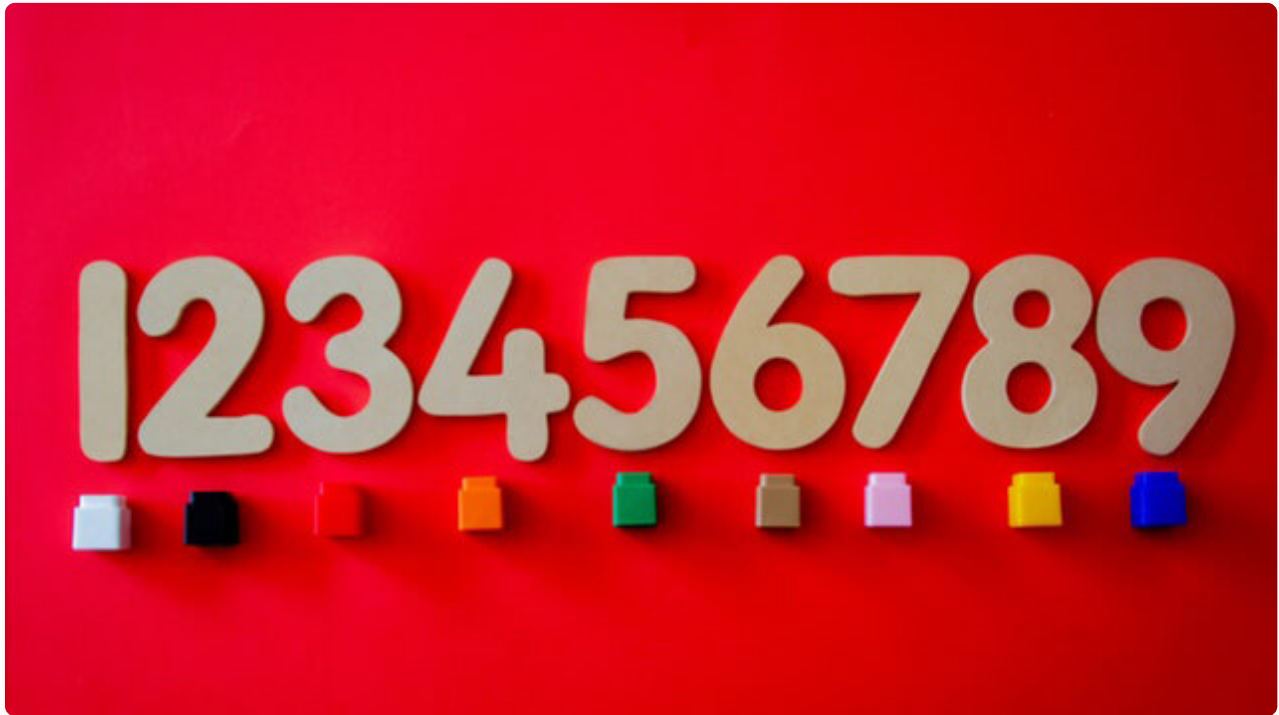


29 历史记录清理：保留代码并删除一年前的提交记录

更新时间：2019-12-04 10:20:18



“

今天应做的事没有做，明天再早也是耽误了。

——裴斯泰洛齐

”

Git 仓库使用久了之后会发现响应会越来越慢，占用的存储空间也会越来越大，出现这些问题的原因是因为 **Git** 会将我们代码的所有历史版本都会存储起来，当我们代码迭代了上千个版本之后就会占用很大的空间。

为了让 **Git** 的响应速度能够快点，我们可以通过一些方式减少 **Git** 仓库的存储空间，这一节当中主要教大家三种方式，分别是：克隆最后一个版本、清空版本记录、清理大文件等。

29.1 克隆最后一个版本

我们在克隆一个使用了很久的 **Git** 仓库时候可能会发现耗费时间非常长，这是因为 **Git** 会拉取所有的历史版本导致，我们如果能让 **Git** 在克隆代码的时候快一些，可以在 `git clone` 命令行后面加 `--depth=1` 参数，这样 **Git** 在克隆代码时候只会克隆最后一个版本的内容，这里我执行命令如下所示：

```
git clone https://gitee.com/songboy/test201907.git tempdemo --depth==1
```


命令执行完毕之后，Git 返回的信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/test201907 (zsh) — 111X4
→ test201907 git:(develop) cd ../test201907 && git checkout develop
已经位于 'develop'
您的分支与上游分支 'origin/develop' 一致。
→ test201907 git:(develop) []
```

在上图中可以看到已经切换到 `develop` 分支当中。

接着我们创建一个新分支，不过在创建的时候我们需要加上一个 `--orphan` 参数，加上这个参数之后创建的分支有点特殊，他只有最后一个版本，而不是把所有的版本都复制过来，严格来说创建出来的不是分支，但很像分支，执行的命令如下所示：

