分布式版本控制系统—Git使用介绍

一、版本控制系统介绍

1.1什么是版本控制系统

我们为什么需要版本控制系统

举个例子，公司要开发一个项目。

> 开发人数有10个人，这10个人同时对这个项目进行开发。
>
> 每个人开发的任务不同，修改的地方也不同。
>
> 这个时候是不是需要一个人来汇总大家所有开发的代码到一个地方？
>
> 他需要每天去**汇总**大家的开发的代码到一起，进行**记录**，**备份**，这样才是一个完整的项目。而且很有可能两个人同时修改了一个文件的代码，那么这个人需要去对比，两个人的代码并进行**合并**等等。
>
> 汇总完成后还要分发给大家，保证大家是基于最新的项目代码进行开发的。

所以**版本控制系统**出现了，它做的工作就是和这个人一样的。

1.2 版本控制系统的基本功能

版本控制的基本功能

1. 记录整个项目的开发过程
2. 汇总每个人的工作成果
3. 可以回溯/查看历史版本
4. 协同开发

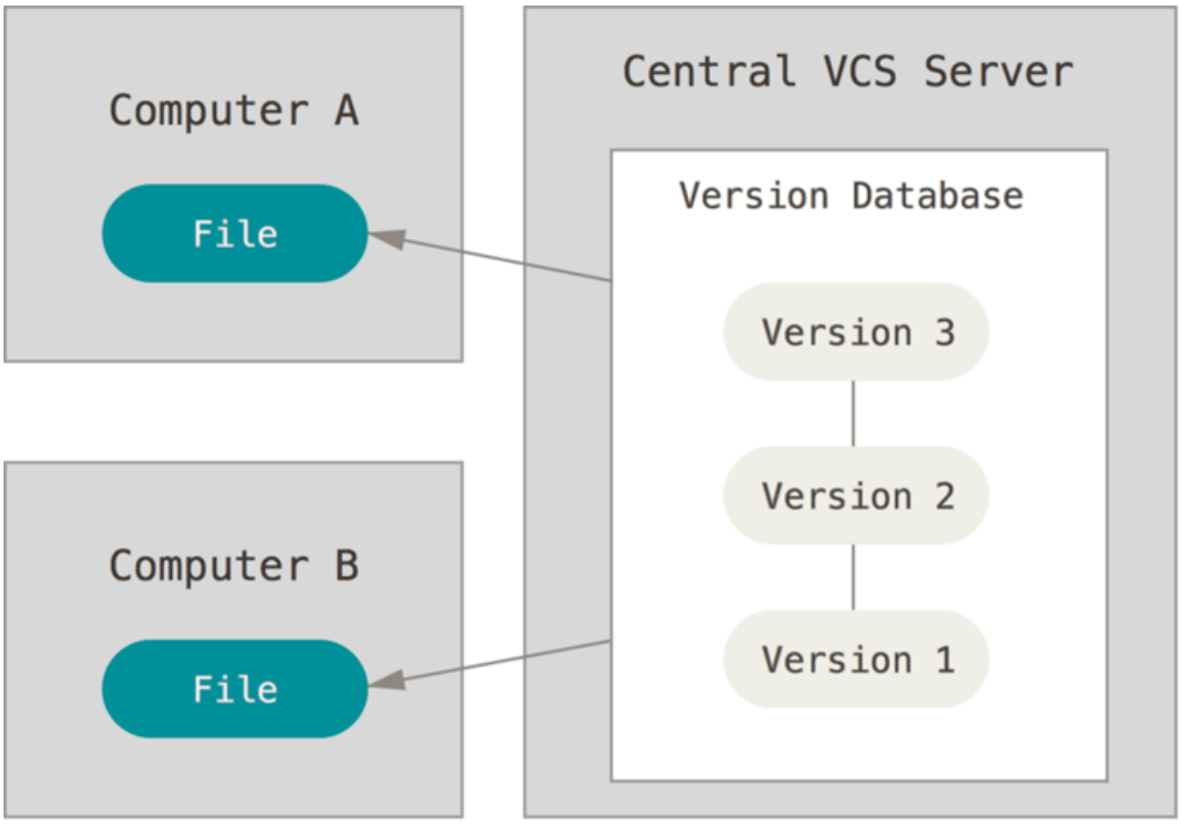
1.3 版本控制系统分类

1. 本地式版本控制
2. 集中式版本控制
3. 分布式版本控制

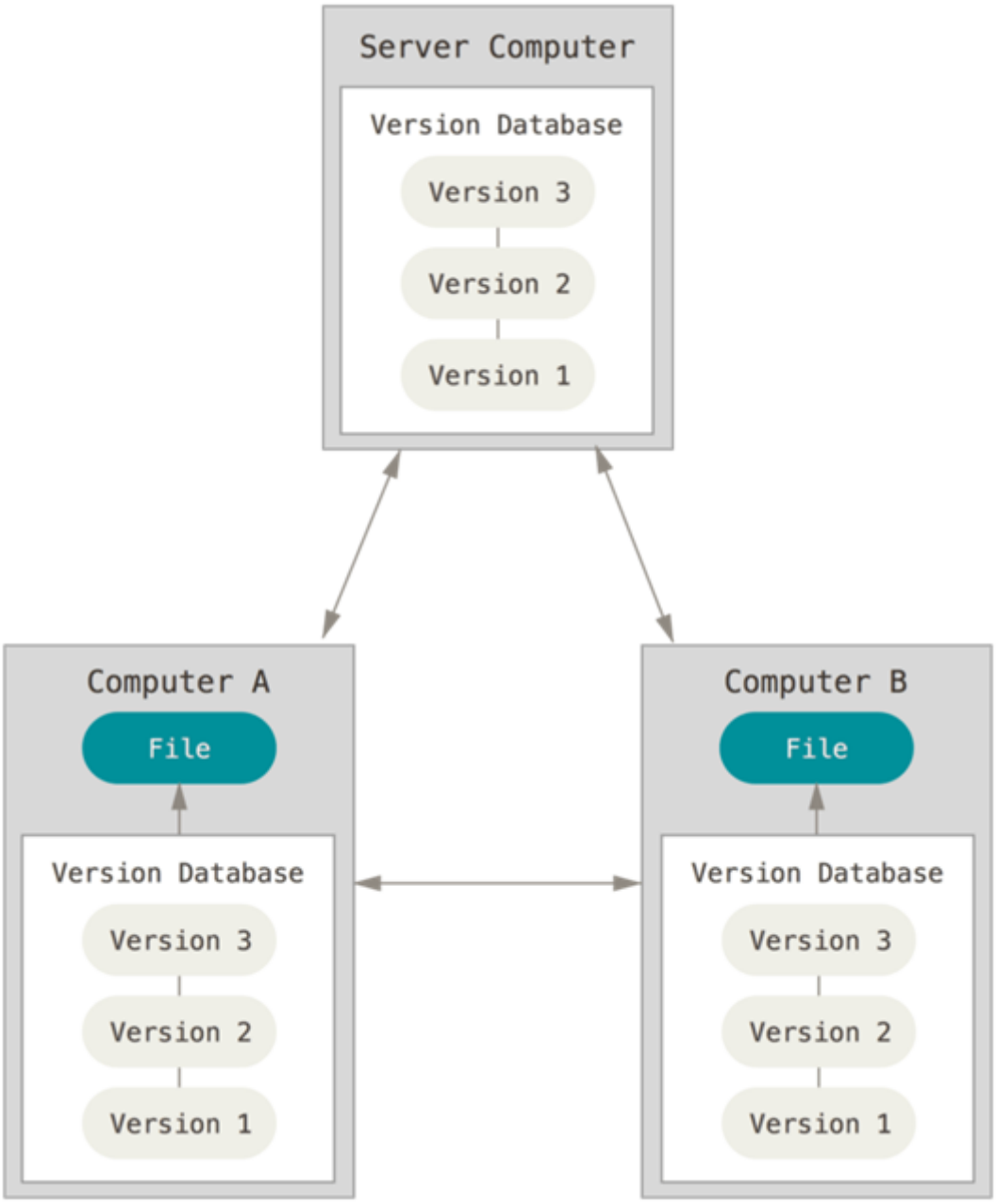
本地式版本控制

本地式版本控制就相当于我们在自己电脑上将某些文件的版本记录了多份，不多介绍。

集中式版本控制



分布式版本控制



分布的意思其实就是不在一个集中的地点去最终数据

Git就是分布式版本控制系统的代表，

分布式版本控制系统每次获取**不仅仅是获取最新的文件快照**，而是获取**整个版本库获取到了本地**，也就是说每个人都拥有一个完整的版本库，查看日志、提交、创建里程碑和分支、合并分支、回退等所有操作都直接在本地完成而不需要网络。

