

git 的一些知识点

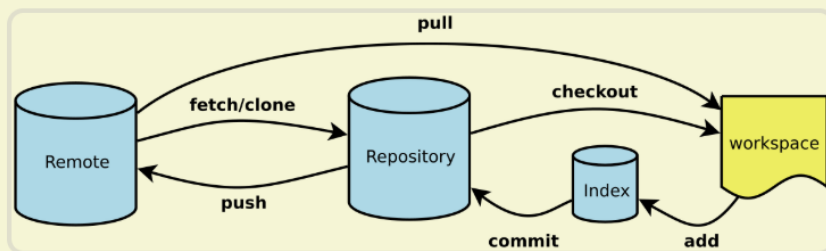
git 常用命令

参考资料: <https://www.ruanyifeng.com/blog/2015/12/git-cheat-sheet.html>

日期: 2015年12月 9日

我每天使用 Git, 但是很多命令记不住。

一般来说, 日常使用只要记住下图6个命令, 就可以了。但是熟练使用, 恐怕要记住60~100个命令。

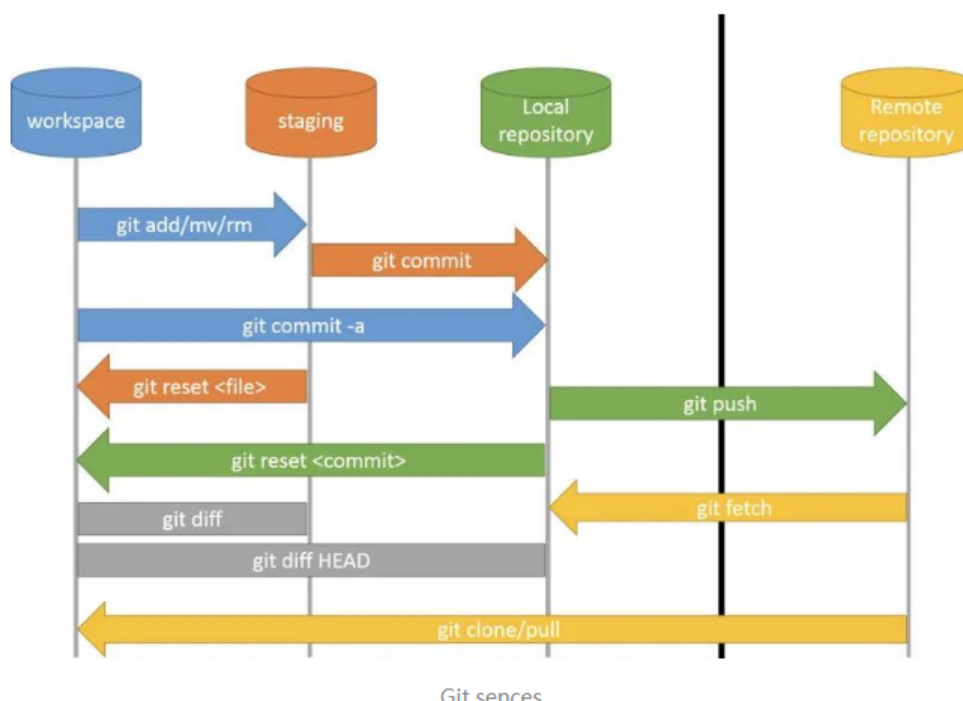


下面是我整理的常用 Git 命令清单。几个专用名词的译名如下。

- Workspace: 工作区
- Index / Stage: 暂存区
- Repository: 仓库区 (或本地仓库)
- Remote: 远程仓库

腾讯关于git的讲解, 感觉也不错: [原文地址](#)

