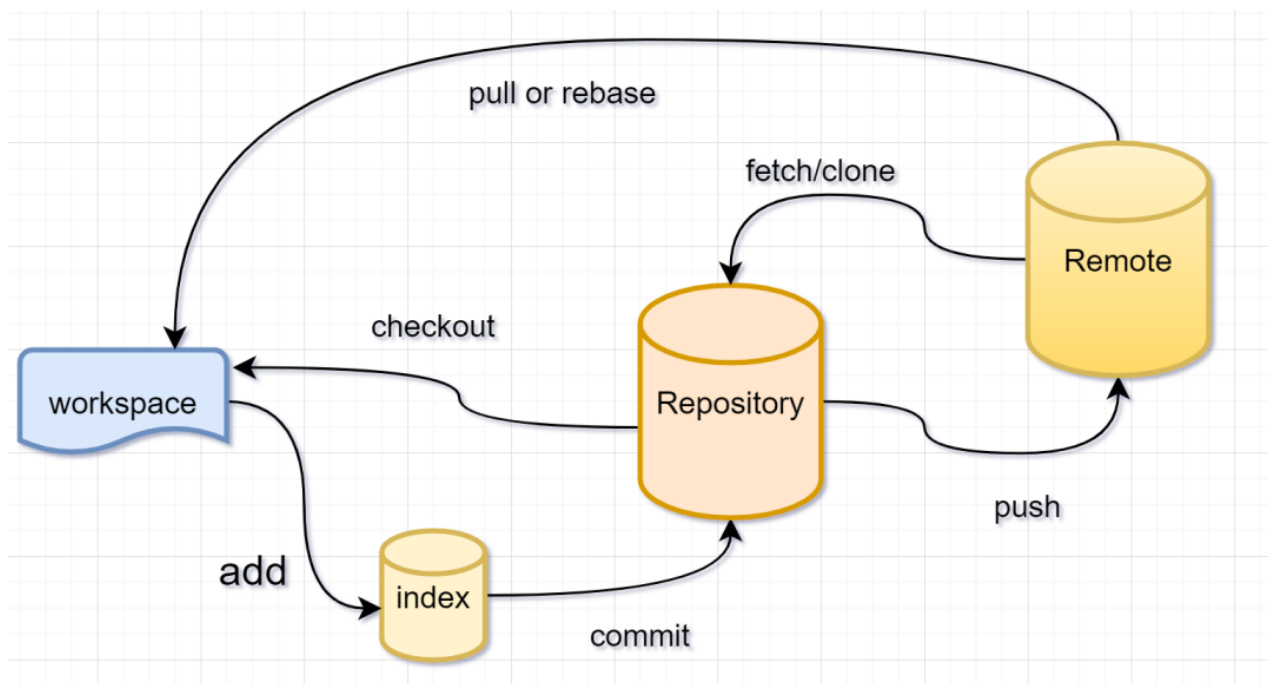
Git文件的四种状态

根据一个文件是否已加入版本控制，可以把文件状态分为：Tracked(已跟踪)和 Untracked(未跟踪)，而tracked(已跟踪)又包括三种工作状态：Unmodified，Modified，Staged



- **Untracked:** 文件还没有加入到git库，还没参与版本控制，即未跟踪状态。这时候的文件，通过git add 状态，可以变为Staged状态
- **Unmodified:** 文件已经加入git库,但是呢，还没修改,就是说版本库中的文件快照内容与文件夹中还完全一致。Unmodified的文件如果被修改,就会变为Modified. 如果使用git remove移出版本库,则成为Untracked文件。
- **Modified:** 文件被修改了，就进入modified状态啦，文件这个状态通过stage命令可以进入staged状态
- **staged:** 暂存状态. 执行git commit则将修改同步到库中,这时库中的文件和本地文件又变为一致, 文件为Unmodified状态.

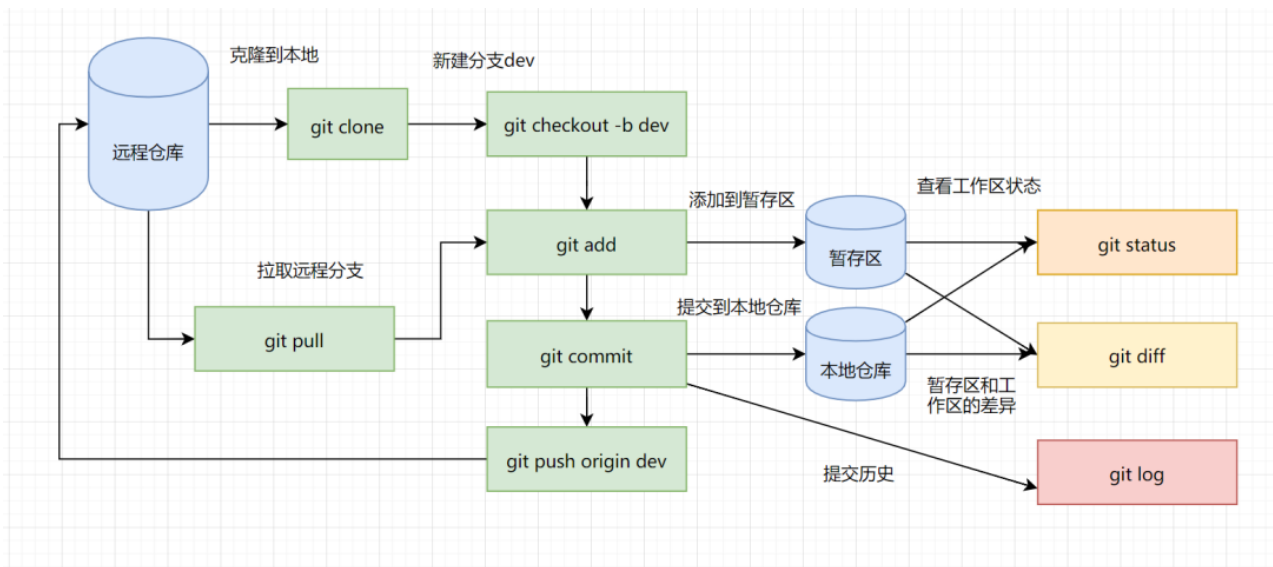
一张图解释Git的工作原理



日常开发中，Git的基本常用命令

- git clone
- git checkout -b dev
- git add
- git commit
- git log
- git diff
- git status
- git pull/git fetch
- git push

这个图只是模拟一下git基本命令使用的大概流程哈~



git clone

当我们要进行开发，第一步就是克隆远程版本库到本地呢

git clone url 克隆远程版本库

```

$ git clone https://github.com/whx123/learngit.git
Cloning into 'learngit'...
remote: Enumerating objects: 18, done.
remote: Total 18 (delta 0), reused 0 (delta 0), pack-reused 18
Unpacking objects: 100% (18/18), done.