```
git checkout --orphan new_branch
```

命令执行完毕之后，Git 返回的信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/test201907 (zsh) — 111X3
→ test201907 git:(develop) git checkout --orphan new_branch
切换到一个新分支 'new_branch'
→ test201907 git:(new_branch) x []
```

在上图中可以看到已经创建了一个 `new_branch` 分支成功，接着我们将这个特殊的分支里面的文件都添加进来，执行的命令如下所示：

```
git add -A && git status
```

命令执行完毕之后，Git 仓库的文件状态如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/test201907 (zsh)
→ test201907 git:(new_branch) x git add -A && git status
位于分支 new_branch

尚无提交

要提交的变更：
（使用 "git rm --cached <文件>..." 以取消暂存）
新文件：.gitattributes
新文件：.gitignore
新文件：README.en.md
新文件：README.md
新文件：aa.txt
新文件：app/diff.php
新文件：bb.txt
新文件：ccc.txt
新文件：config/config.php
新文件：index.php
新文件：test/index.php
新文件：test2/test.txt
新文件：test3/test.txt
→ test201907 git:(new_branch) x []
```

在上图中可以看到所有的文件都处于待添加状态，我们将这些文件使用 `commit` 命令提交到一个版本当中去，执行命令如下所示：

```
git commit -m "new version"
```

命令执行完毕之后，Git 返回的信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/test201907 (zsh)
→ test201907 git:(new_branch) * git commit -m "new version"
[new_branch (根提交) 130d9e1] new version
13 files changed, 98 insertions(+)
create mode 100644 .gitattributes
create mode 100644 .gitignore
create mode 100644 README.en.md
create mode 100644 README.md
create mode 100644 aa.txt
create mode 100644 app/diff.php
create mode 100644 bb.txt
create mode 100644 ccc.txt
create mode 100644 config/config.php
create mode 100755 index.php
create mode 100644 test/index.php
create mode 100644 test2/test.txt
create mode 100644 test3/test.txt
→ test201907 git:(new_branch) []
```

在上图中可以看到提交成功，并生成了一个版本记录，接着我们将原来的 `develop` 分支删除，执行命令如下所示：

```
git branch -D develop
```

命令执行完毕之后，Git 返回的信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/test201907 (zsh)
→ test201907 git:(new_branch) git branch -D develop
已删除分支 develop (曾为 5b0979f) 。
→ test201907 git:(new_branch) []
```

在上图中可以看到已经将 `develop` 分支删除成功，接着我们在将当前所在的 `new_branch` 分支改名为 `develop`，执行命令如下所示：

```
git branch -m develop
```

命令执行完毕之后，Git 返回的信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/test201907 (zsh)
→ test201907 git:(new_branch) git branch -m develop
→ test201907 git:(develop) []
```

在上图中可以看到已经将分支改名成功，接着我们使用 `git push -f` 命令将本地仓库强制推送到远程仓库当中，这里需要注意：

有些仓库有 `master` 分支保护，不允许强制 `push`，需要在远程仓库项目里暂时把项目保护关掉才能推送

```
git push -f origin develop
```

命令执行完毕之后，Git 远程仓库返回的信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/test201907 (zsh)
→ test201907 git:(develop) git push -f origin develop
枚举对象：20，完成。
对象计数中：100% (20/20)，完成。
使用 4 个线程进行压缩
压缩对象中：100% (5/5)，完成。
写入对象中：100% (20/20)，2.31 KiB | 1.15 MiB/s，完成。
总共 20 （差异 0），复用 0 （差异 0）
remote: Powered by GITEE.COM [GNK-3.8]
To https://gitee.com/songboy/test201907.git
+ 5b0979f...130d9e1 develop -> develop (forced update)
→ test201907 git:(develop) []
```

在上图中可以看到强制推送到远程的 `develop` 分支已经成功，此时我们已经将远程的历史版本记录给覆盖。

29.3 清理大文件

上面一小节中，我们可以通过清理版本库来减少 Git 存储的空间，但有时候我们需要保留历史版本记录，但又想减少 Git 存储空间，这个时候我们就可以考虑清理一些大文件。

Git 本身也给我们提供了解决方案，使用 `git branch-filter` 可以遍历 Git 的版本历史信息，然后从历史版本信息中删除大文件，最终 Git 仓库空间减少，在下面的命令中会涉及到很多 Linux 命令，这些命令我们不用细究，按照步骤执行即可。

演示出效果需要一个使用比较久，而且里面有大文件的仓库，这里我以我们公司的一个仓库为例来演示。

现在需要找出大文件的对应 hash 值，这里我们找出前 5 个为例，执行命令如下所示：