Git 基础命令



初始化

git init 初始化本地仓库

- 下载项目
 - git clone 远程Git地址
- 带帐号密码克隆远端分支
 - git clone http://username:password@远端地址
 - username有特殊符号记得进行url编码
 - 远端地址不用带http
 - 实例
 - git clone
http://admin:6666666@192.168.1.249/front/H5/wimift-saas-h5.git
- 配置用户名
 - git config --global user.name "qinyuanqi"
- 配置邮箱
 - git config --global user.email qinyuanqiuse@gmail.com
- 显示当前的Git配置【git config】
 - git config --list
- 增加/删除文件【git add && git rm】
 - 工作区提交代码到暂缓区（只提交修改和新建的文件）
 - git add .
 - 工作区提交代码到暂缓区（只提交修改和删除的文件）
 - git add -u
 - 提交所有 文件
 - git add -A
 - 只把特定文件从暂存区删除, 文件会回退到工作区
 - git rm --cached fileName
 - 删除工作区文件，并且将这次删除放入暂存区
 - git rm [file1] [file2] ...
 - 改名文件，并且将这个改名放入暂存区
 - git mv [file-original] [file-renamed]

- 代码提交【git commit】
 - 提交暂存区到仓库区
 - `git commit -m [message]`
 - 已经通过git commit 提交到本地仓库，没有提交到远程，要修改上次的commit 信息
 - `git commit --amend -m [message]`
 - 只将所有被修改或者已删除的且已经被git管理的文档提交到仓库中
 - `git commit -a -m`
- 分支操作【git branch】
 - 列出所有本地分支
 - `git branch`
 - 列出所有远程分支
 - `git branch -r`
 - 列出所有本地分支和远程分支
 - `git branch -a -r`
 - 查看本地分支关联的远程分支的对应关系
 - `git branch -v`
 - 新建一个分支，但依然停留在当前分支
 - `git branch [branch-name]`
 - 新建一个分支，并切换到该分支
 - `git checkout -b [branch]`
 - 新建一个分支，指向指定commit
 - `git branch [branch] [commit]`
 - 新建一个分支，与指定的远程分支建立追踪关系
 - `git branch --track [branch] [remote-branch]`
 - 切换到指定分支，并更新工作区（直接把远程分支拉到本地）
 - 切换到上一个分支

- git checkout -
- 修改本地分支的名字
- git branch -m oldbranch newbranch
- 建立追踪关系，在现有分支与指定的远程分支之间
- git branch --set-upstream [branch] [remote-branch]
- 合并指定分支到当前分支
- git merge [branch]
- 选择一个commit，合并进当前分支
- git cherry-pick [commit]
- 删除本地分支（本地分支没有提交到远端）
- git branch -D 分支名
- 删除本地分支
- git branch -d 分支名
- 删除远程分支
- git push origin : [remote/branch]
- git branch -r -d origin/branch-name
- git push origin --delete [branch-name]
- git branch -dr [remote/branch]
- 本地存储代码【git stash】
 - 将所有代码缓存到存储区
 - git stash save "xxxxx"
 - 将存储区的第一个缓存取出来并从缓存区删除当前的
 - git stash pop
 - 将存储区把stash取出来，不删除当前的
 - git stash apply
 - 查看缓存区的所有stash
 - git stash list
 - 从存储区删除stash

- git stash drop

-



- 从stash创建分支

- git stash branch 新的分支名称

- 查看指定stash的diff

- 查看缩略的
 - git stash show
- 查看详细的
 - git stash show -p (或者--patch路径)

- 标签【git tag】

- 列出所有tag

- git tag

- 新建一个tag在当前commit tag 默认只再本地，要传到远程需要

- git tag [tag]

- 新建一个tag在指定commit

- git tag [tag] [commit]

- 删除本地tag

- git tag -d [tag]

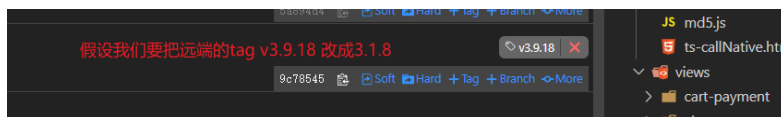
- 删除远程tag

- git push origin :refs/tags/[tagName]
- 必须先把远端分支删除再打标签，不然同名的分支使用此命令无法删除，会报错

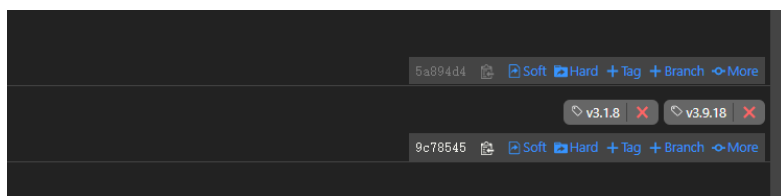
-

```
ASUS@DESKTOP-88HR5SV MINGW64 /f/company/route (master)
$ git push origin :hotfix/v4.0.2
error: dst refspec hotfix/v4.0.2 matches more than one
error: failed to push some refs to 'http://192.168.1.249/front/H5/route.git'
ASUS@DESKTOP-88HR5SV MINGW64 /f/company/route (master)
```