```

```
git checkout -b dev
```

克隆完之后呢，开发新需求的话，我们需要新建一个开发分支，比如新建开发分支dev

创建分支：

`git checkout -b dev` 创建开发分支dev，并切换到该分支下

```
mailnavia@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git checkout -b dev
Switched to a new branch 'dev'

mailnavia@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (dev)
$ |
```

git add

git add的使用格式：

git add . 添加当前目录的所有文件到暂存区
git add [dir] 添加指定目录到暂存区，包括子目录
git add [file1] 添加指定文件到暂存区

有了开发分支dev之后，我们就可以开始开发啦，假设我们开发完HelloWorld.java，可以把它加到暂存区，命令如下

git add HelloWorld.java 把HelloWorld.java文件添加到暂存区去

```
mailnavia@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (dev)
$ git add HelloWorld.java
```

git commit

git commit的使用格式：

git commit -m [message] 提交暂存区到仓库区,message为说明信息
git commit [file1] -m [message] 提交暂存区的指定文件到本地仓库
git commit --amend -m [message] 使用一次新的commit，替代上一次提交

把HelloWorld.java文件加到暂存区后，我们接着可以提交到本地仓库啦~

git commit -m 'helloworld开发'

```
mailnavia@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (dev)
$ git commit -m 'helloworld开发'
[dev f8da941] helloworld开发
1 file changed, 5 insertions(+)
create mode 100644 HelloWorld.java
```

git status

git status,表示查看工作区状态，使用命令格式：

git status 查看当前工作区暂存区变动
git status -s 查看当前工作区暂存区变动，概要信息
git status --show-stash 查询工作区中是否有stash（暂存的文件）

当你忘记是否已把代码文件添加到暂存区或者是否提交到本地仓库，都可以用git status看看哦~


```

weibhuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (dev)
$ git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloWorld.java
        modified:   HelloWorld.java.bak

no changes added to commit (use "git add" and/or "git commit -a")

```

git log

git log，这个命令用得应该比较多，表示查看提交历史/提交日志~

git log 查看提交历史 git log --oneline 以精简模式显示查看提交历史 git log -p <file> 查看指定文件的提交历史 git blame <file> 一列表方式查看指定文件的提交历史

嘻嘻，看看dev分支上的提交历史吧~要回滚代码就经常用它喵喵提交历史~

```

weibhuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (dev)
$ git log --oneline
b89b43f (HEAD -> dev) 修复
f8da941 helloworld开发
5ed5b0e (origin/master, origin/HEAD, master) test
e69458a add test.txt
f78f9b0 remove test.txt
9ff0df4 add test.txt
07e0f84 chexiao
3b0a9b8 add under the GPL
119a974 add distributed
9d50194 readme

```

git diff

git diff 显示暂存区和工作区的差异 git diff filepath filepath 路径文件中，工作区与暂存区的比较差异 git diff HEAD filepath 工作区与HEAD (当前工作分支)的比较差异 git diff branchName filepath 当前分支的文件与branchName分支的文件的比较差异 git diff commitId filepath 与某一次提交的比较差异

如果你想对比一下你改了哪些内容，可以用git diff对比一下文件修改差异哦

```

weibhuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (dev)
$ git diff HelloWorld.java
diff --git a/HelloWorld.java b/HelloWorld.java
index f8f7324..d172f20 100644
--- a/HelloWorld.java
+++ b/HelloWorld.java
@@ -1,5 +1,5 @@
 public class HelloWorld {
     public static void main(String[] args) {
-        System.out.println("Hello, World");
+        System.out.println("Hello<A3><AC><BC><F1><CC><EF><C2> <C4>C<C4>K<A2>");
     }
 }
\ No newline at end of file

```

git pull/git fetch

`git pull` 拉取远程仓库所有分支更新并合并到本地分支。`git pull origin master` 将远程master分支合并到当前本地分支
`git pull origin master:master` 将远程master分支合并到当前本地master分支，冒号后面表示本地分支

`git fetch --all` 拉取所有远端的最新代码
`git fetch origin master` 拉取远程最新master分支代码

我们一般都会用`git pull`拉取最新代码看看的，解决一下冲突，再推送代码到远程仓库的。

```
whx123@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git pull
Already up to date.
```

有些伙伴可能对使用`git pull`还是`git fetch`有点疑惑，其实 `git pull = git fetch + git merge`。`pull`的话，拉取远程分支并与本地分支合并，`fetch`只是拉远程分支，怎么合并，可以自己再做选择。

git push

`git push` 可以推送本地分支、标签到远程仓库，也可以删除远程分支哦。

`git push origin master` 将本地分支的更新全部推送到远程仓库master分支。`git push origin -d <branchname>` 删除远程branchname分支
`git push --tags` 推送所有标签

如果我们在dev开发完，或者就想把文件推送到远程仓库，给别的伙伴看看，就可以使用`git push origin dev`

```
whx123@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (dev)
$ git push origin dev
fatal: HttpRequestException encountered.
-----
Username for 'https://github.com': whx123
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 1008 bytes | 201.00 KiB/s, done.
Total 9 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:   https://github.com/whx123/learngit/pull/new/dev
remote:
To https://github.com/whx123/learngit.git
 * [new branch]      dev -> dev
```

