Git 碰到的一些问题记录

git 查询用户代码数

```
git log --author="fangcong" --pretty=tformat: --numstat | awk '{ add += $1;
subs += $2; loc += $1 - $2 } END { printf "added lines: %s, removed lines: %s,
total lines: %s\n", add, subs, loc }'
```

git 批量删除分支

```
git branch | grep 'branchName' | xargs git branch -D
```

这是通过 shell 管道命令来实现的批量删除分支的功能

```
git branch 输出当前分支列表
grep 是对 git branch 的输出结果进行匹配,匹配值当然就是 branchName
xargs 的作用是将参数列表转换成小块分段传递给其他命令
```

因此, 这条命令的意思就是:

从分支列表中匹配到指定分支,然后一个一个(分成小块)传递给删除分支的命令,最后进行删除。

从而就达到了我们想要批量删除分支的目的。

git更新远程分支列表

```
git remote update origin --prune
```

git 本地仓库关联远程仓库

```
git remote add origin https://github.com/gakkiyomi/study-log.git
```

git 本地分支跟踪远程分支

```
git branch --set-upstream-to=origin/master
```

git diff 不显示修改的代码

```
git diff --cached
src/main/java/net/skycloud/cmdb/common/utils/CommonUtils.java
```

git commit message 提交错了, 对上次提交进行修改

```
git commit --amend -m 'feat:2020.6.16'
```

git 删除本地分支和远程分支

```
git branch -D test. #删除本地分支
git push origin --delete test #删除远程分支s
```

git 查看最近n次的提交内容

指定n为1则可以查看最近一次修改的内容

```
git log -p -n
```

知道commit id的情况

```
git show commit_id
```

git 查看所有分支的提交修改

```
git log --all --graph --decorate
```

git 合并冲突

需要将dev1分支的内容合并到master分支

```
git checkout master #切换到master分支
git merge dev1
```

```
git merge dev2
#此时如果出现情况,不想合并,可以使用一下命令
git merge --abort #返回合并前的状态
#再次执行合并
git merge dev2
#冲突,手动解决
vim xxx.txt
git add .
git merge --continue
```

git message 格式

• feat: 新功能 (feature)

• fix: 修补bug

• docs: 文档 (documentation)

• style: 格式 (不影响代码运行的变动)

• refactor: 重构(即不是新增功能,也不是修改bug的代码变动)

• test: 增加测试

• chore:构建过程或辅助工具的变动

git 历史commit总数

```
git rev-list --all --count
```

包含了所有分支中的提交。

alias

上面的命令非常复杂难记

可以定义一个 alias:

打开 ~/.gitconfig

在 alias 部分增加一行配置

```
[alias]
count = rev-list --all --count
```

```
git count
```

git log graph

```
git log --graph --decorate --oneline --simplify-by-decoration --all
```

- --decorate 标记会让git log显示每个commit的引用(如:分支、tag等)
- --oneline 一行显示
- --simplify-by-decoration 只显示被branch或tag引用的commit
- --all 表示显示所有的branch,这里也可以选择,比如我指向显示分支ABC的关系,则将--all替换为**branchA branchB branchC**

切换远程仓库地址

1. 直接修改

```
git remote set-url origin 新的项目路径
```

2. 先删除远程地址, 然后添加新的仓库地址

```
git remote rm origin
git remote add origin url
```

3. 修改配置文件

修改 .git目录下config文件中的url

全局修改用户名

```
git config --local(global) user.name 'gakkiyomi'
```

仓库级(全局)修改用户邮箱

```
git config --local(global) user.email 'gakkiyomi@gmail.com'
```

.git/objects/pack文件夹变得非常大

首先看一下.git目录

```
├── HEAD
├── branches
├── index
├── logs
```



- **description** 用于GitWeb程序
- config 配置特定于该仓库的设置
- 放置客户端或服务端的hook脚本
- **HEAD** 指明当前处于哪个分支
- *objects* Git对象存储目录
- refs Git引用存储目录
- branches 放置分支引用的目录

每次 $git\ add$ 都会生成一个Git对象,称为 $blob\ 对象$,存放在objects目录下。

这个blob 对象里保存的是什么呢?