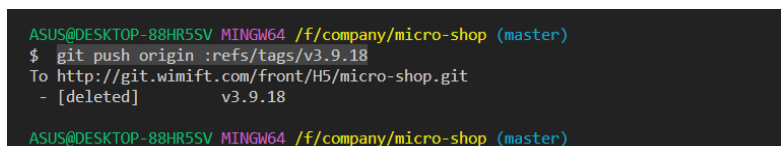
- 查看tag信息
 - `git show [tag]`
- 提交指定tag
 - `git push [remote] [tag]`
- 提交所有tag
 - `git push origin --tags`
 - `git push [remote] --tags`
- 新建一个分支，指向某个tag
 - `git checkout -b [branch] [tag]`
- tag错误，并且已经提交到远端的解决办法
 - 问题场景



- `git tag 新tag名称 旧tag名称`
 - `git tag v3.1.8 v3.9.18`
 - 生效了，本地存在两个tag



- 删除远端tag
 - `git push origin :refs/tags/远端tag名称`



- 执行完删除操作，会看到远端的tag已经被删除了，但是本地的还是会存在, 这里会有问题，假设别人的电脑也已经拉了代码，会怎么样？
 - `git tag -d v3.9.18`

- 查看信息【`git status`】

- 显示有变更的文件
 - `git status`
- 显示当前分支的版本历史
 - `git log`
- 搜索提交历史，根据关键词
 - `git log -S [keyword]`
- 显示今天你写了多少行代码
 - `git diff --shortstat "@{0 day ago}"`
- 远程同步【`git pull` && `git push`】
- 撤销和回滚代码
 - 恢复暂存区的指定文件到工作区
 - `git checkout [file]`
 - 恢复暂存区的所有文件到工作区
 - `git checkout .`
 - 回滚并且删除相关代码到上一次的更改
 - `git reset --hard HEAD`
 - 适用于需要强制把远端回滚到以前的某一个记录：回滚并且删除远端记录（按顺序执行下面的命令）
 1. `git reset --hard commit-id`
 2. `git push origin HEAD --force` 强制提交一次，之前提交记录就从远程仓库删除（配合 `git reset --hard commit-id` 使用）
 3. 请提醒别的同事，把回退的分支名称删除，不然会导致之前辛苦回退的代码，又被提交上来！！！！
 - 适用于已经commit 但是没有push的回滚，会把代码回滚到暂缓区
 - `git reset --soft HEAD^`
 - 回滚把代码还原到 modify 状态(暂缓区)
 - `git reset --soft commit-id`
 - 生成一个可供发布的压缩包

- git archive

git relog 回滚本地代码 (这个东西是本地的，意思就是只会记录当前你工作目录里面的所有git操作，如果你在A文件夹进行回滚，然后进B文件夹拉同样的项目，这个时候是看不到A文件里面的回滚操作的，这个要注意)

追梦人小豪 发布于 2020-05-25

背景

- 程序员 A 在本地进行了三次 commit 'demo1'、'demo2'、'demo3'
- 程序员 A 不小心进行了回滚 git reset --hard 'commit1',回滚到第一次提交
- 程序员 A 又修改了文件并进行了 commit, 'demo4'

问：如何找回被 reset 的两次 commit，并合并最新的一次 commit 'demo4'

