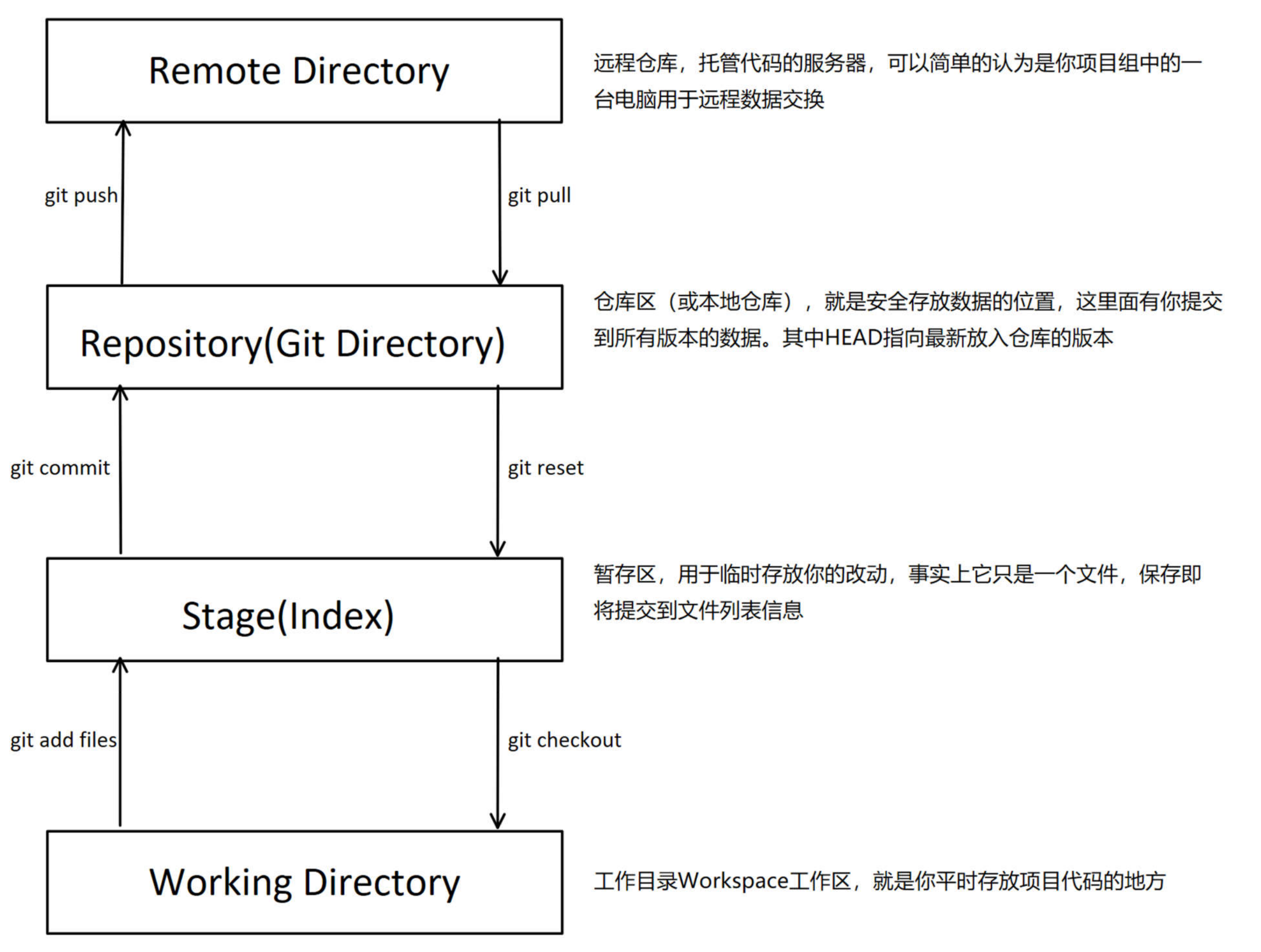
