git 使用笔记

作者：pleaseok

原文链接：https://ld246.com/article/1544358492005
来源网站：链滴

git使用笔记~(watch:liaoxuefeng.com)~

创建版本库

1.创建一个目录，用来存项目

```
$ mkdir projectGit
```

2.git版本控制器上切换到该目录

```
$ cd projectGit
```

3.通过get init命令把这个目录变成Git可以管理的仓库

```
$ git init
Initialized empty Git repository in /YourDir/projectGit/.git/
```

4.因为所有版本控制系统都是只能跟踪文本文件的改动的，所以在projectGit目录下创建一个readme.txt，内容随意

5.使用git add命令把文件添加到仓库

```
$ git add readme.txt
```

6.使用git commit把文件提交到仓库

```
$ git commit -m "wrote a readme file"
balabla一段提示:几个文件被修改，文件内容修改情况
```

修改文件再提交

1.打开readme.txt，把里面的内容随意修改或者增加几个字符

2.运行git status命令看看当前仓库状态

```
$ git status
```

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

文件表示readme.txt被修改过了，但还没有准备提交的修改

3.可以使用git diff来查看被修改的地方,diff->difference

```
$ git diff readme.txt
```

4.把文件重新提交到仓库

```
$ git add readme.txt
```

```
$ git commmit -m "midif file"
```

5.再次使用git status查看仓库当前状态

```
$ git status
```

```
On branch master
nothing to commit, working tree clean
```

版本回退

1.用git log命令查看历史记录

```
$ git log
```

```
commit 1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master)
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Fri May 18 21:06:15 2018 +0800

    append GPL

commit e475afc93c209a690c39c13a46716e8fa000c366
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Fri May 18 21:03:36 2018 +0800

    add distributed
```

以上显示最近一次提交时的备注是'append GPL'，上一次为'add distributed'。·1094ad..·为commit d，这个id也是有用的

2.使用git reset命令回退上一个版本

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

```
HEAD is now at e475afc add distributed
```

HEAD代表此版本，上一个版本就是HEAD^,上上个版本就是HEAD^^,上一百个版本HEAD~100

3.当回退后，再使用git log命令，则“append GPL”已经看不到了。如果回退后反悔了，想要重新到原来的状态，则需要使用下面的命令

```
$ git reset --hard 1094a
```

```
HEAD is now at 83b0afe append GPL
```

版本号没必要写全，前几位能被区别就可以了，Git会自动去找。

4.如果不记得commit id,则可以使用git reflog命令，它的作用是用来记录每一次的命令

```
$ git reflog
```

```
e475afc HEAD@{1}: reset: moving to HEAD^
```

```
1094adb (HEAD -> master) HEAD@{2}: commit: append GPL
```

```
e475afc HEAD@{3}: commit: add distributed
```

创建与合并分支

1.创建dev分支，然后再切换到dev分支

```
$ git checkout -b dev
Switched to a new branch 'dev'
```

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

```
$ git branch dev
```

```
$ git checkout dev
Switched to a new branch 'dev'
```

2.用git branch命令查看当前分支

```
$ git branch
```

```
* dev
  master
```

git branch命令会列出所有分支，当前分支会标一个*号

3.然后，我们就可以在dev分支上做开发了。合并分支的话可以使用git merge命令。首先切换回master分支上。

```
$ git checkout master
Switched to branch 'master'
```

```
$ git merge dev
Updating d46f35e..b17d20e
Fast-forward
readme.txt | 1 +
1 file changed, 1 insertion(+)
```

注意到上面的Fast-forward信息，Git告诉我们，这次合并是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快。

4.合并完成后，就可以放心地删除dev分支了：

```
$ git branch -d dev
Deleted branch dev (was b17d20e).
```

解决冲突

1.当我们在dev分支上对某一个文件修改后，某人在master分支同一个文件同一个地方或不同的地方另外的修改后，这时合并两个分支将会有冲突，如下：

```
$ git merge feature1
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result
```

2.另外git status命令也可以告诉我们冲突的文件

```
$ git status
```

```
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

3.打开文件可提示如下内容:

```
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files.
<<<<<<< HEAD
Creating a new branch is quick & simple.
=======
Creating a new branch is quick AND simple.
>>>>>>> feature1
```

Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容

4.把文件修改后再提交则可解决冲突，用git log --graph命令可以看到分支合并图。

```
$ git log --graph --pretty=oneline --abbrev-commit
```

```
*   cf810e4 (HEAD -> master) conflict fixed
|\
| * 14096d0 (feature1) AND simple
* | 5dc6824 & simple
|/
* b17d20e branch test
* d46f35e (origin/master) remove test.txt
* b84166e add test.txt
* 519219b git tracks changes
* e43a48b understand how stage works
* 1094adb append GPL
* e475afc add distributed
* eaadf4e wrote a readme file
```

多人协作

推送分支

1.查看远程库的信息

```
$ git remote
origin
```

2.显示远程库更详细的信息

```
$ git remote -v
origin  https://github.com/jishuzcn/learngit.git (fetch)
origin  https://github.com/jishuzcn/learngit.git (push)
```

上面显示了可以抓取和推送的origin的地址

3.把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git就会把该分支推送远程库对应的远程分支上

```
$ git push origin master
```

推送dev分支

```
$ git push origin dev
```

在Git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！

抓取分支

1.在另一个目录下克隆

```
$ git clone https://github.com/jishuzcn/learngit.git
Cloning into 'learngit'...
remote: Counting objects: 40, done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 40 (delta 14), reused 40 (delta 14), pack-reused 0
Receiving objects: 100% (40/40), done.
Resolving deltas: 100% (14/14), done.
```

当从远程库clone时，默认情况下，只能看到本地的master分支。这是需要使用如下命令切换到dev支上开发

```
$ git checkout -b dev origin/dev
```

2.两个不同目录推送提交，模拟冲突

```
$ git add env.txt
$ git commit -m "add new env"
$ git push origin dev
```

当第二个人推送时会提示如下信息:

```
To https://github.com/jishuzcn/learngit.git
 ! [rejected]        dev -> dev (non-fast-forward)
error: failed to push some refs to 'https://github.com/jishuzcn/learngit.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

3.解决办法：先用git pull把最新的提交从origin/dev抓下来，然后，在本地合并，解决冲突，再推送

指定本地dev分支与远程origin/dev分支的链接

```
$ git branch --set-upstream-to=origin/dev dev
Branch 'dev' set up to track remote branch 'dev' from 'origin'.
```

pull

```
$ git pull
Auto-merging env.txt
CONFLICT (add/add): Merge conflict in env.txt
Automatic merge failed; fix conflicts and then commit the result.
```

解决冲突后再提交push

```
$ git commit -m "fixenv conflict"
On branch dev
Your branch is ahead of 'origin/dev' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

$ git push origin dev
Enumerating objects: 2, done.
Counting objects: 100% (2/2), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 341 bytes | 341.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/jishuzcn/learngit.git
   b96fc94..3d7fa8c  dev -> dev
```

因此，多人协作的工作模式通常是这样：

1.首先，可以试图用git push origin <branch-name>推送自己的修改；

2.如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；

3.如果合并有冲突，则解决冲突，并在本地提交；

4.没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令it branch --set-upstream-to <branch-name> origin/<branch-name>。

这就是多人协作的工作模式，一旦熟悉了，就非常简单