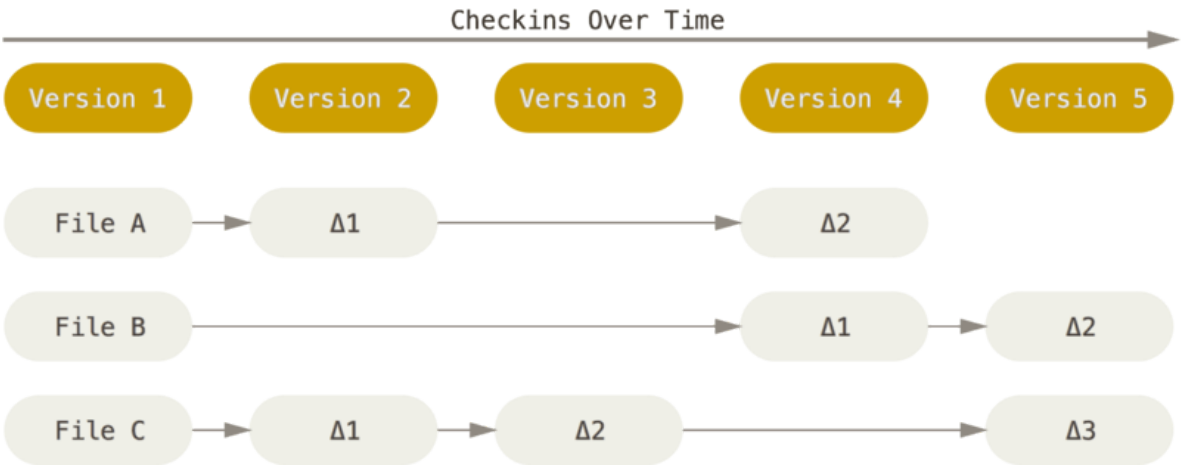
这样做的好处避免了单点故障问题，性能快，因为大部分情况下都不需要网络。

二、Git简介

Git 是一个**分布式版本控制系统**，Gti版本库是基于快照的存储方式的来记录文件的历史版本。

Git与SVN的区别

SVN是基于**差异存储数据**

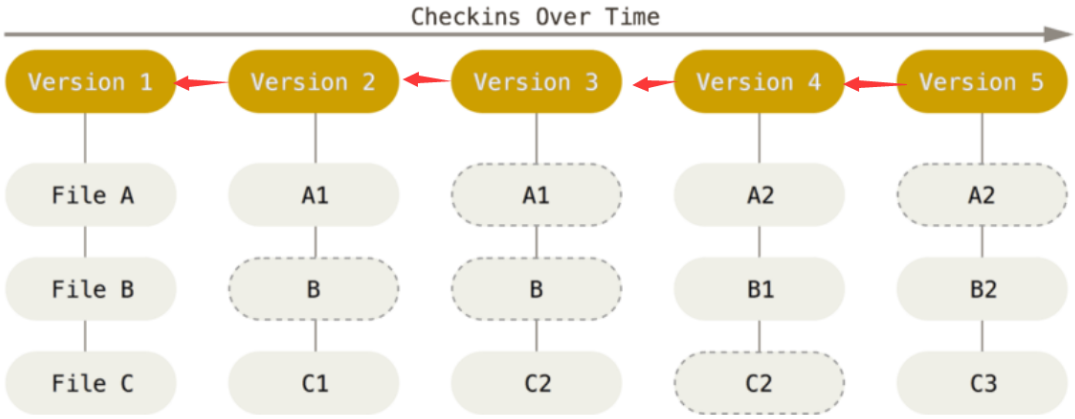


2.1 基于快照的版本控制

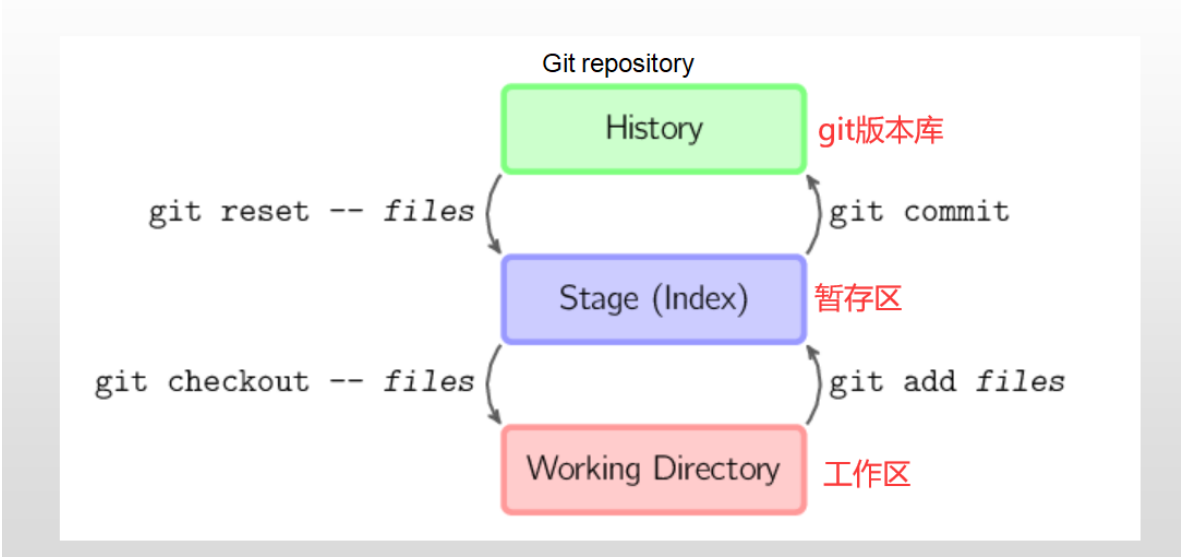