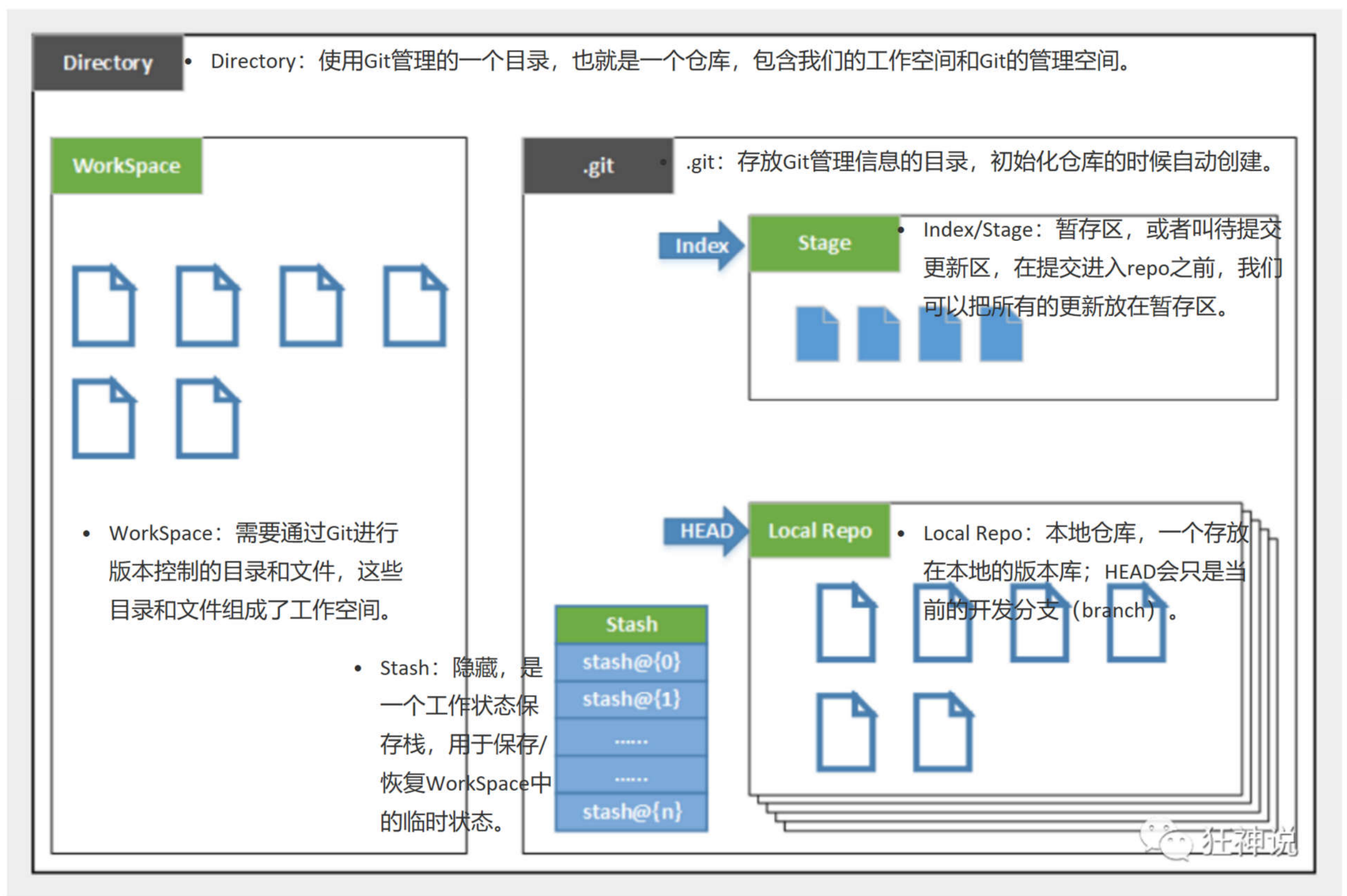
Git 原理