```
git verify-pack -v .git/objects/pack/pack-*.idx | sort -k 3 -g | tail -5
```

命令执行完毕之后，过滤后的 Git 返回信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/work/xiaoyu (zsh)
→ xiaoyu git:(develop) * git verify-pack -v .git/objects/pack/pack-*.idx | sort -k 3 -g | tail -5
08d5d0a258c915e3804c6690c35b63fa7197b8b4 blob 4549417 2760121 24939266 1 6ba572e5b6b9237a29bd883595e82f5a48e62a66
dee2b11f61c64924780a0ba3971e3277290189ae blob 7470823 3587551 719777
28cb021985af294ed78d79e3bcfce953769b3b41 blob 11425117 9743085 16531904
b1448621c13d8f4f2c17894ce2b031835b0bc4f7 blob 16233369 10806827 57826
6ba572e5b6b9237a29bd883595e82f5a48e62a66 blob 21087169 15402148 9537118
→ xiaoyu git:(develop) * []
```

在上图中，可以看到 5 行记录，其中第 3 列代表文件占用空间大小，我们随意挑选一个 hash 值为例，将这个值 `6ba572e5b6b9237a29bd883595e82f5a48e62a66` 复制到剪贴板，然后根据 hash 值找到对应大文件名，执行命令如下所示：

```
git rev-list --objects --all | grep 6ba572e5b6b9237a29bd883595e82f5a48e62a66
```

命令执行完毕之后，Git 返回的信息过滤后如下图所示：

```
song@tangqinongdeMBP: ~/mycode/work/xiaoyu (zsh)
→ xiaoyu git:(develop) x git rev-list --objects --all | grep '6ba572e5b6b9237a29bd883595e82f5a48e62a66'
6ba572e5b6b9237a29bd883595e82f5a48e62a66 vendor.zip
→ xiaoyu git:(develop) x
```

在上图中可以看到这个 hash 所对应的文件为 `vendor.zip` 文件，从文件名可以看出这是一个压缩包，我们要清除这个文件在所有历史中的记录，并强制刷新到所有分支，这里推送到远程仓库需要有强制推送权限。执行删除 `vendor.zip` 文件，在所有历史版本中的记录命令如下所示：

```
git filter-branch --index-filter 'git rm --cached --ignore-unmatch vendor.zip'
```

命令执行完毕之后，Git 返回的信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/xiaoyu (zsh)
Rewrite ee1838bccfd4052066ad9dcabccd4fb48a51106c (1563/1584) (125 seconds passed, remaining 1 predicted) rm 'vendor.zip'
Rewrite acfc8a7783a145f19a2c034c527d81e8bcd0075 (1563/1584) (125 seconds passed, remaining 1 predicted) rm 'vendor.zip'
Rewrite cddd090e8cd2cbadea9a200401df6bc3828a0a66 (1563/1584) (125 seconds passed, remaining 1 predicted) rm 'vendor.zip'
Rewrite ffb6227ffe55b8b291bb4185d4af0686a983b092 (1563/1584) (125 seconds passed, remaining 1 predicted) rm 'vendor.zip'
Rewrite 8a74fd9d2367fd1724c722efdbb696bc81a946e0 (1563/1584) (125 seconds passed, remaining 1 predicted) rm 'vendor.zip'
Rewrite 3c6a59d99e93b3eeffc0ee3fe8c3c302838be1e1 (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'
Rewrite ee95292a717a597247b6164520cc9610657fb63f (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'
Rewrite eb301c717c14ed20212a2051e96339df9c54abf5 (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'
Rewrite 9d4ad9ba9e013bbb2588dadfde5db7e620b696d7 (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'
Rewrite 757424399a0432ce9613f2ff7f2d3681b461af8b (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'
Rewrite 4bda51bb460f26c38220a998b928f4ad84b3a611 (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'
Rewrite d51a2423743782a728e4287ea75ba46a7e5f7098 (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'
Rewrite 8df1cfc289f807955d381ed4ae45d8d0a268a39f (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'
Rewrite c095b518188af7d54974c4e07981b4cd8300e71c (1576/1584) (126 seconds passed, remaining 0 predicted) rm 'vendor.zip'

Ref 'refs/heads/develop' was rewritten
→ xiaoyu git:(develop)
```

在上图中可以看到当前仓库有 `1584` 个版本记录，已全部删除完毕；在上面的命令中我们删除了文件，但是在 Git 的 repo 里面还记录了这些文件的信息，这些信息也会占用一定的空间，我们继续清除这些信息，并收回存储空间，执行命令如下所示：