使用 git log 查看，仅能看到 demo1 和 demo4 的提交记录

```
PS C:\Users\Administrator\Desktop\leetcode> git log
commit 25d44f9d71e5e86cde43352beac66e54d71d8
Author: zhongzhihao3996 <zhongzhihao@junhai.com>
Date: Mon May 25 11:25:36 2020 +0800

    demo4

commit abcc19526cc82376481d842b298698f81994c3d
Author: zhongzhihao3996 <zhongzhihao@junhai.com>
Date: Mon May 25 11:24:26 2020 +0800

    demo1
```

git relog

git relog 可以查看所有分支的所有操作记录（包括已经被删除的 commit 记录和 reset 的操作）

恢复步骤

- 具体说明：

- <https://blog.csdn.net/chaiyu2002/article/details/8177304>

1

```
git relog doesn't traverse HEAD's ancestry at all. The relog is an ordered list of the commits that HEAD has pointed to: it's undo history for your repo. The relog isn't part of the repo itself (it's stored now
in the .git directory).
7 # Note: understanding the relog means you can't really lose data from your repo once it's been committed. If you accidentally reset to an older commit, or release wrongly, or any other operation that visually "rem
上面说得很清楚了，也总结一下：
git log 是显示当前 HEAD 和它的祖先的提交，是包含当前提交的所有提交，父提交的提交，... 这样的原因。
git relog 是显示当前 HEAD 和它的祖先的提交，是包含当前提交的所有提交，父提交的提交，... 这样的原因。
git relog 是显示当前 HEAD 和它的祖先的提交，是包含当前提交的所有提交，父提交的提交，... 这样的原因。
relog 可以很好地帮助你找回被删除的数据，比如你错误地 reset 了一个提交的提交，或者 release，... 这个时候你可以使用 relog 查看在提交之前的信息，并且使用 git reset --hard 去重置。
下面再总结一下这个命令的用法：
先了解一下 git 的本地表示方法：
HEAD@{2} means "where HEAD used to be two moves ago", master@{one week ago} means "where master used to point to one week ago in this local repository"
```

- git relog 回滚本地代码

git 迁移仓库

镜像克隆：

- git clone --mirror https://github.com/./old.git
- cd old.git

然后推送镜像

- git remote set-url --push origin git@gitcafe.com/.../new.git
- git push --mirror

或者推送新建remote再推送：

- git remote add mirror origin git@gitcafe.com/.../new.git

- `git push mirror --all`
- `git push mirror --tags`
- 参考资料: <https://segmentfault.com/q/1010000006816225>

• 如何使用github

- <https://www.zhihu.com/question/20070065/answer/79557687>

• git配置别名

• git加速

• git 默认不区分文件夹和文件名大小写

• git 删除远程大写或者小写文件

• gitflow

• git删除某些文件的全部提交记录commit log

• git代码统计

• git 小技巧

• git ssh key 配置

• 1. 设置Git的user name和email: (如果是第一次的话)

- `git config --global user.name "qinyuanqi"`
- `git config --global user.email "qinyuanqiuse@gmail.com"`