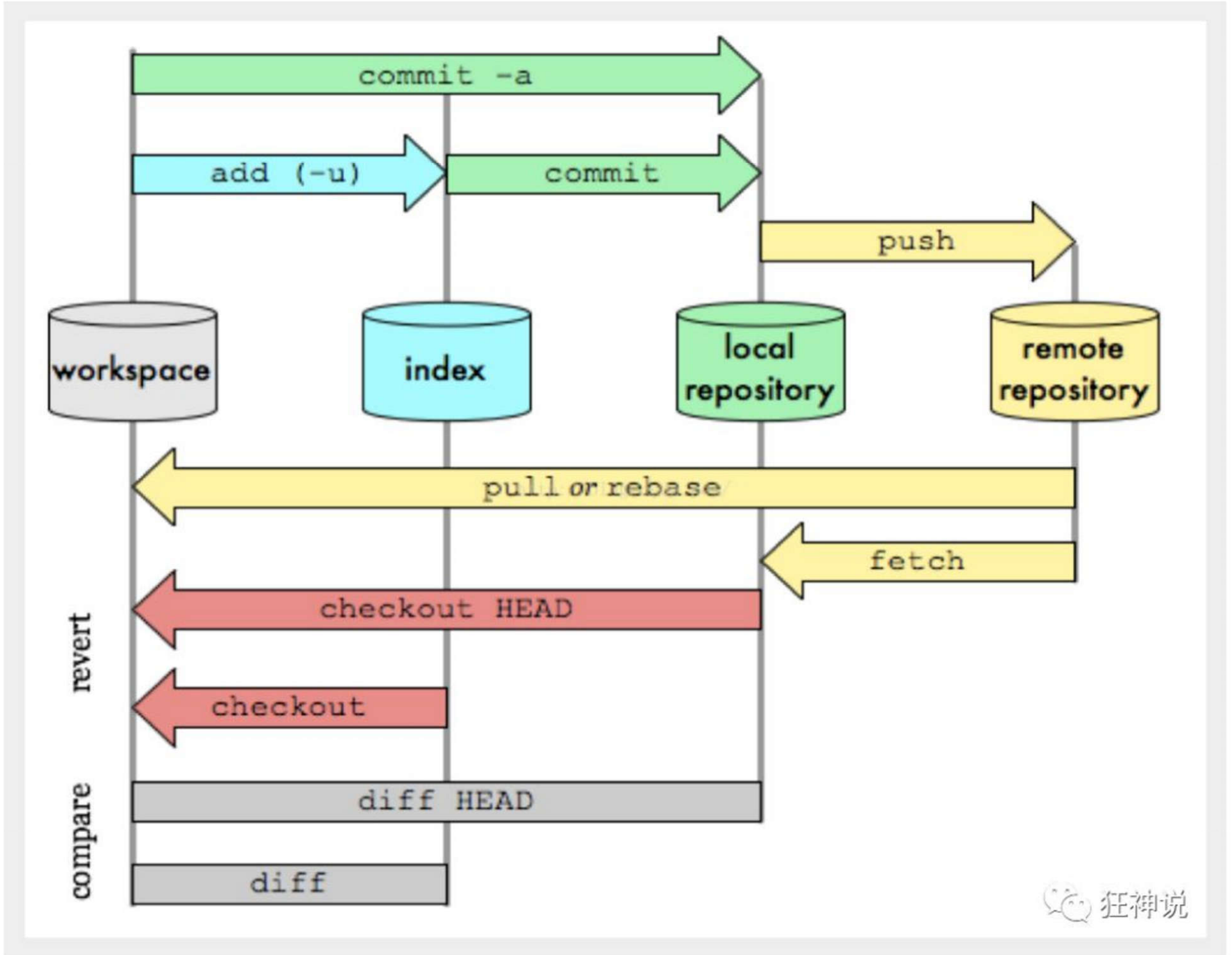
2020年11月13日
19:14



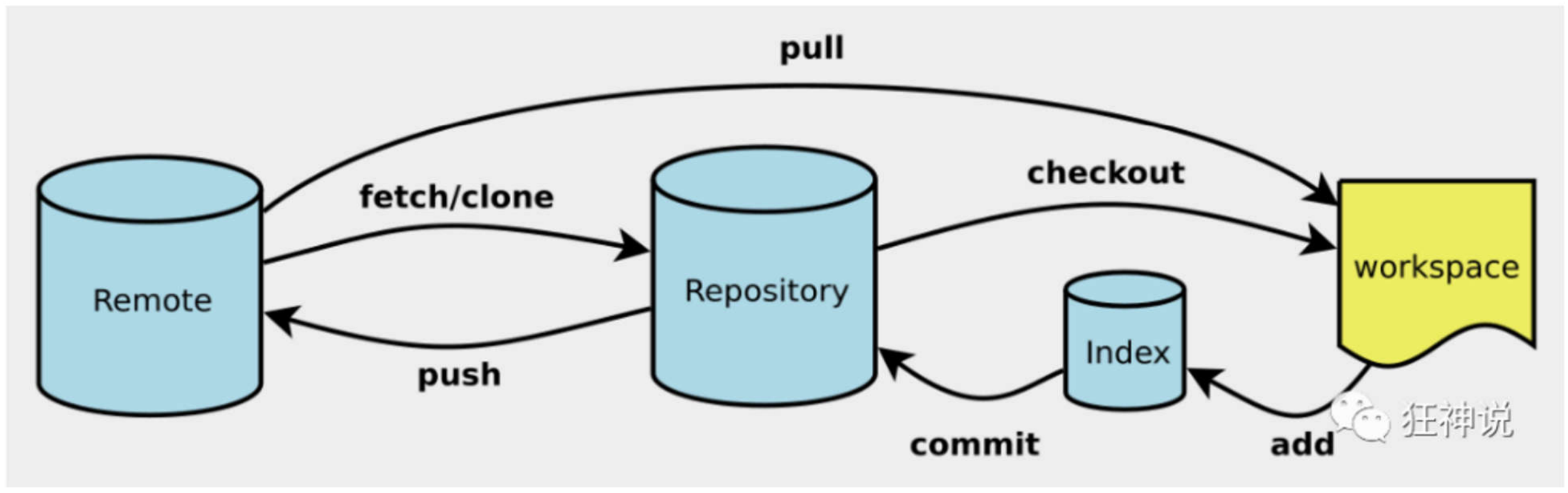
本地的三个区域确切的说应该是git仓库中HEAD指向的版本：

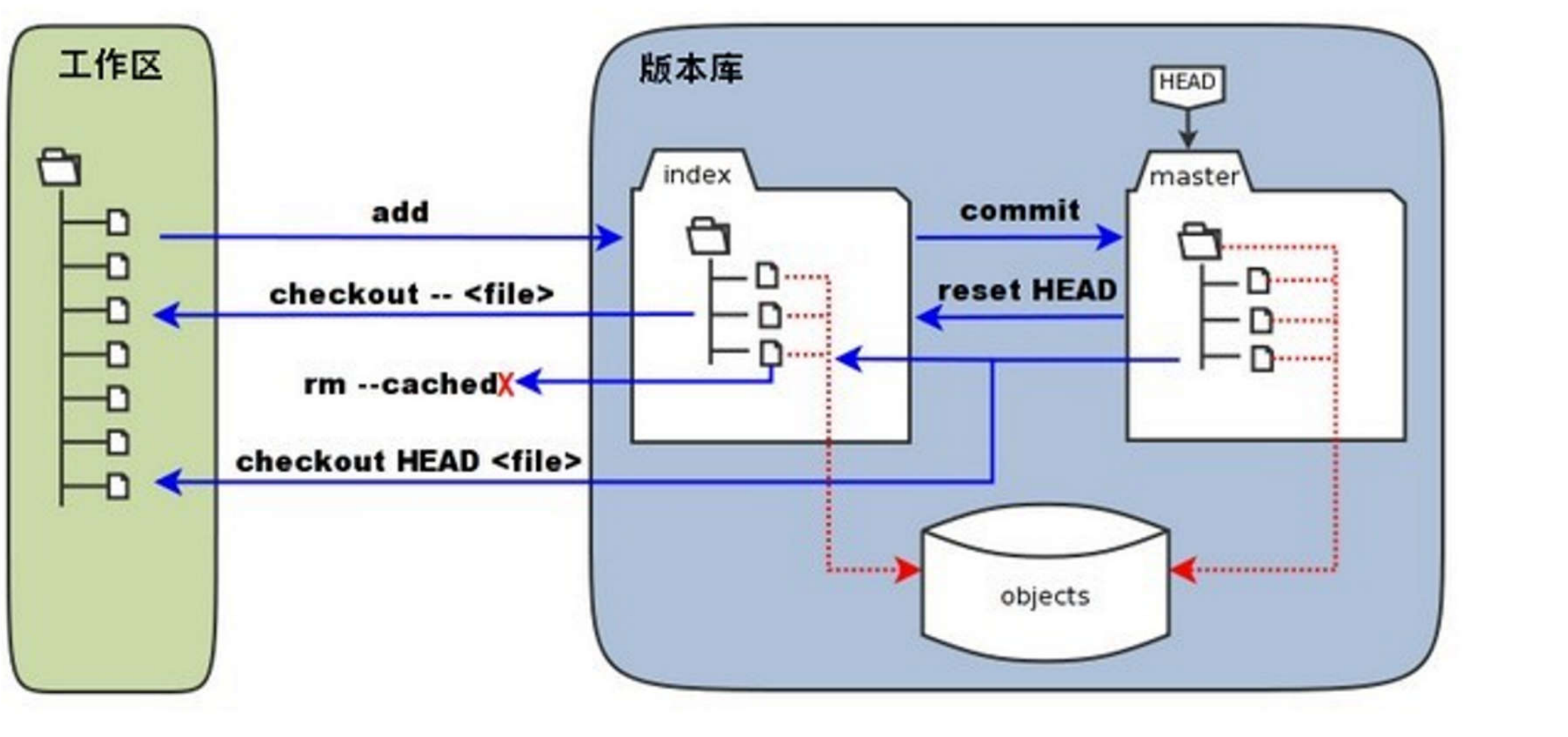


工作流程



常用指令





本地仓库搭建

创建本地仓库的方法有两种：一种是创建全新的仓库，另一种是克隆远程仓库。

1、创建全新的仓库，需要用GIT管理的项目的根目录执行：

```
# 在当前目录新建一个Git代码库
$ git init
