Git 原理与实践

演变历史

版本控制类型	代表性工具	解决的问题
本地版本控制	RCS	本地代码的版本控制
集中式版本控制	SVN	提供一个远端服务器来维护代码版本，本地不保存代码版本，解决多人协作问题
分布式版本控制	Git	每个仓库都能记录版本历史，解决只有一个服务器保存版本的问题

本地版本控制

在早期的时候，项目版本控制都是在本地通过**复制文件以及文件命名**进行控制的，例如将不同版本的项目命名为 v1.0、v2.0 等。

早期较为成熟的版本控制工具如 RCS 就是利用这种方式，当然 RCS 并不是保存整个文件，**而是增量更新**，只有初代版本会保存所有内容，后续版本将以增加更新的形式保存，例如“某文件某偏移增加或删除了XXX内容”。

当然，本地版本控制的弊端非常明显，**协同合作非常困难**，因此演变出了集中式的版本控制。

集中式版本控制

SVN 是集中式版本控制的代言人，曾经流行过一段时间，SVN 将所有的版本变更以**增量更新（补丁）**的方式保存在远程服务器，允许开发人员从服务器拉去或提交。

但是集中式版本控制是中心化的，可靠性依赖于远程服务器，并且不支持复杂的分支操作， 于是 Git 孕育而生。

分布式版本控制

Git 允许本地暂存变更，所以说 Git 的每个仓库都能记录版本历史，开发人员无须联网也能进行版本控制，解决了 SVN 只有一个服务器保存版本的问题，是当下主流的版本控制工具。

Git 不同于之前的工具，**Git 每次更新都是会保存全部的文件而非增量更新，因此 Git 通常不支持 Commit 大文件。**

Git 原理

版本控制原理

Git 版本管理主要是通过 .git/Objects/ 路径下的文件进行管理，假设这里只简单的考虑 Commit，该路径下主要会存储三大重要信息：

- commit 信息：保存用户的 commit 记录，包括 commit message、author、committer、parent 以及最重要的 tree ID，**tree ID 是指向 tree 的文件。**
- tree 信息：用户 commit 可能涉及到多个文件的修改，一个 tree 文件中将保存多个 blog ID，blog 是指具体发生修改的文件。
- blog 信息：blog 即是指具体发生改动的内容文件，Git 会全量保存文件内容。

接着我们在本地进行实践，首先初始化 Git 仓库，然后创建 Readme 文件，在 Readme 文件中随便写点内容，提交到本地仓库中：

```
git init
vim Readme.md  # 然后自己写点内容
git add .           # 添加当前目录变更
git commit -m "create readme"
```

执行完上述步骤后，我们打印 .git 目录下文件：

这里可以发现 objects 目录下多出了三个文件，先通过 git log 命令查看下更新的日志：

```
[root]# git log
commit 854ebf0cbe82f7c6b06c35dfb48d5af2149d4643
Author: happysnaker <xxx@qq.com>
Date:   Sat May 21 15:26:44 2022 +0800

create readme
```

发现 commit 信息文件 ID 是 854ebf0cbe82f7c6b06c35dfb48d5af2149d4643，通过 git cat-file 命令查看文件：

```
[root]# git cat-file -p 854ebf0cbe82f7c6b06c35dfb48d5af2149d4643
tree 58c31878a99ff474aba5a0ad3cc751285003acc7
author happysnaker <xxx@qq.com> 1653118004 +0800
committer happysnaker <xxx@qq.com> 1653118004 +0800
```

```
create readme
```

这包含了我们了我们的提交信息，其中还包含了 tree 信息文件，同样查看一下 tree 文件：

```
[root]# git cat-file -p 58c31878a99ff474aba5a0ad3cc751285003acc7
100644 blob 557db03de997c86a4a028e1ebd3a1ceb225be238    Readme.md
```

tree 信息包含了 blog 文件信息，这就是我们修改的文件，**当然 tree 文件内可以包含多个修改的文件**，查看 blog 文件：

```
[root]# git cat-file -p 557db03de997c86a4a028e1ebd3a1ceb225be238
Hello World
```

他就是我们文件的全量内容，这就是 git 的版本控制原理，现在我们尝试对 readme 文件进行更改并且提交：

```
[root]# vim Readme.md
[root]# git add .
[root]# git commit -m "update readme"
```