快照是Git很重要的一个概念，理解了快照你就很容易掌握Git。

下图展示了什么是**基于快照的版本控制**

在 Git 中，每当你**提交更新**时，它基本上就会对当时的**全部文件**创建一个**快照**并保存这个快照的索引。为了效率，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。



2.2 Git仓库的三个区域



关系。

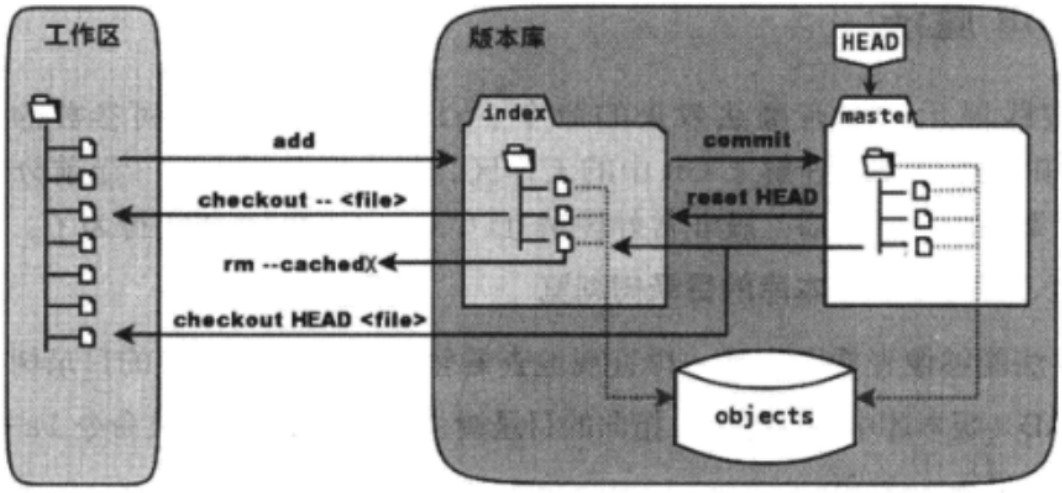
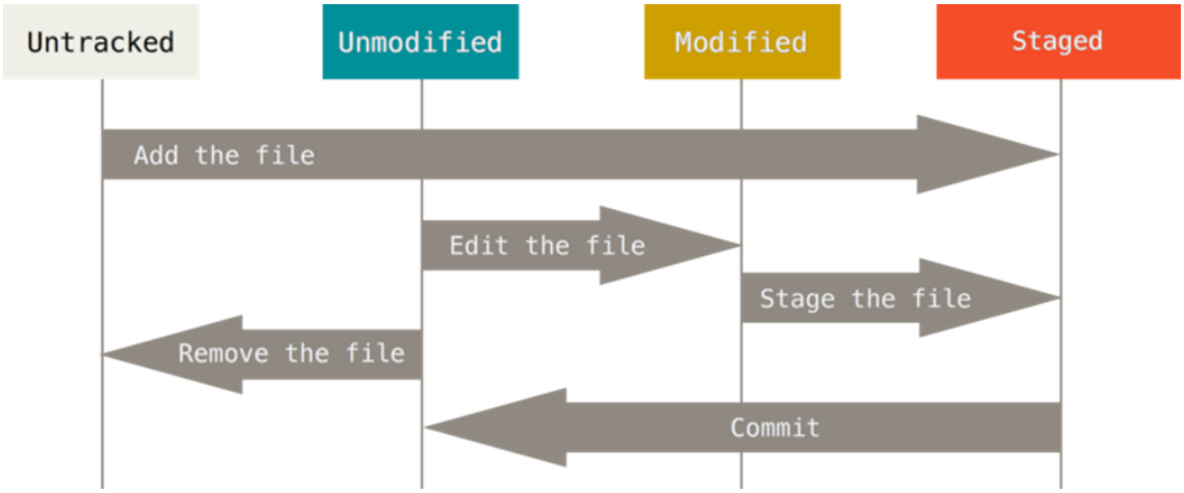


