Git学习笔记

1. Git是一个分布式版本控制系统，而CVS和SVN是集中式版本控制系统区别如下：

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| 集中式 | 将开发资源统一管理，一致性保持很容易做到。 | 必须联网才能工作，提交/取回受网络环境影响大。使用自己的电脑开发时必须先从中央服务器获取最新版本，然后修改，修改完后再推回中央服务器。 |
| 分布式 | 每个人的电脑都是一个完整的版本库，只有需要和别人同步的时候才需要联网。安全性也高，当一个节点宕掉后其他节点还有副本。 | 无法保证所有节点的版本内容完全一致，但是可以根据情况选择github或gitlab作为中心节点同步所有人的工作。 |

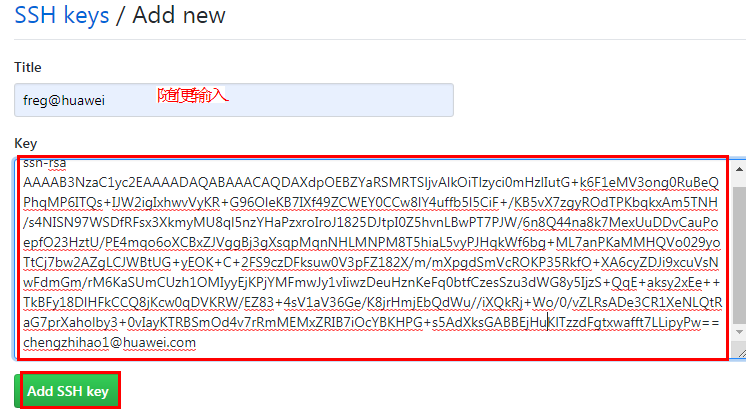
1. 安装后需要配置个人信息，例如姓名、email等，用于区分不同节点的修改提交

可以git config --global user.name "Your Name"，也可以直接去修改~/.gitconfig文件，如下：



1. 创建ssh秘钥。创建ssh秘钥主要用于和远端gitlab或github进行认证，认证之后每次提交/更新都不需要输入用户名和密码。
2. ssh-keygen -t rsa -C “YourEmail@hw.com”，之后会在/home/freg/.ssh生成id\_rsa.pub
3. 以github举例，选择“SSH and GPG keys”并添加新的ssh key，将id\_rsa.pub里面的内容拷贝至key中即可





1. 设置提交过滤。有些时候我们并不想把项目中的所有文件一并同步到git中，例如\*.pyc，\*.swp，\*~等临时文件，需要在git仓库下的.gitignore设置



1. 拉取和更新

使用git clone remote\_url拉取远程仓库

更新本地的remote branch，使得本地的remote branch和远程服务器保持一致，再同步本地分支和远程分支，使得两者内容保持一致（包括本地修改和远程修改内容）

git fetch origin

git rebase origin/master

上面两条指令和git pull等效，但推荐使用上面两条指令。git rebase合并分支发生冲突时提供的手段相对灵活。

查看远程仓库的地址：git remote -v

1. 推送

完成本地作业后，需要将本地修改推送到远程服务器，在推送之前需要更新。推送操作：

git add FILE // 修改的文件或文件夹

git rm [–r] FILE // 删除不需要的文件或文件夹

git status // 可省略，主要用于查看本次提交做的修改

git commit –m ‘modify operation’ // 本次推送的主要修改内容，给团队中别的人看

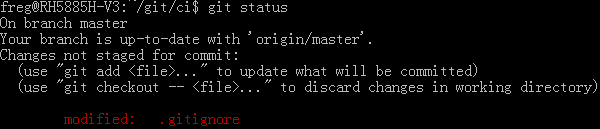
git push // 以上步骤只是更新本地分支，这个是更新远程仓库

1. 如何处理工作区和分支的不一致
2. 暂存区和工作区。在工作区对项目进行了修改，通过git add指令将修改从工作区同步至暂存区（stage），再通过git commit将暂存区内容同步至分支。

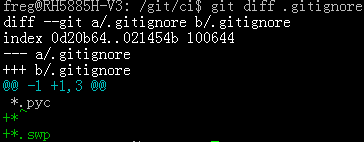


1. 在同步至暂存区之前（执行git add之前）发现问题，想要放弃修改

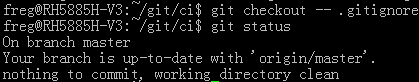
使用git status查看本地和远程仓库的差异，发现有一个叫.gitignore的文件被修改却没有同步。