继续查看一些文件树：

```
[root@VM-16-3-centos GitStudy]# tree .git/objects
.git/objects
|-- 3c
|   `-- 8de40439f7917b188e7c3271409794ec635ef3
|-- 55
|   `-- 7db03de997c86a4a028e1ebd3a1ceb225be238
|-- 58
|   `-- c31878a99ff474aba5a0ad3cc751285003acc7
|-- 85
|   `-- 4ebf0cbe82f7c6b06c35dfb48d5af2149d4643
|-- 99
|   `-- 3562adc29d6b58ea4ec77a3cf5f8cc8da4c97d
|-- ce
|   `-- 81273cc3c46c1e7543355e01010685aab9412d
|-- info
`-- pack
```

发现这里又多了三个文件，这反应出**Git 每次提交都会新建 commit、tree 和 blog 文件**，通过 git log 查看 commit 文件信息：

```
[root]# git log
commit 3c8de40439f7917b188e7c3271409794ec635ef3
Author: happysnaker <1637318597@qq.com>
Date:   Sat May 21 15:40:09 2022 +0800
```

```
    update readme

commit 854ebf0cbe82f7c6b06c35dfb48d5af2149d4643
Author: happysnaker <1637318597@qq.com>
Date:   Sat May 21 15:26:44 2022 +0800

    create readme
```

这里最新的 commit 文件是 3c8de40439f7917b188e7c3271409794ec635ef3，查看这个文件内容：

```
[root]# git cat-file -p 3c8de40439f7917b188e7c3271409794ec635ef3
tree ce81273cc3c46c1e7543355e01010685aab9412d
parent 854ebf0cbe82f7c6b06c35dfb48d5af2149d4643
author happysnaker <1637318597@qq.com> 1653118809 +0800
committer happysnaker <1637318597@qq.com> 1653118809 +0800

update readme
```

这与我们之前所说的是一模一样的，唯一的不同是这里多了个 parent 字段，它刚好指向我们上一个 commit 版本。

所以说，Git 其实就是通过 commit、tree 和 blog 文件来进行版本控制，Git 将进行全量保存而非增量更新，Git 只需只需记录下每次 commit 所产生的 commit 文件，进行版本切换的时候，直接找到 commit 文件，并找到对应的 blog 文件，将工作区文件替换成 blog 文件便可完成版本控制。

分支控制原理

那 Git 的分支切换呢？Git 是如何进行分支控制的？一个分支是如何不对另一个分支产生影响的呢？

其实这很简单，在上面我们已经讲了 commit 相关原理，每一次 commit 都会产生一个 commit 文件，那这样我们可以将每个 commit 看作是一个节点，**分支的移动变更其实就是在节点中不停的切换**，那对应到 Git 的实现中呢，其实就是在分支文件中保存不同的 commit 文件 ID。

这里可以去 Git 可视化网站体验一下：Learn Git Branching

如果仔细观察 .git 目录，你就会发现目录下 HEAD 是指向当前工作区的版本，而分支则在 refs/heads 路径下。

我们通过实践来感受，在上面的基础上，建立 dev 分支，并且更改 Readme 内容后并提交：

```
[root@VM-16-3-centos GitStudy]# git checkout -b dev
Switched to a new branch 'dev'
```

```
[root@VM-16-3-centos GitStudy]# vim Readme.md
[root@VM-16-3-centos GitStudy]# git add .
[root@VM-16-3-centos GitStudy]# git commit -m "dev update"
```

先查看一下 log：

```
[root@VM-16-3-centos GitStudy]# git log
commit 2f4e8231544311974d9ddd5db47ec3cc6dcc462f
Author: happysnaker <1637318597@qq.com>
Date:   Sat May 21 15:59:32 2022 +0800

    dev update

commit 3c8de40439f7917b188e7c3271409794ec635ef3
Author: happysnaker <1637318597@qq.com>
Date:   Sat May 21 15:40:09 2022 +0800

    update readme

commit 854ebf0cbe82f7c6b06c35dfb48d5af2149d4643
Author: happysnaker <1637318597@qq.com>
Date:   Sat May 21 15:26:44 2022 +0800

    create readme
```

这次在 dev 分支的提交 ID 是 2f4e8231544311974d9ddd5db47ec3cc6dcc462f，现在我们打印 .git 路径树：

首先来看看这次最新提交的 commit 文件内容：

```
[root]# git cat-file -p 2f4e8231544311974d9ddd5db47ec3cc6dcc462f
tree 1f16b25244adda7d4590841975baabda6a41c328
parent 3c8de40439f7917b188e7c3271409794ec635ef3
author happysnaker <1637318597@qq.com> 1653119972 +0800
committer happysnaker <1637318597@qq.com> 1653119972 +0800

dev update
```