图 5-1　工作区、版本库、暂存区原理图

2.3 Git版本库文件的三个状态

Git仓库目录中的文件有两种：tracked（git认识的）、untracked（git不认识的）

纳入Git版本库的文件有三种状态：unmodified（**未修改**）、modified（**已修改**）、staged（**已暂存**）。



三、Git基本使用

3.1 获取一个本地的Git仓库

```
#从远程仓库克隆到本机
git clone <远程仓库url> [目录]
#本机初始化一个的Git仓库
git init [目录]
```

3.2 查看文件状态

```
#显示工作区文件状态
git status
git status -s 简要显示
```

会显示工作区和暂存区文件的状态。

如果工作区、暂存区、Git仓库HEAD指向的提交commit中的快照一样。

那么会显示nothing to commit,working tree clean.

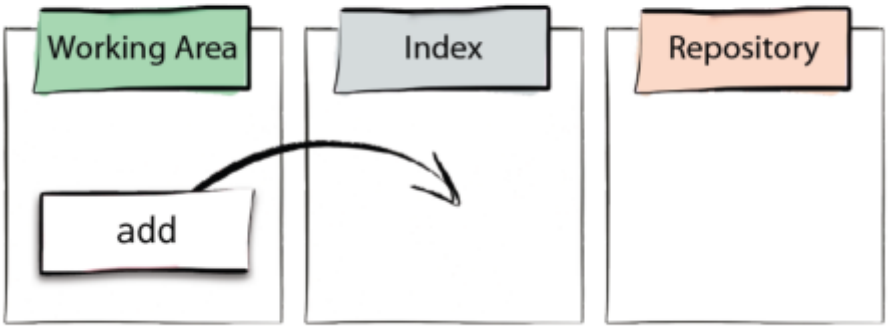
3.3 向git版本库添加文件

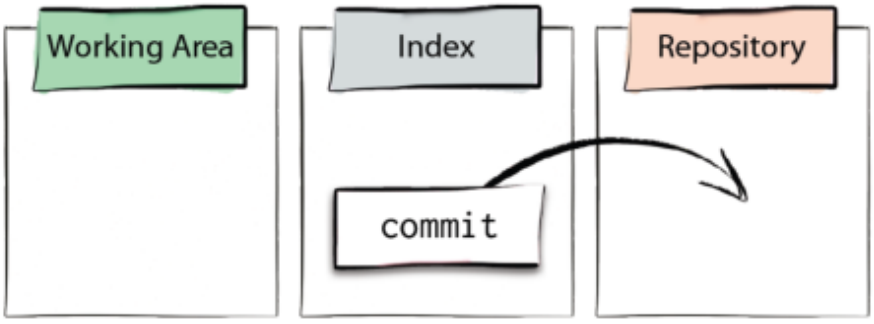
3.3.1 将工作区文件添加到暂存区

添加文件到Git仓库分为两步：

1. 首先将要添加进Git仓库的文件添加到暂存区。
2. 将暂存区的文件快照提交到Git仓库。

添加的是当前工作区文件的一个**快照**，此刻git会生成一个索引指向该快照






```
#git add命令将当前文件快照添加到暂存区
#用法
 git add <file>…  添加指定文件到暂存区
 git add -u 添加已修改或删除的文件到暂存区
 git add . 添加所有文件到暂存区
```

3.3.2 提交到Git版本库

```
#语法
#git commit      -- 会创建一个commit对象，该对象指向了了工作区的完整快照
git commit -m “提交说明”
git commit --amend 修改最近一次提交
```

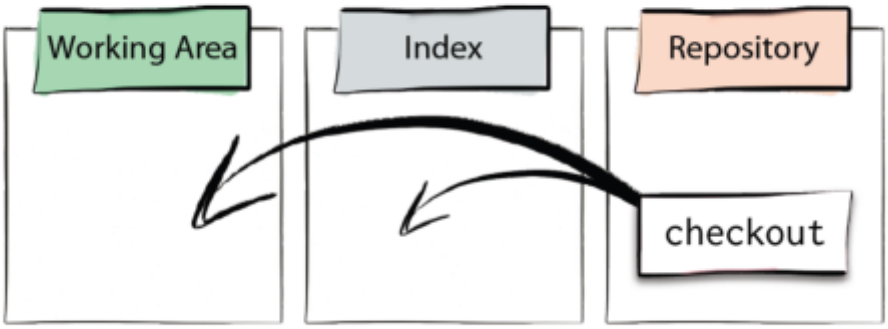
3.4查看提交日志

```
git log [branch] --oneline 显示提交日志
# branch 是指要显示的分支名，不写则以显示当前所在分支的日志
# --oneline 是将每个提交信息缩短为一行
```