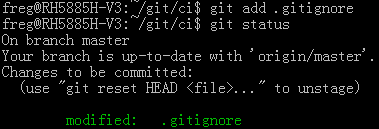
可以通过git diff详细查看修改内容（修改后文件和远程仓库对应文件的具体差异）



使用git checkout放弃本地修改。使用git checkout -- FILE后发现，本地修改被放弃，已完成和远程仓库同步。



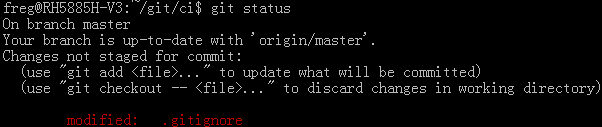
1. 已经同步至暂存区，在同步至分支之前（执行git commit之前）发现问题，想要放弃修改



使用git reset HEAD FILE撤销暂存区的修改，放弃本次commit



此时发现只有工作区的内容被修改

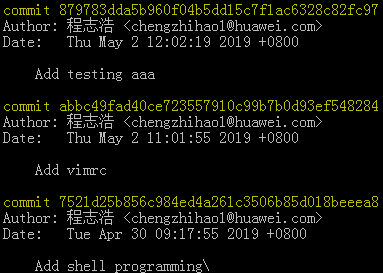


执行git checkout后发现，和分支内容已经同步



1. Git版本控制
2. 回退至历史某一版本

git log可以查到所有历史提交记录（根据commit内容可以对每个版本有个大致的了解，所以commit内容很重要，不要乱填）



同时可以使用git show COMMIT\_ID查看某次commit的具体修改

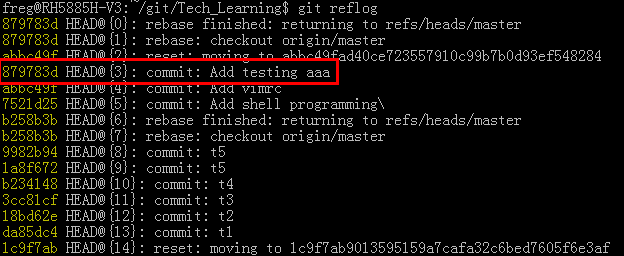


如果确定了对应版本的commit\_id，则可以使用git reset –hard COMMIT\_ID回退至某一版本，最后再git fetch origin更新一下。



1. 撤销回退版本

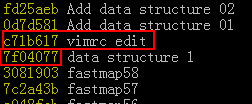
如果回退版本并关机后发现回退错了，还可以再回退会最开始的版本，使用git reflog可以查看git历史命令，找到对应的COMMIT\_ID，再git reset回退即可。





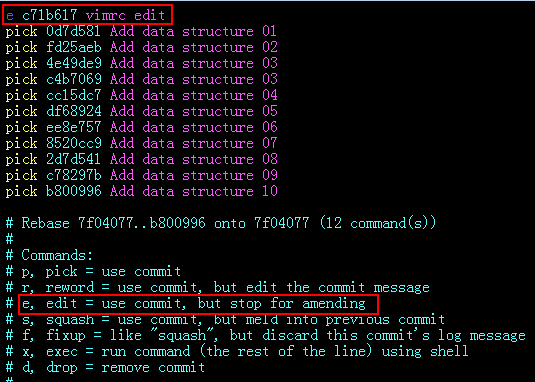
1. 修改历史某次提交

例如需要修改vimrc edit的commit信息。



首先rebase到需要修改的commit前一个commit ID，git rebase -i 7f04077

其次，根据提示我们修改对应commit标识为edit，保存后退出



然后我们可以修改vimrc或其他源文件，修改后git add，之后执行git commit -amend可以修改commit信息

最后使用git rebase --continue恢复之后的commit，一切就按修改意见恢复了，可以发现从修改的commit信息之后所有的commit号也发生了变化。