Git进阶之分支处理

Git一般都是存在多个分支的，开发分支，回归测试分支以及主干分支等，所以Git分支处理的命令也需要很熟悉的呀~

- git branch
- git checkout
- git merge

git branch

git branch用处多多呢，比如新建分支、查看分支、删除分支等等

新建分支：

git checkout -b dev2 新建一个分支，并且切换到新的分支dev2
git branch dev2 新建一个分支，但是仍停留在原来分支

```
wsl@wsl:~/test/learn-git (dev)
$ git checkout -b dev2
Switched to a new branch 'dev2'
```

查看分支：

git branch 查看本地所有的分支
git branch -r 查看所有远程的分支
git branch -a 查看所有远程分支和本地分支

```
wsl@wsl:~/test/learn-git (dev2)
$ git branch
dev
* dev2
master

wsl@wsl:~/test/learn-git (dev2)
$ git branch -r
origin/HEAD -> origin/master
origin/dev
origin/master

wsl@wsl:~/test/learn-git (dev2)
$ git branch -a
dev
* dev2
master
remotes/origin/HEAD -> origin/master
remotes/origin/dev
remotes/origin/master
```

删除分支：

git branch -D <branchname> 删除本地branchname分支

```

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git branch
dev
dev2
* master

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git branch -D dev2
Deleted branch dev2 (was 5ed5b0e).

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git branch
dev
* master

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ |

```

git checkout

切换分支：

git checkout master 切换到master分支

```

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (dev2)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

```

git merge

我们在开发分支dev开发、测试完成在发布之前，我们一般需把开发分支dev代码合并到master，所以git merge也是程序员必备的一个命令。

git merge master 在当前分支上合并master分支过来 git merge --no-ff origin/dev 在当前分支上合并远程分支dev git merge --abort 终止本次merge，并回到merge前的状态

比如，你开发完需求后，发版需把代码合到主干master分支，如下：

```

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git merge dev
Updating 5ed5b0e..05a16fb
Fast-forward
 HelloWorld.java | 5 +++++
 1 file changed, 5 insertions(+)
 create mode 100644 HelloWorld.java

```

Git进阶之处理冲突

Git版本控制，是多个人一起搞的，多个分支并存的，这就难免会有冲突出现~

Git合并分支，冲突出现

同一个文件，在合并分支的时候，如果同一行被多个分支或者不同人都修改了，合并的时候就会出现冲突。

举个栗子吧，我们现在在dev分支，修改HelloWorld.java文件，假设修改了第三行，并且commit提交到本地仓库，修改内容如下：


```
public class HelloWorld {    public static void main(String[] args) {  
System.out.println("Hello，捡田螺的小男孩！");    }}
```

我们切回到master分支，也修改HelloWorld.java同一位置内容，如下：

```
public class HelloWorld {    public static void main(String[] args) {  
System.out.println("Hello，jay！！");    }}
```

再然后呢，我们提交一下master分支的这个改动，并把dev分支合并过下，就出现冲突啦，如图所示：

```
weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)  
$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git checkout -- <file>..." to discard changes in working directory)  
  
        modified:   HelloWorld.java  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
        HelloWorld.java.bak  
  
no changes added to commit (use "git add" and/or "git commit -a")  
weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)  
$ git add .  
  
weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)  
$ git commit -m 'master modified'  
[master 0b92600] master modified  
2 files changed, 6 insertions(+), 1 deletion(-)  
create mode 100644 HelloWorld.java.bak  
  
weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)  
$ git merge dev  
Auto-merging HelloWorld.java.bak  
CONFLICT (add/add): Merge conflict in HelloWorld.java.bak  
Auto-merging HelloWorld.java  
CONFLICT (content): Merge conflict in HelloWorld.java  
Automatic merge failed; fix conflicts and then commit the result.
```



Git解决冲突

Git 解决冲突步骤如下：

- 查看冲突文件内容
- 确定冲突内容保留哪些部分，修改文件
- 重新提交，done

1.查看冲突文件内容

git merge提示冲突后，我们切换到对应文件，看看冲突内容哈，，如下：

```

weihuanjie@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master|MERGING)
$ cat HelloWorld.java
public class HelloWorld {
    public static void main(String[] args) {
<<<<<<< HEAD
        System.out.println("Hello jay");
=====
        System.out.println("Hello  Ck");
>>>>>>> dev
    }
}

```

2.确定冲突内容保留哪些部分，修改文件

- Git用<<<<<<<，=====，>>>>>>>标记出不同分支的内容，
- <<<<<<<HEAD是指主分支修改的内容，>>>>>>> dev是指dev分支上修改的内容