3.5 恢复工作区的修改

git checkout 会重写工作区与暂存区。

最常用的用法就是**覆盖工作区**。




```
#将暂存区的文件快照覆盖到工作区的指定文件，相当于还原在工作区的修改
#用法 git checkout [commit] [--] <file>…
#如果commit省略，则将暂存区的文件覆盖到工作区
git checkout -- <file>...
#commit不省略，将以该提交的快照文件覆盖工作区和暂存区。
git checkout HEAD -- <file>..
```

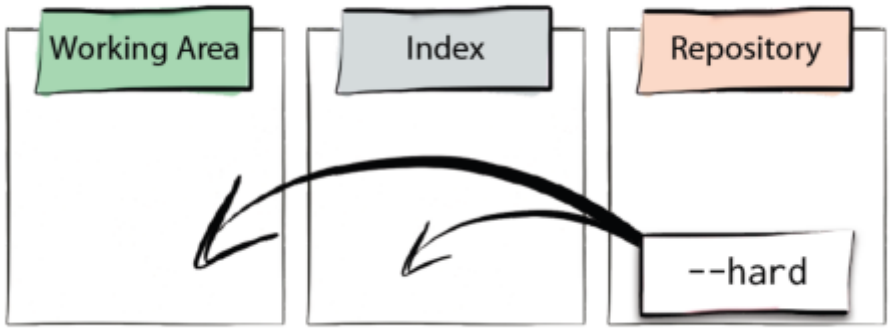
3.6 重置回退

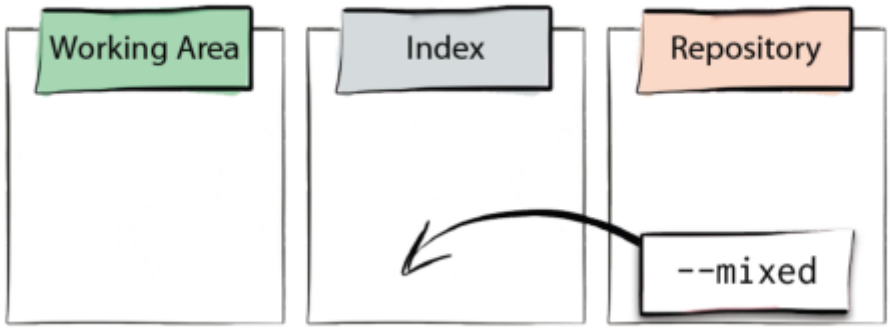
通常用来恢复暂存区的文件。

```
#git reset 有两种用法
#用法1 恢复暂存区文件 ，相当于git add 的反向操作，
#git reset [<commit>] [--] <paths>..  #如果省略commit，则以当前HEAD指针指向的提交进行还原
git reset -- file.txt  #将Git仓库file.txt快照覆盖到暂存区
git reset HEAD^ file.txt #将Git仓库上一个版本的file.txt覆盖到暂存区

#用法2 回退到某一版本，通过参数来选择覆盖哪些区域
#git reset [--soft|--mixed|--hard]  [<commit>]
git reset --hard HRAD^^  #回退两个版本 并将该版本的快照覆盖工作区和暂存区
git reset --hard HEAD # 将工作区和暂存区回退到Git仓库的最新版本，相当于撤销工作区和暂存区的所有修改
```

重置命令图解





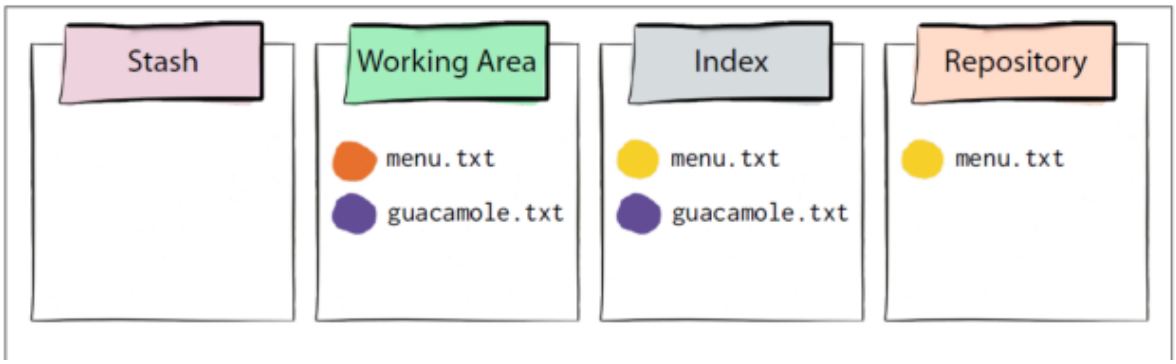
3.7 删除文件

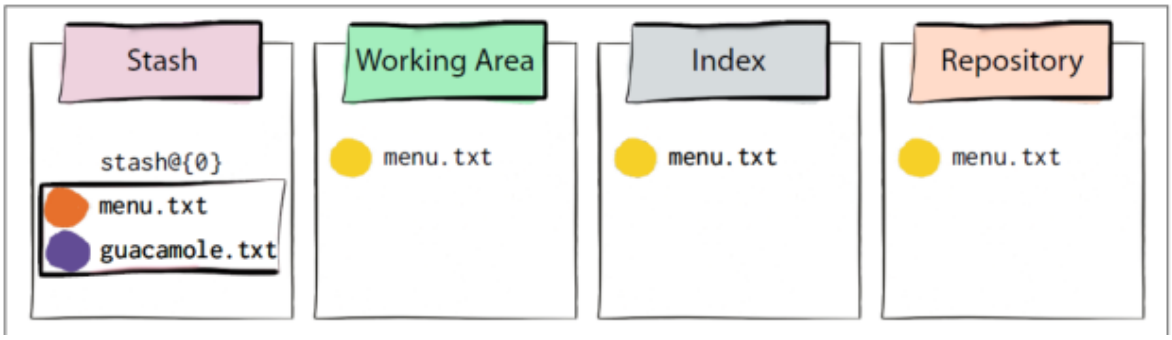
```
#工作区和暂存区都会被删除
git rm <file>…
#只删除暂存区不删除工作区
git rm --cached <file>...
```