```

2、执行后可以看到，仅仅在项目目录多出了一个.git目录，关于版本等的所有信息都在这个目录里面。

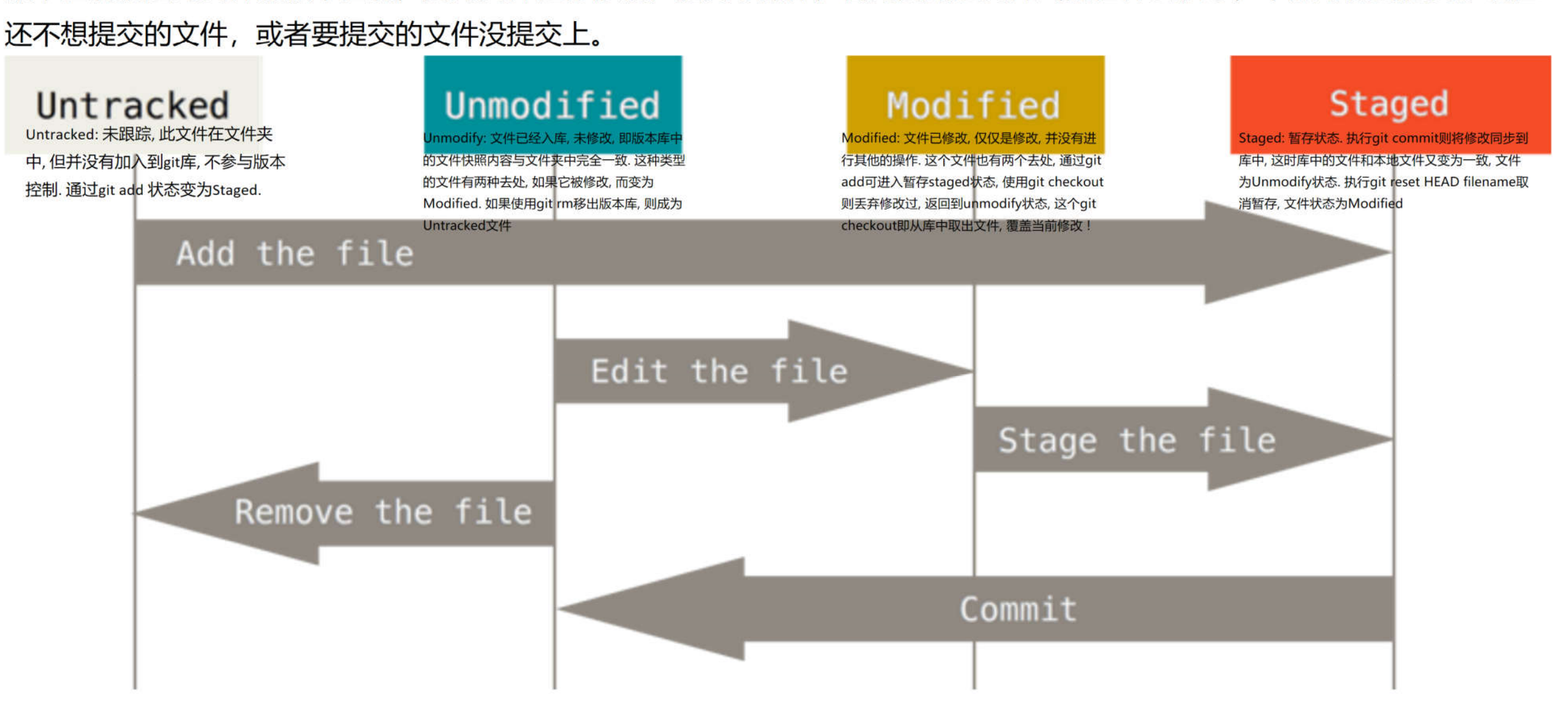
克隆远程仓库

1、另一种方式是克隆远程目录，由于是将远程服务器上的仓库完全镜像一份至本地！

```
# 克隆一个项目和它的整个代码历史(版本信息)
$ git clone [url]  # https://gitee.com/kuangstudy/openclass.git
```

文件的四种状态

版本控制就是对文件的版本控制，要对文件进行修改、提交等操作，首先要知道文件当前在什么状态，不然可能会提交了现在还不想提交的文件，或者要提交的文件没提交上。



git merge 把另一分支的更改在master分支下进行重演并提交一个新的commit

```
      A---B---C topic
     /
D---E---F---G master
```

git merge topic

```
      A---B---C topic
     /         \
D---E---F---G---H master
```

作用：用于合作开发，可以分辨时间顺序

优点：比较容易理解

缺点：作者修改历史可能会乱

适合的合作模式：大部分情况 / 合作者相互熟悉并且沟通频繁

Warning: 不建议在未commit的已更改状态下用git merge

git rebase 把其他分支一个个整个转移到master分支上面整合成一条分支

```
      A---B---C topic
     /
D---E---F---G master
```

git rebase master
git rebase master topic

```
              A'--B'--C' topic
             /
D---E---F---G master
```