• 2. 生成密钥

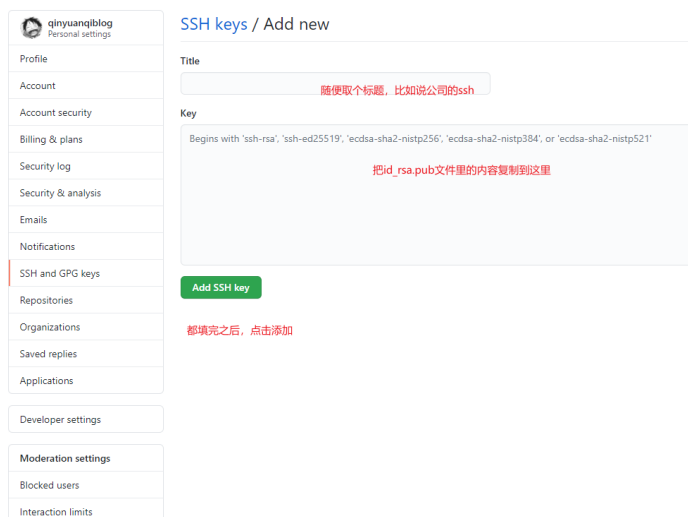
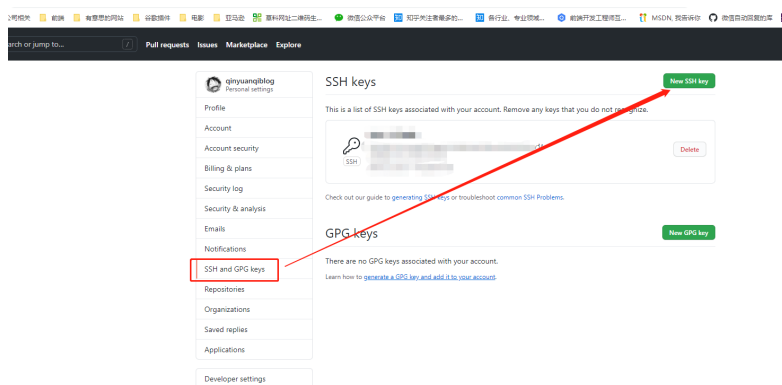
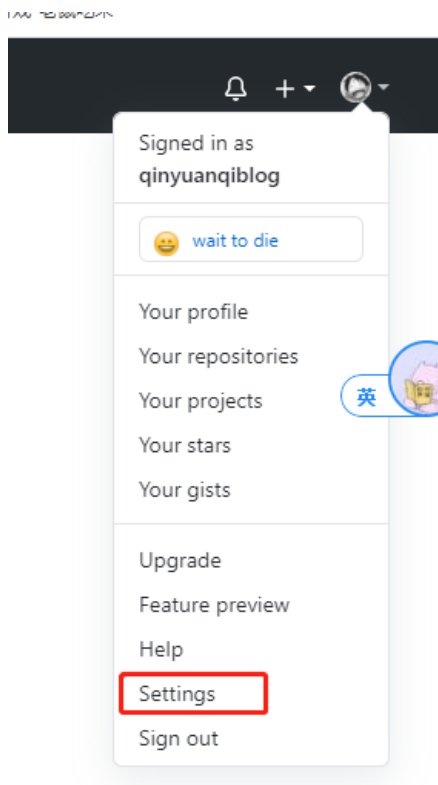
- `ssh-keygen -t rsa -C "qinyuanqiuse@gmail.com"`

• 3. 添加密钥到ssh-agent

- `eval "$(ssh-agent -s)"` 确保 ssh-agent 是可用的。
ssh-agent是一种控制用来保存公钥身份验证所使用的私钥的程序, 其实ssh-agent就是一个密钥管理器, 运行ssh-agent以后, 使用ssh-add将私钥交给ssh-agent保管, 其他程序需要身份验证的时候可以将验证申请交给ssh-agent来完成整个认证过程。
- `ssh-add ~/.ssh/id_rsa` 添加生成的 SSH key 到 ssh-agent。

• 4. 登陆Github, 添加 ssh

把id_rsa.pub文件里的内容复制到这里



5. 测试ssh是否配置成功

- `ssh -T git@github.com`

- 如果看到Hi后面是你的用户名，就说明成功了

你将会看到:

```
The authenticity of host 'github.com (207.97.227.239)' can't be
established.
RSA key fingerprint is '16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48'.
Are you sure you want to continue connecting (yes/no)?
选择 yes
Hi humingx You've successfully authenticated, but GitHub does not
provide shell access.
如果看到Hi后面是你的用户名，就说明成功了。
```

6. 修改.git文件夹下config中的url（如果你之前是http协议的话，才要改哦）

修改前

- `[remote "origin"] url =`
`https://github.com/humingx/humingx.github.io.git`
`fetch = +refs/heads/*:refs/remotes/origin/*`

修改后

- `[remote "origin"] url =`
`git@github.com:qinyuanqiblog/myExpress.git fetch =`
`+refs/heads/*:refs/remotes/origin/*`

解决github提交代码没有绿色格子问题

- git远程删除分支后，本地git branch -a 依然能看到的解决办法

- `git remote show origin`

- `git remote prune origin`

vscode 在线预览github项目

参考资料

- 一招 git clone 加速

- git技巧

- git删除某些文件的全部提交记录commit log

- git识别文件（夹）名大小写

- 配置别名

- 常用 Git 命令清单

- [git代码统计](#)