3.8 保存/恢复工作进度

stash是Git的**第四个区域**，用于保留工作区的进度。

stash区域专门用来存储临时保存的快照，是只有在本地仓库才会有的一个区域，不会记录到Git仓库中。






```
#git stash 命令，可以记录当前工作区文件及暂存区的文件快照到stash区域
git stash #保存当前工作区和暂存区的工作进度到stash区域
git stash save "保存当前的工作进度说明"   #保存工作区和暂存区的工作进度到stash区域，附说明,方便查找

git stash pop #恢复最佳一次保存的进度，恢复后会删除该进度
git stash list #查看所有保存在stash区域的列表
git stash apply #恢复最佳一次报错的进度，不会删除该进度
git stash pop
```

四、远程仓库

4.1 将远程仓库克隆到本地

```
#克隆远程仓库到本地
git clone <url> 会自动绑定远程仓库
#已有本地仓库，添加绑定远程仓库
git remote add origin <url>
#查看远程仓库的信息
git remote -v
```

4.2 从远程仓库获取最新的文件

```
#git pull 远程仓库名 分支  ，获取远程仓库上的分支到本地分支
#如
git pull origin master  # 获取远程仓库的master分支到本地分支

git fetch 远程仓库名 分支  ，获取远程仓库的分支到本地远程分支
如 git fetch origin master
# git pull 命令相当于git fetch 命令和git merge命令一起执行
#git pull origin master = git fetch origin master + git merge origin/master
```

4.3 将本地分支推送到远程分支

```
#语法 git push -u 远程仓库名 分支名  ，首次推送 -u参数表示创建和本地分支的关联关系
#如 git push origin master
```

快进式推送fast-forward

就是要推送的本地版本库的提交是简历在远程版本库相应分值的现有提交基础上的，即远程版本库响应分值的最新提交是本地版本库最新提交的祖先提交。

```
git rev-list HEAD #查看本地库的最新提交及其历史提交

git ls-remote origin 分支名 #显示远程版本库的引用对应的SHA1哈希值
```

实际上是推送至远程仓库的时候，Git就是利用类似方法判断出当前的推送不是一个快进式的推送，于是产生警告并终止。

如果非快进式推送，如何才能推送成功呢?

1、强制推送

2、合并并推送

五、Git分支

常用的场景

release branch 发布分支

feature branch 特性分支

5.1 分支 git branch

```
```

```
#显示本地分支列表，以及当前所在分支
 git branch
#显示远程分支列表
 git branch -r
#新建一个分支
 git branch 分支名
#新建一个分支并切换到该分支上，可以用该命令建立一个本地分支以跟踪远程分支
 git checkout -b 分支名
#切换分支
 git checkout 分支名
#删除分支，在删除分支<branchname>时会检查索要删除的分支是否已经合并到其他分支中，否则拒绝删除。
git branch -d <branchname>
#强制删除分支<branchname>，即使该分支没有合并到任何一个分支中。
git branch -D <branchname>
```

5.2 分支合并

```
#合并分支，会将目标分支与当前分支进行合并
git merge 目标分支名
```

合并操作并非总会成功，因为合并的不同提交可能同时修改了同一文件相同区域的内容，导致**冲突。**

```
#解决冲突
git mergetool
```

六、里程碑

可在gitlab上建立里程碑，这样大家在pull代码的时候也会拉取

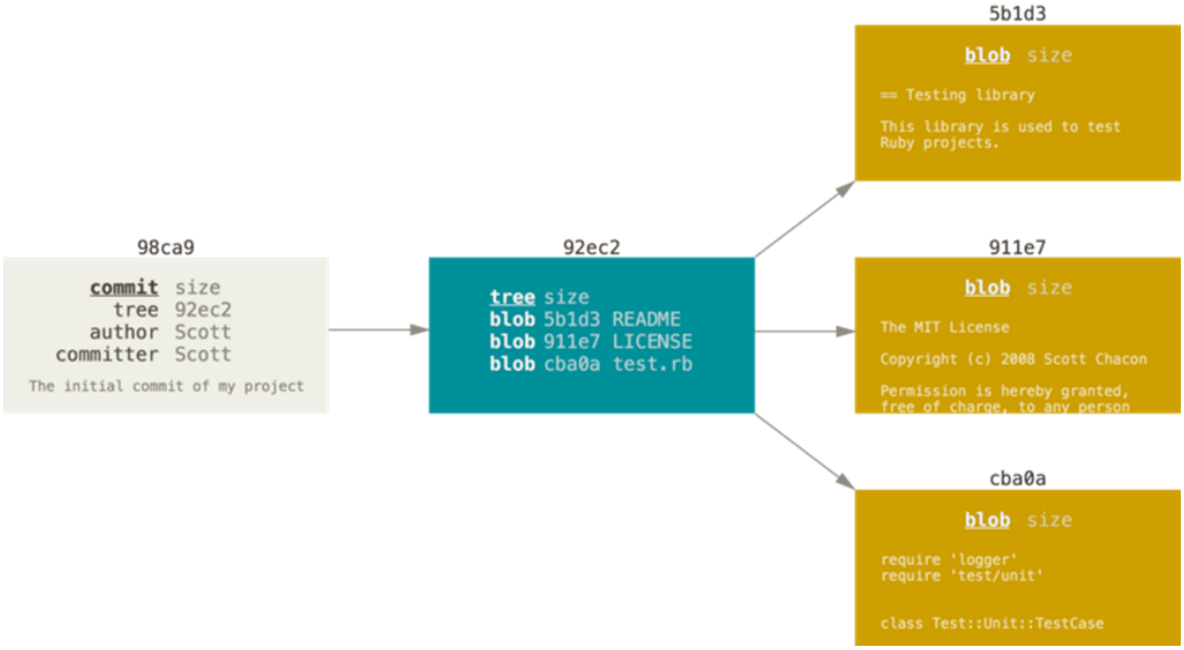
```
 #建一个本地的里程碑
 git tag  <tagname> -m "里程碑说明"
 #基于里程碑显示最新提交
 git describe
 #将里程碑共享到上游版本库
 git push origin mytag
```