所以呢，我们确定到底保留哪个分支内容，还是两个分支内容都保留呢，然后再去修改文件冲突内容~

3.修改完冲突文件内容，我们重新提交，冲突done

```

weihuanjie@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master|MERGING)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
  new file:   test.txt

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   HelloWorld.java
    both added:      HelloWorld.java.bak

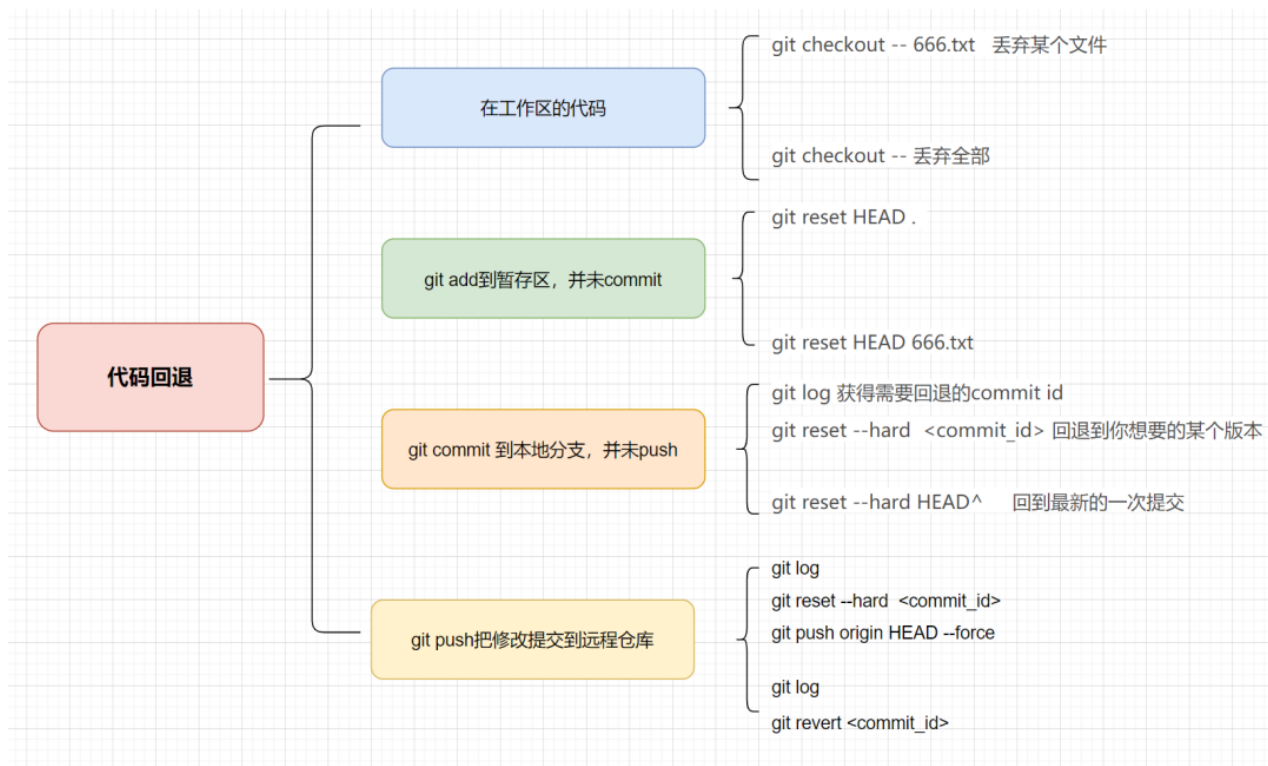
weihuanjie@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master|MERGING)
$ git add .

```

Git进阶之撤销与回退

Git的撤销与回退，在日常工作中使用的比较频繁。比如我们想将某个修改后的文件撤销到上一个版本，或者想撤销某次多余的提交，都要用到git的撤销和回退操作。

代码在Git的每个工作区域都是用哪些命令撤销或者回退的呢，如下图所示：



有关于Git的撤销与回退，一般就以下几个核心命令

- `git checkout`
- `git reset`
- `git revert`

git checkout

如果文件还在**工作区**，还没添加到暂存区，可以使用`git checkout`撤销

`git checkout [file]` 丢弃某个文件file
`git checkout .` 丢弃所有文件

以下demo，使用`git checkout -- test.txt` 撤销了test.txt的修改

```

$ git status
On branch dev
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   test.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   test.txt

root@kali:~/test/learngit (dev)
$ git diff test.txt
diff --git a/test.txt b/test.txt
index ed4df3c..de11717 100644
--- a/test.txt
+++ b/test.txt
@@ -1,2 @@
-666
\ No newline at end of file
+666
+哈哈
root@kali:~/test/learngit (dev)
$ git checkout -- test.txt
root@kali:~/test/learngit (dev)
$ git diff test.txt
root@kali:~/test/learngit (dev)
$ cat test.txt
666