如果upstream已经做了和你同样的修改（A' and A introduce the same set of changes, but have different committer information)

```
      A---B---C topic
     /
D---E---A'---F master
```

git rebase master
git rebase master topic

```
               B'---C' topic
              /
D---E---A'---F master
```

git rebase --onto

```
o---o---o---o---o  master
     \
      o---o---o---o---o  next
                       \
                        o---o---o  topic
```

git rebase --onto master next topic

```
o---o---o---o---o  master
    |            \
    |             o'--o'--o'  topic
     \
      o---o---o---o---o  next
```

```
                         H---I---J topicB
                        /
                E---F---G  topicA
               /
A---B---C---D  master
```

git rebase --onto master topicA topicB

```
             H'--I'--J'  topicB
            /
            | E---F---G  topicA
            |/
A---B---C---D  master
```

作用：通常用来解决一些merge不了的分支conflict

优点：可以按作者清晰区分commit

缺点：时间顺序会被打乱。理解起来较为复杂

适合的合作模式： 大企业多个团队， 相互交流较少，各种集中开发

git revert:

- **Rollback changes** you have committed.
- Creates a **new commit** from a specified commit by inverting it. Hence, adds a new commit history to the project, but it does not modify the existing one.
- Has the potential to overwrite files in the working directory.

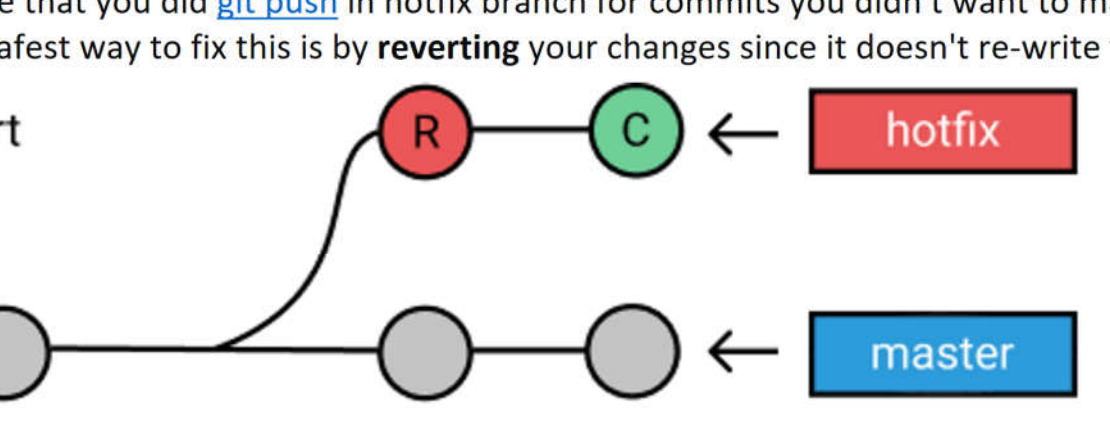
git reset:

- Use this to **return** the *entire* working tree to the last committed state. *This will discard commits in a private branch or throw away uncommitted changes!*
- Changes which commit a branch HEAD is currently pointing at. It alters the existing commit history.
- Can be used to **unstage** a file.

Undo Public Changes

Scenario: Image that you did git push in hotfix branch for commits you didn't want to make yet.
Solution: The safest way to fix this is by **reverting** your changes since it doesn't re-write the commit history.