七、推荐用法

推荐大家多用**分支**，如**定制化项目**单独做一条分支，这样改动不会影响到主要维护的产品，也方便管理。

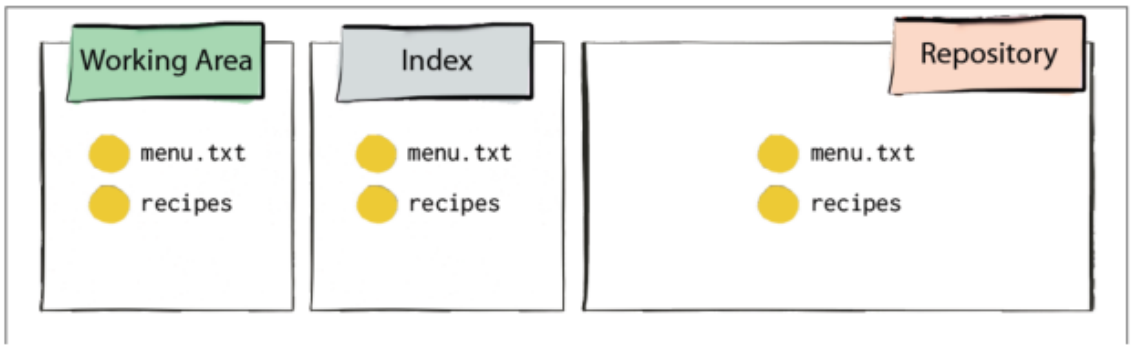
在本地分支进行开发，rebase整理好提交信息，然后合并到跟踪远程分支的本地分支，推送到远程仓库后，由管理员进行合并。