```

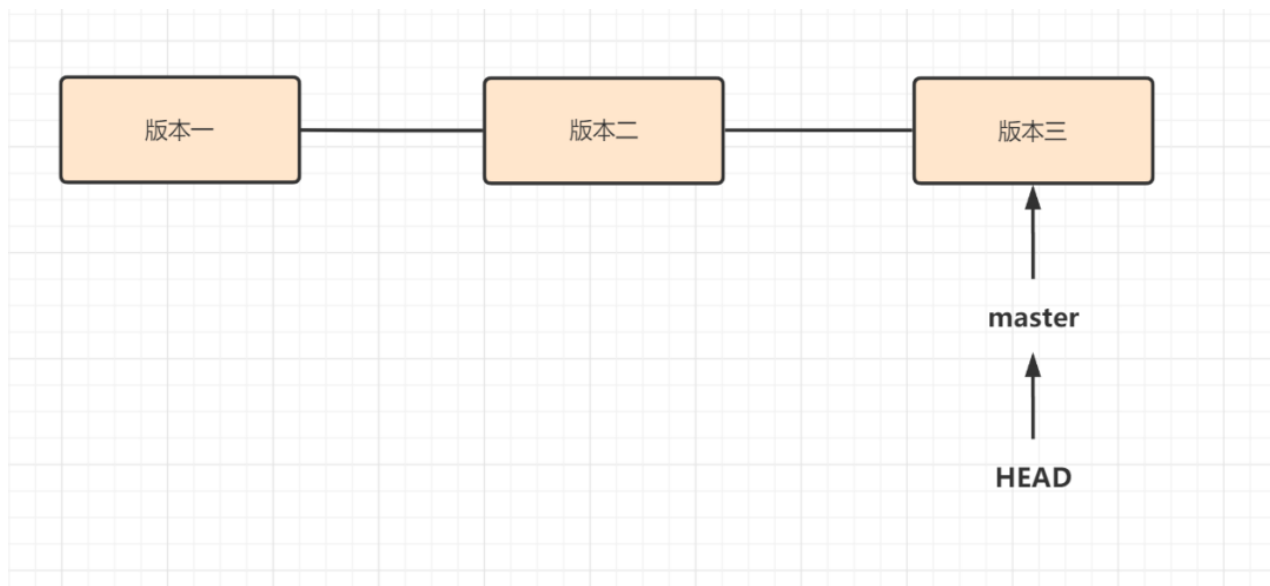
git reset

git reset的理解

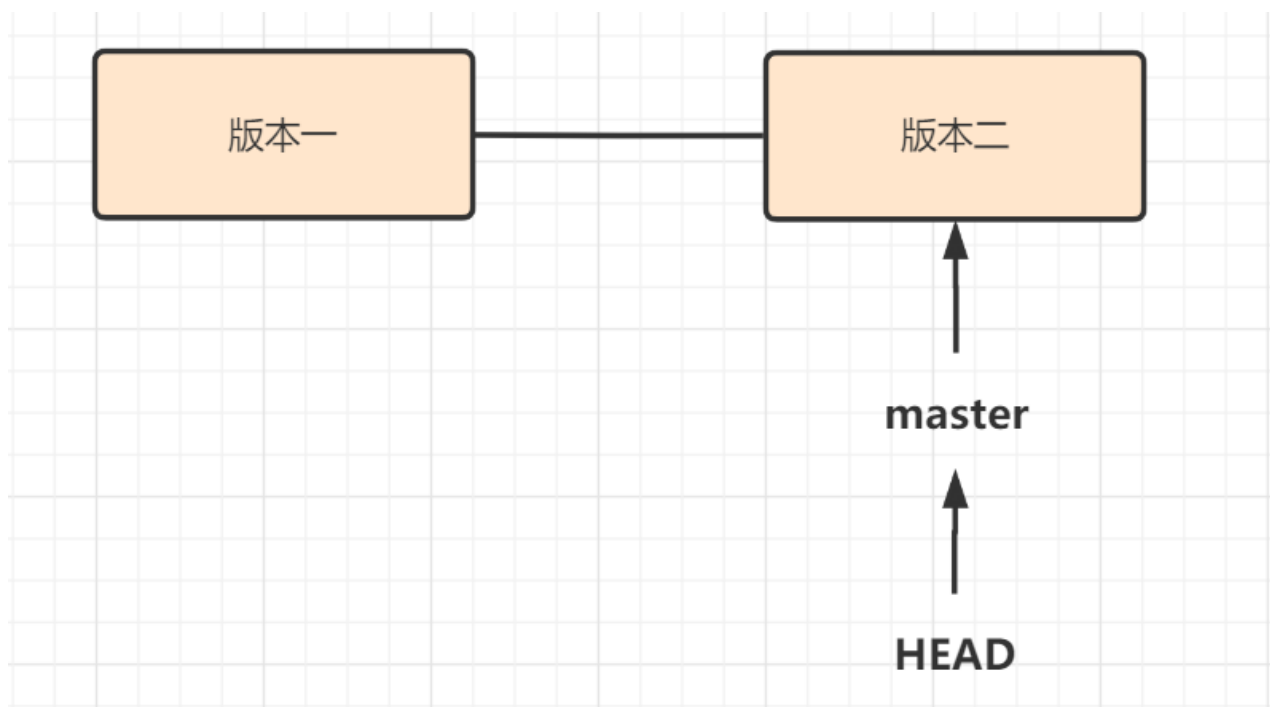
git reset的作用是修改HEAD的位置，即将HEAD指向的位置改变为之前存在的某个版本。

为了更好地理解git reset，我们来回顾一下Git的版本管理及HEAD的理解

Git的所有提交，会连成一条时间轴线，这就是分支。如果当前分支是master，HEAD指针一般指向当前分支，如下：

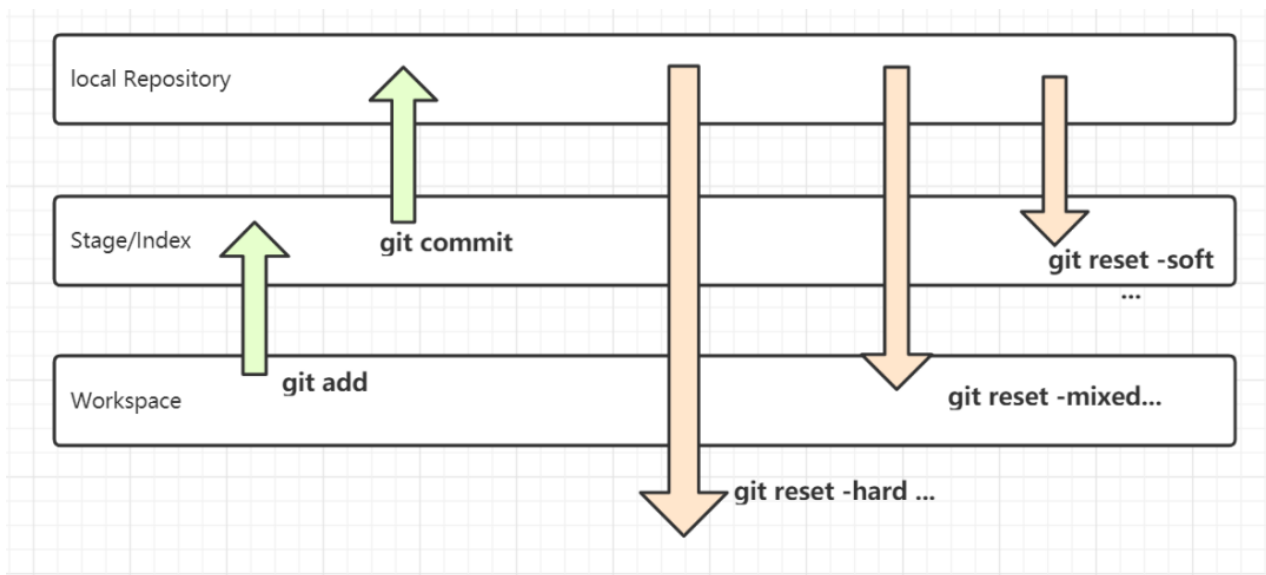


假设执行git reset，回退到版本二之后，版本三不见了哦,如下：



git reset的使用

Git Reset的几种使用模式



git reset HEAD --file 回退暂存区里的某个文件，回退到当前版本工作区状态
 git reset --soft 目标版本号 可以把版本库上的提交回退到暂存区，修改记录保留
 git reset --mixed 目标版本号 可以把版本库上的提交回退到工作区，修改记录保留
 git reset --hard 目标版本号 可以把版本库上的提交彻底回退，修改的记录全部revert。

