常用Git命令速查笔记

2017-01-02

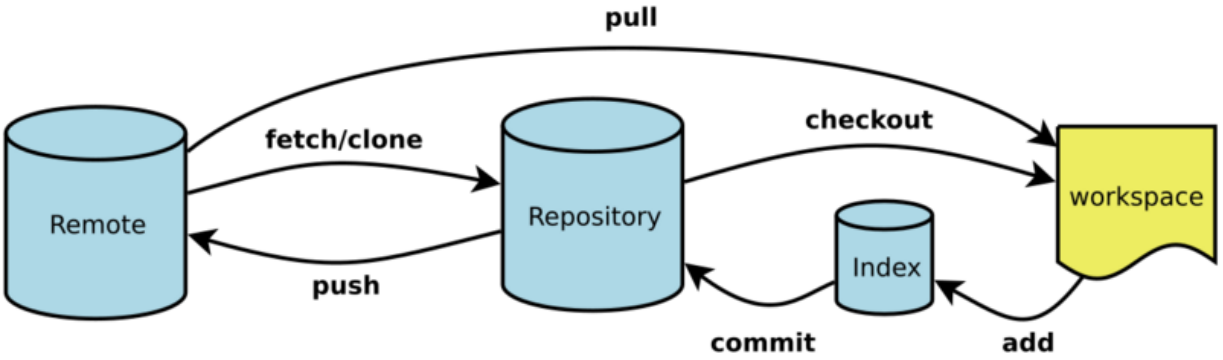
PROGRAMME

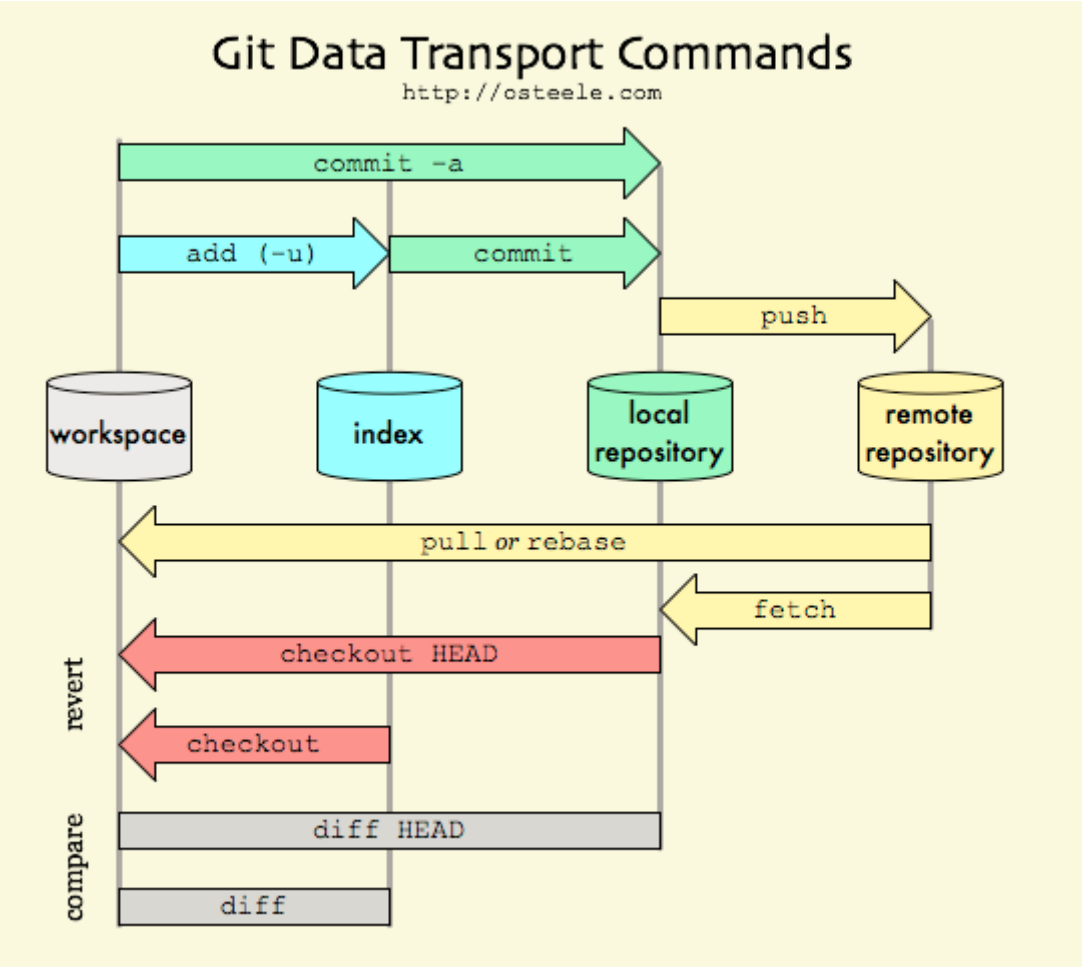
常用Git命令总结与速查笔记

参考文档: “Pro Git—Scott Chacon”