Git在add文件时,会把文件完整的保存成一个新的**blob** 对象。通过 *git* 打包或者每次*git* 的时候Git都会自动执行一次打包过程,将Blob对象合并成一个包文件,同时会生成一个索引文件,索引文件中包含了每个Blob对象在包文件中的偏移信息,Git在打包的过程中使用了增量编码方案,只保存Blob对象的不同版本之间的差异,这使得仓库会瘦身。

既然Git会对Blob对象进行合并优化,那么objects文件夹为什么还会那么大呢?

因为当Blob对象在合并时不能对.a进行差异化比较,所以每次在添加.a文件时,都会保存一份.a文件,用于后续代码还原时使用。

所以当频繁更换.a文件时,objects下的pack文件会越来越大。虽然这个.a文件后续可能用不到删除了,但是pack中的这个.a文件的缓存还是会一直存在。

删除pack中无用的大文件缓存

首先先找出git中最大的文件:

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

执行结果:

第一行的字母其实相当于文件的id、用以下命令可以找出id对应的文件名:

git rev-list --objects --all | grep 8f10eff91bb6aa2de1f5d096ee2e1687b0eab007

找到最大的几个文件后, 怎么删除呢?

能够胜任这个任务的命令叫做 filter-branch:

git filter-branch --force --prune-empty --index-filter 'git rm -rf --cached --ignore-unmatch nmap-driver.tar' --tag-name-filter cat -- --all

等命令执行完后,要提交到远程:

git push --force --all

git pull报错 fatal: refusing to merge unrelated histories

原因是两个分支是两个不同的版本,具有不同的提交历史

git pull origin main --allow-unrelated-histories

使用上面的命令可以允许不相关历史提,强制合并

git rebase -i

使用git rebase -i 来将多次commit合并为一个commit。

我们分别提交2个commit rebase 1 and rebase 2 and rebase 3, rebase 4 and rebase 5 and rebase 6

然后执行

git rebase -i HEAD~2

会弹出一个界面

```
git (vim)
ick 52ca5e5 rebase 1 and rebase 2 and rebase 3
pick 5c9646f rebase 4 and rebase 5 and rebase 6
pick 11d608e rebase 7
# Rebase c7bc422..11d608e onto c7bc422 (3 commands)
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
f f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
 t, reset <label> = reset HEAD to a label
 m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
         create a merge commit using the original merge commit's
         message (or the oneline, if no original merge commit was
         specified). Use -c <commit> to reword the commit message.
 These lines can be re-ordered; they are executed from top to bottom.
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
 Note that empty commits are commented out
```

将你要合并的那些commit 的pick 改成s(squash)压缩

```
pick 52ca5e5 rebase 1 and rebase 2 and rebase 3
s 5c9646f rebase 4 and rebase 5 and rebase 6
 11d608e rebase 7
# Rebase c7bc422..11d608e onto c7bc422 (3 commands)
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
         create a merge commit using the original merge commit's
         message (or the oneline, if no original merge commit was
         specified). Use -c <commit> to reword the commit message.
# These lines can be re-ordered; they are executed from top to bottom.
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
# Note that empty commits are commented out
```

然后又会弹出一个界面,填写这几个commit合并后的commit-massage 这里我写了rebase 1 to 8

```
git (vim)
This is a combination of 4 commits.
This is the 1st commit message:
ebase 1 to 8
This is the commit message #2:
rebase 4 and rebase 5 and rebase 6
This is the commit message #3:
rebase 7
This is the commit message #4:
78
Please enter the commit message for your changes. Lines starting
with '#' will be ignored, and an empty message aborts the commit.
          Fri Mar 5 13:51:46 2021 +0800
Date:
interactive rebase in progress; onto c7bc422
Last commands done (4 commands done):
   squash 11d608e rebase 7
   squash a9d9acb 78
No commands remaining.
You are currently rebasing branch 'master' on 'c7bc422'.
Changes to be committed:
      modified: md/Git.md
```

退出后查看git log 合并完成

```
commit 76ed35789634429d97245d048421aba15f32c3d6 (HEAD -> master)
Author: fangcong <fangcong@sky-cloud.net>
Date: Fri Mar 5 13:51:46 2021 +0800

    rebase 1 to 8

commit c7bc4229d468e222948afdf2471f2511b3da8031 (origin/master)
Author: fangcong <fangcong@sky-cloud.net>
Date: Fri Mar 5 13:49:12 2021 +0800

    feat

commit a2803bac173fafb33a39debe8a24b5bc871e0fdf
Author: fangcong <fangcong@sky-cloud.net>
Date: Fri Mar 5 13:45:17 2021 +0800

by 2021-03-05 13:45:17
```