先看一个栗子demo吧，代码git add到暂存区，并未commit提交,可以酱紫回退，如下：

git reset HEAD file 取消暂存 git checkout file 撤销修改

```
w3333333@LAPTOP-QKRMA6P8 MINGW64 ~/test/learn git (dev)
$ git status
On branch dev
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   test.txt

w3333333@LAPTOP-QKRMA6P8 MINGW64 ~/test/learn git (dev)
$ git reset HEAD test.txt

w3333333@LAPTOP-QKRMA6P8 MINGW64 ~/test/learn git (dev)
$ git status
On branch dev
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    test.txt

nothing added to commit but untracked files present (use "git add" to track)
```

再看另外一个栗子吧，代码已经git commit了，但是还没有push：

git log 获取到想要回退的commit_id
 git reset --hard commit_id 想回到过去，回到过去的commit_id

```

weihua123@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git log
commit 44cc9167ef0527965b9f28f3f1c1aa70b921f25c (HEAD -> master)
Author: whx123 <327658337@qq.com>
Date: Sat Jun 27 08:53:46 2020 +0800

    又修改了test.txt文件

commit 397cd6bc6bf1af676f9acb45883241fc5c9d01a8a
Author: whx123 <327658337@qq.com>
Date: Sat Jun 27 08:49:35 2020 +0800

    修改test文件

commit 8901fd8862d4f1264ac472bea58d63df78e76816 (origin/master, origin/HEAD)
Merge: 0b92600 6e6d760
Author: whx123 <327658337@qq.com>
Date: Fri Jun 26 23:51:32 2020 +0800

    解决冲突

commit 0b926001c99db5e85614275c938556e7dd5635be
Author: whx123 <327658337@qq.com>
Date: Fri Jun 26 22:14:41 2020 +0800

    master modified

commit 6e6d76052f67302549adb1585b270d6b982aa65a (dev)
Author: whx123 <327658337@qq.com>
Date: Fri Jun 26 22:10:57 2020 +0800

    ev modiffied

commit 15fb5166f71692b447b4a2dfa25bfa4d7bc4e5b4
Author: whx123 <327658337@qq.com>
Date: Fri Jun 26 22:06:22 2020 +0800

    修改

commit e5801a17d38d1608829a11196a322b4e3e193391
Author: whx123 <327658337@qq.com>
Date: Fri Jun 26 21:33:38 2020 +0800

    修改

weihua123@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git reset --hard 397cd6bc6bf1af676f9acb45883241fc5c9d01a8a
HEAD is now at 397cd6c 修改test文件

```

git log

回退

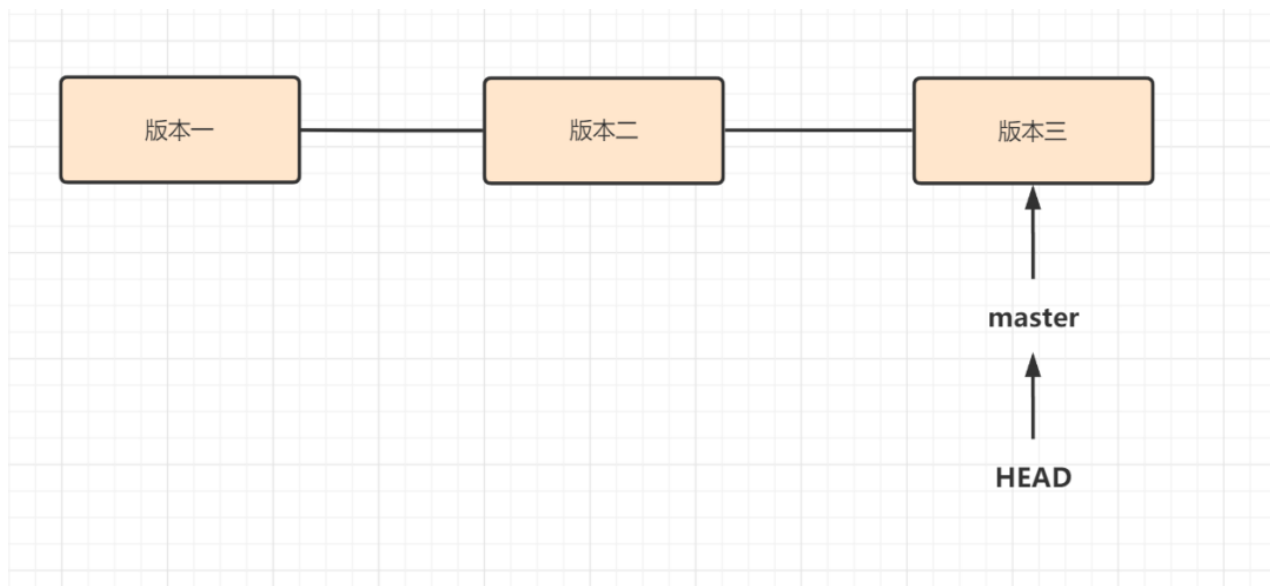
如果代码已经push到远程仓库了呢，也可以使用reset回滚哦(这里大家可以自己操作实践一下哦)~

```
git loggit reset --hard commit_idgit push origin HEAD --force
```