Git基础操作

几幅有助于理解GIT操作流程的示意图：





创建项目的Git仓库

从当前目录初始化

```
$ git init 将当前目录初始化为git仓库
$ git init [project-name] 新建一个目录，将其初始化为Git代码库
```

初始化后，在当前目录下会出现一个名为.git 的目录，所有Git需要的数据和资源都存放在这个目录中。

```
$ git add [file] 添加到暂存区
$ git commit -m 'comments' check in到本地仓库
$ git commit --amend -m 'comments' 补充上一次的commit comment(若没有文件的变更)
$ git commit --amend [file_name] 补充上一次的commit，提交新的文件的改动
```

从现有仓库clone

```
$ git clone git://github.com/schacon/grit.git 从现有仓库clone(本地，远程仓库同名)
$ git clone git://github.com/schacon/grit.git [rep_name] 本地一rep_name命名
$ git clone -b [branch_name] git://github.com/schacon/grit.git clone指定branch_name名的分支，默认为master分支
```

查看状态

```
$ git status 显示有变更的文件
```

查看diff

三种diff的查看方式:

```
$ git diff 查看当前文件（工作区）与暂存区的差异
$ git diff --cached 查看暂存文件与上次提交的差异
$ git diff HEAD 查看工作区与HEAD的差异
```

跳过使用暂存区域

```
$ git commit -a -m 'comments'
```

移除文件

```
$ git rm [file] 从仓库移除，同时也从本地文件夹删除
$ git rm --cached [file] 从仓库移除，但不从本地文件夹删除
```

移动/重命名文件

```
$ git mv [file_from] [file_to]
```

查看提交历史

```
整个工程级的查看:
$ git log 查看提交历史
$ git log --stat 显示历史提交信息及相关变更的文件
$ git log -p 查看提交内容diff
$ git log -p -2 查看提交内容diff， 显示2个
```

```
$ git log --author='Roy Luo' 查看指定作者名
$ git log --pretty=oneline 结果显示一行
文件级的查看:
$ git log [file_name] 查看指定文件的提交历史
$ git log -p [file_name] 查看制定文件的提交历史+diff信息

查看指定commit id:
$ git show [commit-id]
example:
$ git show 541ff6e2f886cc79d0396f78e5520acb108a6287
$ git show 541ff
$ git show HEAD 最近一次的commit
$ git show HEAD^
$ git show HEAD^^
$ git show HEAD~4
```

撤销操作

修改最后一次提交

```
$ git add [forgotten_file] 补上暂存操作
$ git commit --amend 运行 --amend 提交
```

撤销commit操作，保留working tree 和 index file

回退到git commit之前

```
$ git reset --soft HEAD^
```

彻底撤销commit操作,不保留working tree 和 index file

撤销销毁最近一次的commit

```
$ git reset --hard HEAD^
```

撤销暂存区的文件（即撤销已经git add操作）

```
$ git reset HEAD <file>
```

从committed状态回退到git add之前

```
$ git reset --mixed HEAD^
```

取消对文件的修改

```
$ git checkout -- <file>
```

在历史版本之间切换

```
$ git reflog 查看命令历史， 可以显示
历史commit id
$ git reset --hard [commit_id]
```

Git 分支

查看分支信息

```
```

For Example:

```
$ git branch
* master
$ git branch -r
origin/HEAD -> origin/master
origin/master #三个远程分支
origin/sanscoutv1
origin/sanscoutv2_filterv1
$ git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/master
remotes/origin/sanscoutv1
remotes/origin/sanscoutv2_filterv1
```

新建分支

```
$ git branch [branch-name] 新建一个分支，但依然停留在当前分支
$ git checkout -b [branch] 新建一个分支, 并切换到该分支
```

切换分支

```
$ git checkout [branch-name] 切换到指定分支
```

合并分支

```
$ git merge [branch-name] 合并指定分支到当前分支
```

删除分支

```
$ git branch -d [branch-name] 删除指定分支
```

解决冲突

```
$ git status 查看文件冲突信息
$ git add [file_name] 将冲突文件标记为解决
$ git mergetool 使用可视化的合并工具
```

更新分支

rebase: 拉取更新并合并到本地分支，本地分支的更改作为新的commit重新合入.

```
$ git pull --rebase 默认为当前分支与与之关联的远程分支
$ git rebase master 从本地分支master rebase到当前分支
若有冲突，解决完后：
$ git rebase --continue
```

merge操作会生成一个新的节点，之前的提交分开显示。而rebase操作不会生成新的节点，是将两个分支融合成一个线性的提交.

git cherry-pick

```
$ git cherry-pick [commit-id] 从其它分支的指定commit id合入到当前分支
```

远程仓库的使用

显示远程仓库

clone时，所使用的remote自动被Git命名为**origin**;