```
rm -rf .git/refs/original/ && git reflog expire --expire=now --all
```

命令执行完毕之后，Git 返回的信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/xiaoyu (zsh)
→ xiaoyu git:(develop) rm -rf .git/refs/original/ && git reflog expire --expire=now --all
→ xiaoyu git:(develop)
```

在清除多余信息之后，我们需要重新建立文件与 Git 仓库的关联关系，执行命令如下：

```
git fsck --full --unreachable
```


命令执行完毕之后，Git 返回信息如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/xiaoyu (zsh)
不可达 tree 46febb7621f8e41003461a60021e8f2e4e85e075
不可达 tree 7ffe2fc2607d63f82d04e2cf8b55cbc19970280b3
不可达 tree 8bfe6dac5b16bfdc859aa3ac4cadfb3a7e930147
不可达 blob 93fe7b48d4a1577c0340e86107f1707235f92ac7
不可达 tree 95febbb3903eac8dc06e0579bb71ea35ad5ef45a
不可达 tree 9afeadd881e18306655b7e91c4fd5c7678cf3801
不可达 tree a6fe215fb5c80ea0bc33922df12de417640e553
不可达 blob bdfec1a9edc663ad4b45f0972c17f165253c540b
不可达 tree c3fe6fcffa45a6f1772cd99e8d91e367ff426c27
不可达 tree d8febf4d72043409ece94f2429a41d97661b6994
不可达 tree 29ffbd04ae9d7b93c5c5e3bad3553d2afe01a41c
不可达 blob 35ffe3f9be71e0e64ac3bc9471647f1b94d61b2a
不可达 tree 9dff39fc7572c2bc1631c9b49bd765edb5f5f00bc
不可达 tree ceff4fe047d5e99d748ad3c7b7656963c61a38f3
不可达 tree eeffc51e30565323684231e01fd64b808cd9ef0b
不可达 blob f6ffe3b6a267b6ecd2ab748a7eaa144824f98c1
→ xiaoyu git:(develop) []
```

接着我们需要重新压缩代码，减少仓库体积：

```
git repack -A -d
```

命令执行完毕之后，如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/xiaoyu (zsh) — 135X8
→ xiaoyu git:(develop) git repack -A -d
枚举对象：38091，完成。
对象计数中：100% (38091/38091)，完成。
使用 4 个线程进行压缩
压缩对象中：100% (17250/17250)，完成。
写入对象中：100% (38091/38091)，完成。
总共 38091 （差异 22435），复用 33087 （差异 19962）
→ xiaoyu git:(develop) []
```

在上图中可以看到压缩任务已经执行完成，最后可以通过 Git 的 GC 清理一些垃圾数据，执行命令如下所示：

```
git gc --aggressive --prune=now
```

命令执行完毕之后，Git 仓库的文件状态如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/xiaoyu (zsh) — 135X8
→ xiaoyu git:(develop) git gc --aggressive --prune=now
枚举对象：38091，完成。
对象计数中：100% (38091/38091)，完成。
使用 4 个线程进行压缩
压缩对象中：100% (37212/37212)，完成。
写入对象中：100% (38091/38091)，完成。
总共 38091 （差异 23424），复用 14563 （差异 0）
→ xiaoyu git:(develop) []
```

在上图中可以看到已经回收完成，总记录表有 38091 个对象，可用的只有 14563，其余的便被回收了，刚才这些操作都是在本地，我们如果能让远程仓库也清理，可以强制推送到远程仓库，执行命令如下所示：

```
git push --force origin master
```

命令执行完毕之后，Git 仓库的文件状态如下图所示：

```
song@tangqinongdeMBP: ~/mycode/test/xiaoyu (zsh)
→ xiaoyu git:(develop) git push --force origin develop -f
枚举对象：10235, 完成.
对象计数中：100% (10048/10048), 完成.
使用 4 个线程进行压缩
压缩对象中：100% (3377/3377), 完成.
写入对象中：100% (9930/9930), 7.09 MiB | 3.91 MiB/s, 完成.
总共 9930 (差异 6747) , 复用 9646 (差异 6501)
remote: Resolving deltas: 100% (6747/6747), completed with 67 local objects.
To http://localhost:8090/root/testgitlab.git
+ c095b518...cee95dad develop -> develop (forced update)
→ xiaoyu git:(develop) █
```

在上图中可以看到已经推送到远程仓库成功。

29.4 小结

在这一节当中主要学习了如何让 Git 仓库瘦身，以达到让 Git 响应速度变快的同时也减少占用仓库的空间。

导致 Git 仓库太大的原因，通常是因为有大文件和迭代版本次数过多导致，前者我们尽量不要把大文件放到存储空间中去，后者的话很难避免，不过我们可以定期清理版本记录。

}