1. 分支管理

创建分支：git branch BRANCH\_NAME

查看分支：git branch

查看远程分支：git branch –r

创建并切换分支：git checkout –b BRANCH\_NAME

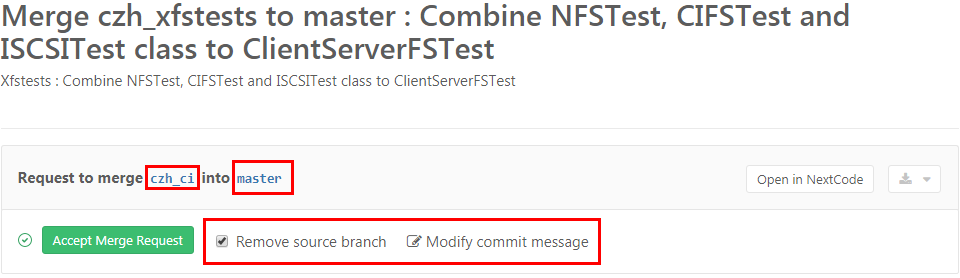
切换分支：git checkout BRANCH\_NAME

合并指定分支到当前分支：git merge BRANCH\_NAME

删除分支：git branch -d BRANCH\_NAME

将本地分支推送到远程分支：git push origin LOCAL\_BRANCH\_NAME

在github/gitlab界面选择推送上去的本地分支，点击“Merge Request”可以编辑合并到哪个分支、审核人和合并后是否删去分支，如图：



解决分支合并冲突：两个分支对同一个文件的同一行做了不同修改，使用git merge会报错，使用git status找出产生冲突的文件，git diff查看具体冲突内容。手动解决（修改对应文件冲突内容保持两个分支一致）冲突后合并。

使用git rebase合并分支，把远程仓库A(my-xfstests/master)的内容合并到远程仓库B(xfstests-dev/master)，然后再将合并后的仓库master分支推到远程仓库A中，my-xfstests是基于xfstests-dev某个版本拉出来自行维护的仓库。

1. 从仓库B克隆

$ git clone git://git.kernel.org/pub/scm/fs/xfs/xfstests-dev.git

1. 从仓库A克隆，并将其作为仓库B的一个分支

$ cd xfstests-dev # 进到仓库B的目录

$ git remote add my\_xfs git://my-xfstests.git # 添加远程仓库A的URL

$ git fetch my\_xfs # 获取远程仓库A的内容

$ git checkout -b my\_branch my\_xfs/master # 将仓库A的master分支作为仓库B的my\_branch分支

$ git checkout master # 切换回仓库B的分支



1. 合并仓库A（对应my\_branch）的所有commit到仓库B（对应master），期间可能有冲突，需要手动解决

$ git rebase my\_branch # 合并

# 如果发生冲突

$ git status # 查看冲突文件

$ git diff # 查看冲突内容

$ git add -u # 添加修改（用于解决冲突）

$ git rebase --continue # 继续合并

1. 将合并后的分支强行push到远程仓库A

$ git push -f my\_xfs # 远程仓库A对应的remote名字是my\_xfs

1. 保留工作现场
2. 应用场景：在自己分支下开发时遇到下游上报的紧急bug，需要将当前工作现场先保存，创建新分支去修复bug，新分支与master合并后删除新分支，最后恢复刚才的工作现场。
3. 使用方法：

git stash // 保留工作现场，此时git status是干净的

git checkout -b/git merge/git branch –d // 开新分支去修复bug，并合并删除新分支

git stash list // 寻找刚才的工作现场

git stash pop // 恢复工作现场并删除stash中的工作现场

git stash apply // 仅恢复工作现场，后续要删除

git stash drop // 删除工作现场

1. 生成补丁。Commit之后推送到远程仓库前希望生成patch，以便发给同事校验修改后决定要不要推远程仓库

git format-patch HEAD^ // 生成补丁



git apply --check PATCH // 检查补丁是否可以打入

git apply PATCH // 打入补丁，不包含commit内容

git am PATCH // 打入补丁，包含commit内容

1. 发送邮件

git format-patch -n11 --cover-letter --subject-prefix="PATCH Hulk-4.4-next" // 对于多个补丁要生成patch set，并指定统一的标题头

git send-email --suppress-cc=all --cc=linux-mtd@lists.infradead.org,linux-kernel@vger.kernel.org --to=david.oberhollenzer@sigma-star.at,richard@nod.at \*.patch --from chengzhihao1@huawei.com // 发送补丁邮件

1. 参考资料：

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>

<https://blog.csdn.net/qq_33829154/article/details/81364047>

<https://git-scm.com/book/zh/v2>

使用git rebase合并分支：<http://jartto.wang/2018/12/11/git-rebase/> <https://blog.csdn.net/wenph2008/article/details/44733239>