然后我们打印一下 HEAD 文件、refs/heads/dev 和 refs/heads/master 文件看看里面的内容到底是什么：

```
[root@VM-16-3-centos GitStudy]# git cat-file -p HEAD
tree 1f16b25244adda7d4590841975baabda6a41c328
```

```
parent 3c8de40439f7917b188e7c3271409794ec635ef3
author happysnaker <1637318597@qq.com> 1653119972 +0800
committer happysnaker <1637318597@qq.com> 1653119972 +0800

dev update
```

发现 HEAD 文件和本次 commit 文件是一致的，这很正常，HEAD 是你当前的工作版本，因此 HEAD 与当前提交是一致的，来看看分支文件：

```
[root@VM-16-3-centos GitStudy]# git cat-file -p master
tree ce81273cc3c46c1e7543355e01010685aab9412d
parent 854ebf0cbe82f7c6b06c35dfb48d5af2149d4643
author happysnaker <1637318597@qq.com> 1653118809 +0800
committer happysnaker <1637318597@qq.com> 1653118809 +0800

update readme
[root@VM-16-3-centos GitStudy]# git cat-file -p dev
tree 1f16b25244adda7d4590841975baabda6a41c328
parent 3c8de40439f7917b188e7c3271409794ec635ef3
author happysnaker <1637318597@qq.com> 1653119972 +0800
committer happysnaker <1637318597@qq.com> 1653119972 +0800

dev update
```

你发现了吗？master 分支仍然是指向上一次的版本更改，这其实就是 Git 版本管理的原理，**Git 维护当前所处的分支，并且会实时更新 HEAD 文件和 ref/heads 目录下当前分支的内容，而其他分支文件内容不会更改，所以其他分支是察觉不到本次更改的，当分支切换时，HEAD 也将切换到对应分支，实现了隔离管理。**

试着切回 master 分支，并查看 HEAD：

```
[root@VM-16-3-centos GitStudy]# git checkout master
Switched to branch 'master'
[root@VM-16-3-centos GitStudy]# git cat-file -p HEAD
tree ce81273cc3c46c1e7543355e01010685aab9412d
parent 854ebf0cbe82f7c6b06c35dfb48d5af2149d4643
author happysnaker <1637318597@qq.com> 1653118809 +0800
committer happysnaker <1637318597@qq.com> 1653118809 +0800

update readme
```

发现 HEAD 也切换到了之前 master 的版本，在这种状态下，我们当然是感知不到 dev 分支所做的更改，因为当前 commit 指向的 blog 文件还是之前的版本。

那合并分支呢？合并分支的原理其实也很简单，例如在 master 分支下执行`git merge dev`命令，这将选取两者分支中最新的一次提交，**并将当前所处的分支快速推进到最新的提交版本。**

```
[root@VM-16-3-centos GitStudy]# git merge dev
Updating 3c8de40..2f4e823
Fast-forward
 Readme.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
[root@VM-16-3-centos GitStudy]# git cat-file -p master
tree 1f16b25244adda7d4590841975baabda6a41c328
parent 3c8de40439f7917b188e7c3271409794ec635ef3
author happysnaker <1637318597@qq.com> 1653119972 +0800
committer happysnaker <1637318597@qq.com> 1653119972 +0800

dev update
```

看，合并后，master 分支被推进到最新的版本了！本地提交是不会产生冲突的，但 push 到远程时，**发现远程分支版本已经不是上一次 clone 时的版本了**，那么便会产生冲突，因此在 push 之前我们通常需要 git pull 拉取最新的更改并合并之后再做提交，在这一步时，如果远程分支与你恰好修改了同一个文件，这会产生冲突，Git 会提示你并将冲突文件展现出来，这包含双方的修改，你可以自行选择删除或修改一些内容。解决冲突后，分支已经合并，现在可以正常 push 了。

标签管理

标签管理原理非常简单，每个标签文件(refs/tag 目录下)其实就是保存着一个稳定版本的 commit 文件，标签通常不会被修改。

常用语句

1. git remote add orgin git@github.com:username/repo.git，通过 ssh 建立远程仓库。
2. git push -u origin master，提交到远程仓库的 master 分支。
3. git merge xxx，将当前分支与 xxx 分支合并，并将当前分支推进到最新的分支。
4. git pull，等价于 git fetch + git merge，可以通过 –rebase 参数使分支树更加整洁。
5. git rebase xxx，同 merge，rebase 将会使分支树更加整洁，通过 -i 参数可交互式的调整分支版本。
6. git commit –amend，修改 commit 的信息。
7. git reset，将 HEAD 推进到对应版本，例如 git reset HEAD^，这将推进到 HEAD 的上一个版本，注意，一旦 HEAD 和你当前分支的 commit 分离，此时 Git 不允许你提交。
8. git cat-file -p filename，查看 git 目录下文件内容。
9. git log，打印 commit 日志。