附1：commit快照对象

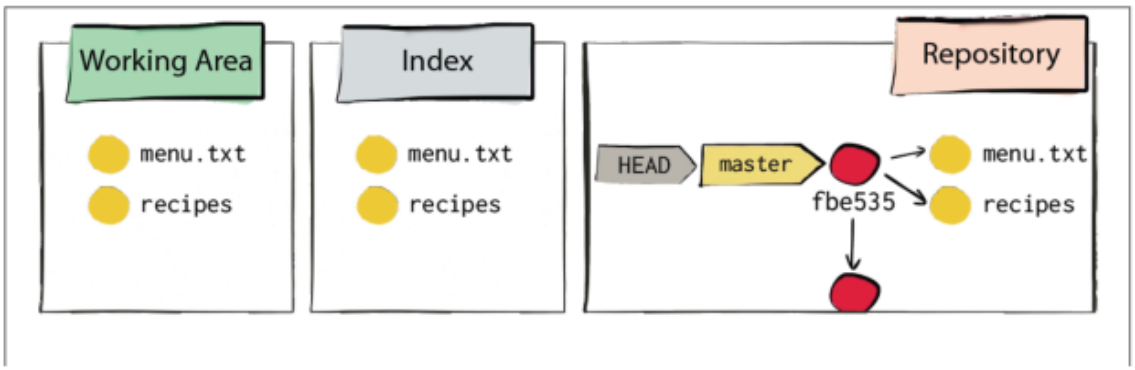


附2：Git指针指向变动过程

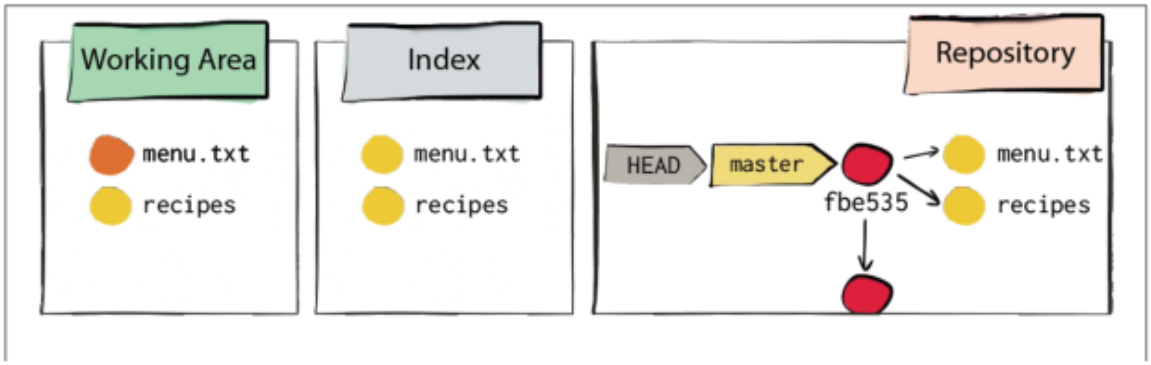
仓库中有两个文件



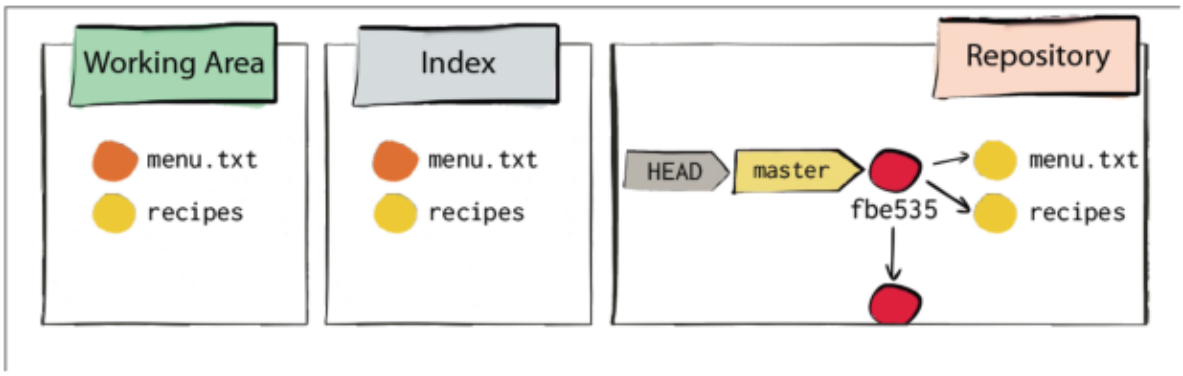
可以看到Git仓库存的commit提交是怎样的



这时候修改了menu.txt文件

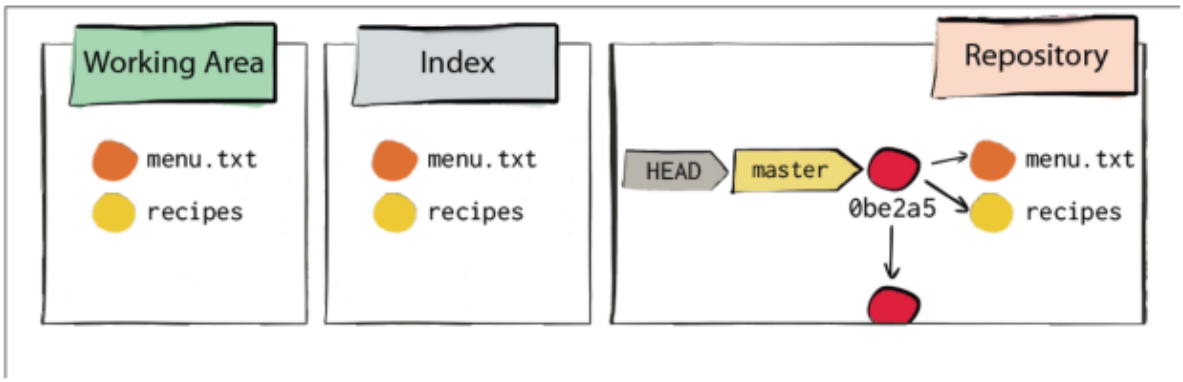


通过git add 添加到暂存区

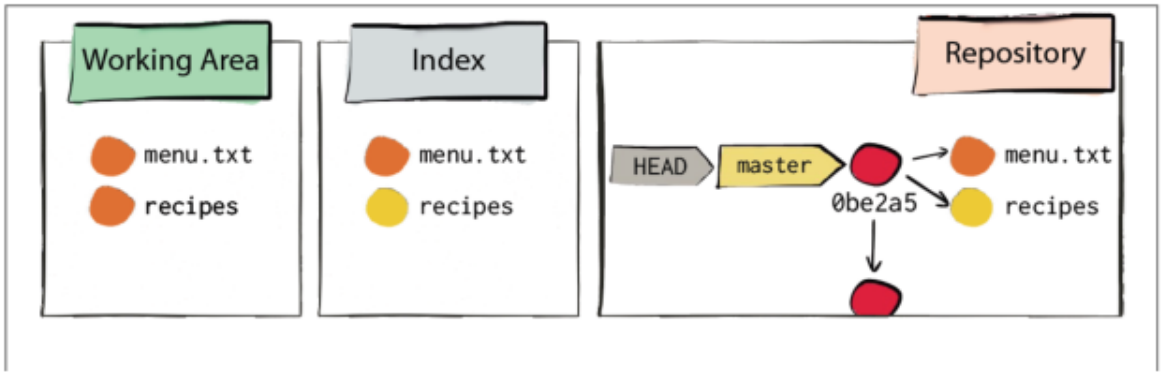


menu.txt暂存区文件快照提交到Git仓库，可以看到指针变了，

obe2a5指针的父指针是指向fbe535



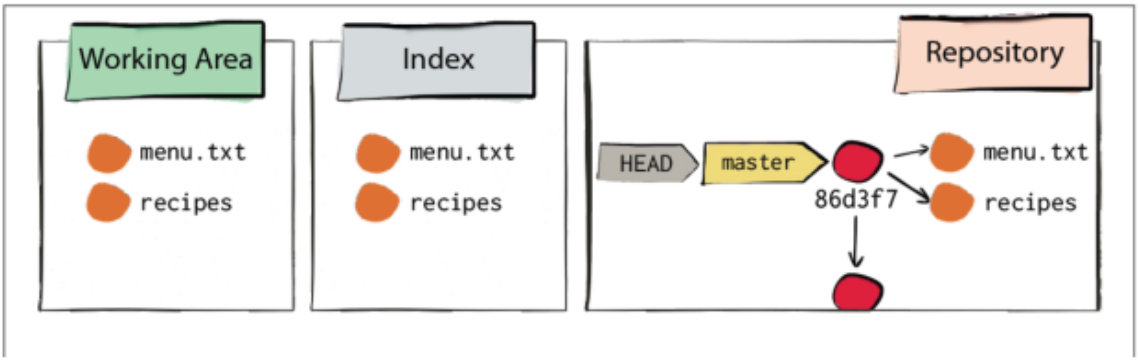
再修改recipes文件



git add recipes 将本地修改的文件快照添加到暂存区，然后git commit到Git仓库

可以看到指针又变了

86d3f7的父级指针是指向obe2a5



obe2a5指针的父指针是指向fbe535