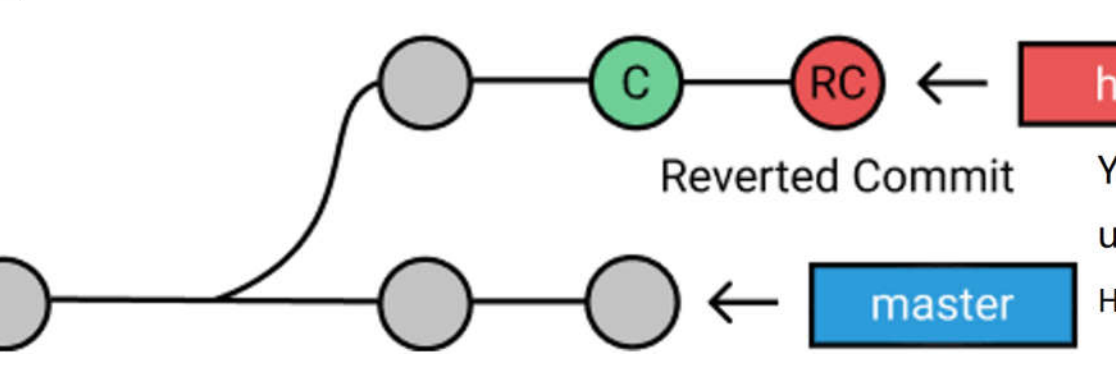
Before Revert



Result: You have successfully undone committed changes! Everything that was changed in the old commit will be reverted with this new commit. *Git forces you to commit or stash any changes in the working directory that will be lost during the checkout.*

```
git checkout hotfix
git revert HEAD~1
```

After Revert



You can think of git revert as a tool for undoing **committed** changes, while git reset HEAD is for undoing **uncommitted** changes.

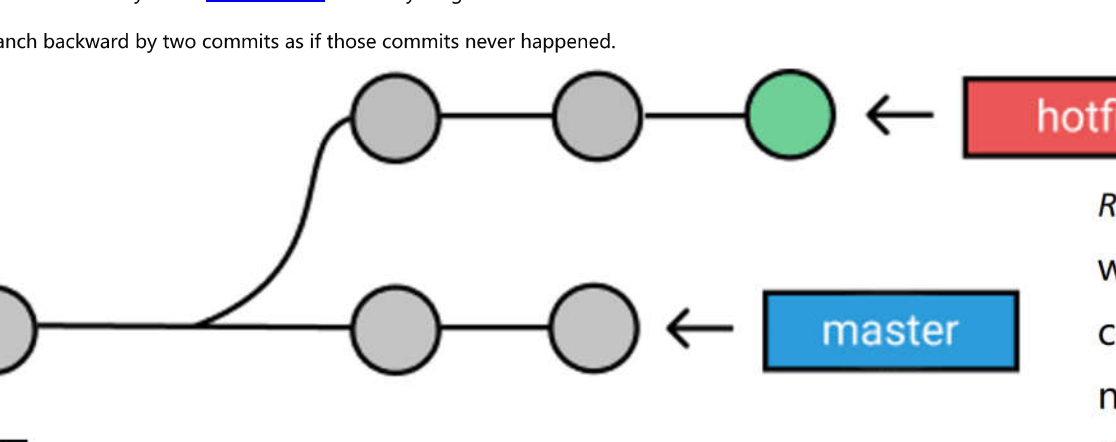
Undo Local Changes

Scenario: You started working on a feature, but you didn't like the end result. These changes haven't been **shared* with anyone else.*
Solution: You want to **undo everything** in that files to the previous state, just the way it looked in the last commit.
git checkout file_name.rb
Result: File file_name.rb has been **reverted** to a state previously known to Git. Note that this removes all of the subsequent changes to the file
You can use git checkout branch_name to switch between branches. Git forces you to commit or stash any changes in the working directory that will be lost during the checkout operation.

Undo Private Changes

Scenario: You've made some commits locally in the hotfix branch but everything is terrible! You want to remove the last two commits from the current branch.
Solution: **Reset** the hotfix branch backward by two commits as if those commits never happened.

Before Reset



Result: Your git repository has been **rewinded** all the way back to the specified commit. Those left out commits are now **orphaned** and will be removed the next time Git performs a **garbage collection**. For now, then their contents are still on disk.

```
git checkout hotfix
git reset HEAD~2
```

After Reset