若用其它名称，则使用git clone -o 选项；

```
$ git remote 显示仓库名
$ git remote -v 显示详细信息
$ git remote show [remote_name] 显示指定remote的信息
```

For Example:

```
$ git remote
origin
$ git remote -v
origin git://github.com/schacon/grit.git (fetch)
origin git://github.com/schacon/grit.git (push)
$ git remote show origin
* remote origin
 Fetch URL: git://github.com/schacon/grit.git
 Push URL: git://github.com/schacon/grit.git
 HEAD branch: master
 Remote branches:
 dev tracked
 master tracked
 Local branch configured for 'git pull':
 master merges with remote master
 Local ref configured for 'git push':
 master pushes to master (local out of date)
```

添加远程仓库

```
$ git remote add [remote-name] [url] 添加url指定的远程仓库到本地，命名为remote-name
$ git remote rm [remote-name] 删除
$ git remote rename [old-remote-name] [new-remote-name] 改名
```

从远程仓库update数据

git fetch

```
$ git fetch [remote-name]
```

此命令会到远程仓库中拉取所有你本地仓库中还没有的数据.

需要记住，fetch 命令只是将远端的数据拉到本地仓库，并不自动合并到当前工作分支，只有当你确实准备好了，才能手工合并(git merge)

For example:

```
$ git status #git fetch [remote-name] 后的本地仓库的状态
Your branch is up-to-date with 'origin/master'. 表示当前本地分支与远程分支一致
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded. 表示当前本地分支落后远程分支一个commit， 即远程分支有更新；
```

默认情况下，git fetch取回所有分支（branch）的更新。如果只想取回特定分支的更新，可以指定分支名。

```
$ git fetch [remote_name] [branch_name]
example:
$ git fetch origin sanscoutv1
```

所取回的更新，在本地主机上要用”remote_name/branch_name”的形式读取。比如origin主机的master，就要用origin/master读取, origin/sanscoutv1;

当然，git fetch取回的信息并不会merge进本地仓库，所以git fetch是比较安全的；但是git fetch取回来的信息到底存放在哪里了呢，其实git fetch取回的只是远程commit信息，包括commit id等，存放在git配置文件关于远程仓库管理信息中，所以使用git branch -r或者git status命令看到的即是 配置信息中的信息。如果要真正读取实际内容的话必须要创建新的分支来关联；

```
$ git checkout -b [branch_name] [remote_branch_name]
example:
$ git checkout -b newBranch origin/sanscoutv1
```

可以使用如下的命令一键搞定：

```
$ git fetch [remote-name] [remote-branch-name/local-new-branch-name]
example:
$ git fetch origin sanscoutv1:local_newBranch
```

现在本地的分支与远程分支关联在了一起，可以进行后续的merge，或者直接merge：

```
$ git merge origin/master
```

git pull

git pull的主要作用是将远程分支的更新取回并merge到指定的本地分支,而fetch并不进行merge；

```
$ git pull [remote] [remote-branch]:[local-branch] 取回远程仓库的更新，并与本地分支合并
$ git pull [remote] [remote-branch] 默认与当前本地分支进行merge
example:
$ git pull origin sanscoutv1:local_sanscoutv1
$ git pull origin master
```

推送数据到远程仓库

```
$ git push [remote-name] [local-branch-name]:[remote-branch-name] 上传本地指定分支到远程仓库
example:
$ git push origin master:master
$ git push origin master 本地master分支与远程master分支存在关联
```

```
$ git push origin master 本地master分支与远程master分支存在关联
```

跟踪远程分支

将远程分支与本地分支关联

```
$ git checkout -b [分支名] [远程名]/[分支名]
```

删除远程分支

```
$ git push [remote-name] :[remote-branch-name]
example:
$ git push origin :sanscoutv1 删除远程分支sanscoutv1
```

标签操作

列显已有的标签

```
$ git tag
```

新建标签

含附注的标签

```
$ git tag -a [tag_name] -m 'comments'
```

轻量级标签

```
$ git tag [tag_name]
```

查看标签信息

```
$ git show [tag_name]
```

推送标签到远程仓库

```
$ git push [remote] [tag_name]
```

新建一个分支，指向某个tag

```
git checkout -b [branch]
[tag_name]
```

打标签

git tag -a v1.01 -m "Relase version 1.01"

注解：git tag 是打标签的命令，-a 是添加标签，其后要跟新标签号，-m 及后面的字符串是对该标签的注释。

提交标签到远程仓库

git push origin -tags

注解：就像git push origin master 把本地修改提交到远程仓库一样，-tags可以把本地的打的标签全部提交到远程仓库。

删除标签

git tag -d v1.01

注解：-d 表示删除，后面跟要删除的tag名字

删除远程标签

git push origin :refs/tags/v1.01

注解：就像git push origin :branch_1 可以删除远程仓库的分支branch_1一样， 冒号前为空表示删除远程仓库的tag。

查看标签

git tag