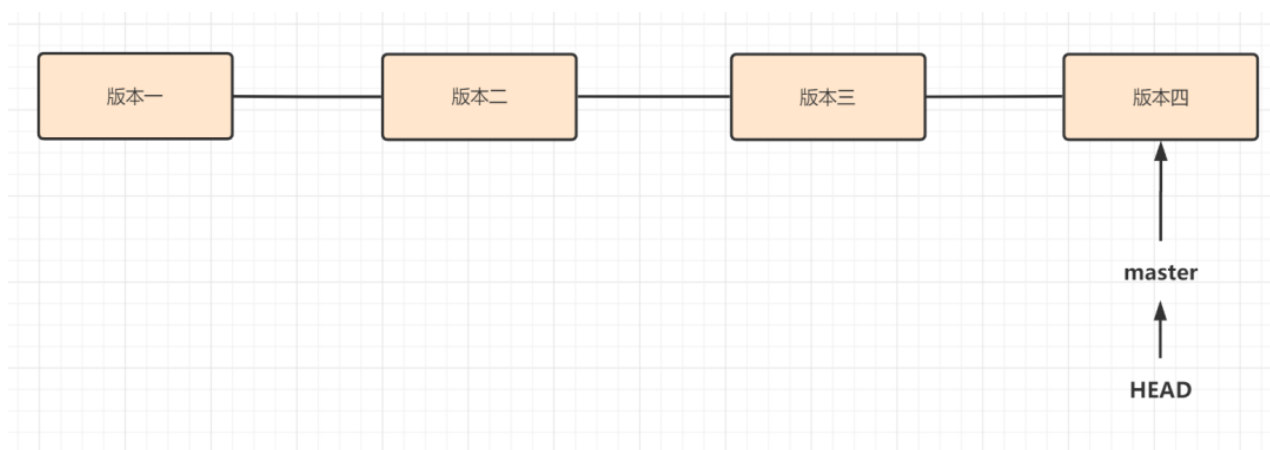
git revert

与git reset不同的是，revert复制了那个想要回退到的历史版本，将它加在当前分支的最前端。

revert之前：



revert 之后：



当然，如果代码已经推送到远程的话，还可以考虑revert回滚呢

git log 得到你需要回退一次提交的commit id
git revert -n <commit_id> 撤销指定的版本，撤销也会作为一次提交进行保存


```
weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnngit (master)
$ git log -n 3
commit b979cc5a0106bf5a8a25c29dfa2c3f84b5294d6 (HEAD -> master, origin/master, o
rigin/HEAD)
Author: whx123 <327658337@qq.com>
Date: Sat Jun 27 10:04:19 2020 +0800

    revert 代码回滚2

commit b576eac95a590db33cb665b24d646c6129fe3843
Author: whx123 <327658337@qq.com>
Date: Sat Jun 27 10:03:24 2020 +0800

    revert 代码回滚

commit 36b6081ba2ed3fd28c3f7dc1bcc6c67ae09ee4ec
Author: whx123 <327658337@qq.com>
Date: Sat Jun 27 09:59:34 2020 +0800

    结局冲突

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnngit (master)
$ git revert -n b576eac95a590db33cb665b24d646c6129fe3843
error: could not revert b576eac... revert 代码回滚
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnngit (master|REVERTING)
$ git add .

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnngit (master|REVERTING)
$ git commit -m '解决冲突'
[master 8dcf48b] 解决冲突
1 file changed, 3 insertions(+)

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnngit (master)
$ git push
fatal: HttpRequestException encountered.
-----
Username for 'https://github.com': whx123
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 345 bytes | 172.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/whx123/learnngit.git
   b979cc5..8dcf48b  master -> master
```

最近版本历史

回滚

有时候有冲突要
解决，OK再提交

搞完再推送

Git进阶之标签tag

打tag就是对发布的版本标注一个版本号，如果版本发布有问题，就把该版本拉取出来，修复bug，再合回去。

git tag 列出所有tag
git tag [tag] 新建一个tag在当前commit
git tag [tag] [commit] 新建一个tag在指定commit
git tag -d [tag] 删除本地tag
git push origin [tag] 推送tag到远程
git show [tag] 查看tag
git checkout -b [branch] [tag] 新建一个分支，指向某个tag

```

weihua@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnGit (master)
$ git tag
1.0.0
weihua@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnGit (master)
$ git tag 1.0.1
weihua@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnGit (master)
$ git tag
1.0.0
1.0.1
weihua@LAPTOP-QKRMA6P8 MINGW64 ~/test/learnGit (master)
$ git show 1.0.0
commit 8dcf48ba740e6e5b32f017e911fc105768b719bc (HEAD -> master, tag: 1.0.1, tag: 1.0.0, origin/master, origin/HEAD)
Author: whx123 <327658337@qq.com>
Date: Sat Jun 27 10:43:40 2020 +0800

    解决冲突

diff --git a/test.txt b/test.txt
index 403a448..3e5dc92 100644
--- a/test.txt
+++ b/test.txt
@@ -1,5 +1,8 @@
 666
 888
 我想测试代码回滚
+<<<<<< HEAD
+revert 代码回滚
+revert 代码回滚2
+=====
+>>>>>> parent of b576eac... revert 代码回滚

```

Git其他一些经典命令

git rebase

rebase又称为衍合，是合并的另外一种选择。

假设有两个分支master和test

```
D --- E test / A --- B --- C --- F --- master
```

执行 git merge test得到的结果

```
D ----- E / \ A --- B --- C --- F --- G --- test, master
```

执行git rebase test，得到的结果

```
A --- B --- D --- E --- C' --- F' --- test, master
```

rebase好处是： 获得更优雅的提交树，可以线性的看到每一次提交，并且没有增加提交节点。所以很多时候，看到有些伙伴都是这个命令拉代码：git pull --rebase，就是因为想更优雅，哈哈

git stash

stash命令可用于临时保存和恢复修改

git stash 把当前的工作隐藏起来 等以后恢复现场后继续工作git stash list 显示保存的工作进度列表git stash pop stash@{num} 恢复工作进度到工作区git stash show : 显示做了哪些改动git stash drop stash@{num} : 删除一条保存的工作进度git stash clear 删除所有缓存的stash。

```
wetihuaixiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   test.txt

no changes added to commit (use "git add" and/or "git commit -a")

wetihuaixiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git stash save '保存一下'
Saved working directory and index state On master: '保存一下'

wetihuaixiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git stash list
stash@{0}: On master: '保存一下'

wetihuaixiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git stash show
test.txt | 7 +-----
1 file changed, 1 insertion(+), 6 deletions(-)

wetihuaixiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

wetihuaixiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git stash clear
```

git reflog

显示当前分支的最近几次提交

```

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git reflog
8dcf48b (HEAD -> master, tag: 1.0.1, tag: 1.0.0, origin/master, origin/HEAD) HEAD@{0}: commit: 冲突
b979cc5 HEAD@{1}: reset: moving to b979cc5a0106bf3dbae25c29dfa2c3f84b5294d6
b979cc5 HEAD@{2}: commit: revert 代码回滚2
b576eac HEAD@{3}: commit: revert 代码回滚
36b6081 HEAD@{4}: commit: 结局冲突
df25ba7 HEAD@{5}: reset: moving to df25ba7b06a36bbb67663f04982476139d85c896
df25ba7 HEAD@{6}: commit: 我还想测试revert代码回滚
2207c92 HEAD@{7}: commit: 我想测试代码回滚
30df714 HEAD@{8}: revert: Revert "修改"
5ad3692 HEAD@{9}: commit: 测试
397cdbc HEAD@{10}: reset: moving to 397cdbc6bf1af676f9acb45883241fc5c9d01a8a
44cc4c6 HEAD@{11}: commit: 又修改了test.txt文件
397cdbc HEAD@{12}: commit: 修改test文件
8901fd8 HEAD@{13}: commit (merge): 解决冲突
0b92600 HEAD@{14}: commit: master modified
15fb516 HEAD@{15}: pull origin master: Fast-forward
05a16fb HEAD@{16}: checkout: moving from dev to master
6e6d760 (dev) HEAD@{17}: checkout: moving from master to dev
05a16fb HEAD@{18}: checkout: moving from dev to master
6e6d760 (dev) HEAD@{19}: commit: ev modiffied
e5801a1 HEAD@{20}: commit: 修改
05a16fb HEAD@{21}: checkout: moving from master to dev
05a16fb HEAD@{22}: merge dev: Fast-forward
5ed5b0e HEAD@{23}: checkout: moving from dev2 to master
5ed5b0e HEAD@{24}: checkout: moving from master to dev2
5ed5b0e HEAD@{25}: checkout: moving from dev2 to master
05a16fb HEAD@{26}: checkout: moving from dev to dev2
05a16fb HEAD@{27}: commit: 删除bak文件
d9a4740 (origin/dev) HEAD@{28}: checkout: moving from master to dev

```

git blame filepath

git blame 记录了某个文件的更改历史和更改人，可以查看背锅人，哈哈

```

weihuaxiao@LAPTOP-QKRMA6P8 MINGW64 ~/test/learngit (master)
$ git blame test.txt
397cdbc6 (whx123 2020-06-27 08:49:35 +0800 1) 666
2207c92d (whx123 2020-06-27 09:41:38 +0800 2) 888
36b6081b (whx123 2020-06-27 09:59:34 +0800 3) 我想测试代码回滚
8dcf48ba (whx123 2020-06-27 10:43:40 +0800 4) <<<<<< HEAD
b576eac9 (whx123 2020-06-27 10:03:24 +0800 5) revert 代码回滚
b979cc5a (whx123 2020-06-27 10:04:19 +0800 6) revert 代码回滚2
8dcf48ba (whx123 2020-06-27 10:43:40 +0800 7) =====
8dcf48ba (whx123 2020-06-27 10:43:40 +0800 8) >>>>>> parent of b576eac... revert 代码回滚

```

git remote

git remote 查看关联的远程仓库的名称
git remote add url 添加一个远程仓库
git remote show [remote] 显示某个远程仓库的信息

参考与感谢

感谢各位前辈的文章：

- 一个小时学会Git
(https://www.cnblogs.com/best/p/7474442.html#_label3_4_o_4)
- 【Git】(1)---工作区、暂存区、版本库、远程仓库
(<https://www.cnblogs.com/qdhxhz/p/9757390.html>)
- Git Reset 三种模式 (<https://www.jianshu.com/p/c2ec5f06cf1a>)

- Git恢复之前版本的两种方法reset、revert (图文详解)
(<https://blog.csdn.net/yxlshk/article/details/79944535>)
- Git撤销&回滚操作(git reset 和 get revert)
(<https://blog.csdn.net/asoar/article/details/84111841>)
- 为什么要使用git pull --rebase? (<https://www.jianshu.com/p/dc367c8dca8e>)

后记

工作需要，调研了桌面端开发常用的一些技术栈，分享出来，供需要的小伙伴参考。

JavaFX则是将界面和逻辑都分开处理了，就像Android开发那样。

跨平台桌面应用程序：JavaFX 与 Electron之间的选择？

JavaFX与JRE捆绑在一起，因此“轻量级”是有争议的。并且，JavaFX 目前使用的人太少了，你几乎在身边找不到使用 JavaFX 做过项目的小伙伴！并且，Java 本就不适合桌面应用程序开发。

我更推荐使用使用 Electron。

桌面端开发技术栈大揭秘

Microsoft阵营

1. **Winform**：大多数人开发CS程序都是基于Winform去做的，它的有点在于简单、高效，但是它的缺点在于，如果你想深入的美化UI，需要耗费很大的力气
2. **WPF**：基于XML+C#+CSS，相比于Winform，UI 设计上更加灵活。
3. **UWP**：微软为了针对移动端市场开放的开发框架，不过目前基本已经属于淘汰的状态。

Java阵营

1. **Swing**：默认样式贼丑,不过有一些ui开源库可以使用，代码和样式逻辑冗余在一起。不推荐使用！
2. **JavaFx**：相比于 Swing，JavaFx则是将界面和逻辑都分开处理了，就像Android开发那样。但是，这玩意目前也没啥人用。

其他

1. **QT**：有非常多的跨平台Desktop Application是基于QT编写的，它不仅能够保证跨平台，而且能够将运行效率最大化。QT另外有一个优势在于，它在UI上似乎要比之前几位要方便一些，在它的QML中甚至可以直接使用JavaScript（当然，Java也内置了JS引擎），同时QT中也包含了大量的标准CSS样式表可以使用。
2. **Electron**：相比于 QT 执行效率低很多。Electron 是目前开发桌面应用成本最低的办法，但是最大的硬伤就是体积和资源占用

综上，跨平台 Desktop Application的开发，我只推荐两种：**Electron**和**QT**

但是，如果你的程序只是跑在Windows上，就不用考虑了，WPF是你最好的选择。如果你的程序只跑在 Mac 上，可以基于Swift来做

JavaGuide

我整理的4本Java原创PDF文档，公众号“后端技术进阶”后台回复“面试突击”即可免费获取。

文章有帮助可以点个「在看」或「分享」，都是支持，我都喜欢！

我是Guide哥，Java后端开发，会一点前端知识，喜欢烹饪，自由的少年。一个三观比主角还正的技术人。我们下期再见！



JavaGuide