#初始化git仓库

git init

#将仓库中所有内容添加到缓冲区

git add \*

#将缓冲区内容提交到版本库，引号内为版本信息

git commit -m 'my object'

#查看版本状态

git status

#将修改后的文件添加到缓冲区

git add test.txt

git status

#将缓冲区内容删除

git reset HEAD test.txt

git status

git add test.txt

git status

#比较目前文件的修改状态

git diff test.txt

#查看提交过的版本，以便进行回滚等操作

git log

#单行方式查看提交过的版本

git log --pretty=oneline

#回退到上一版本

git reset --hard HEAD^

#回退到上上版本

git reset --hard HEAD^^

#指定编码回退版本

git reset --hard eb272aa

#如果已回退，又想恢复，忘了版本号，用该命令查询

git reflog

#如果仅仅修改了本地文件，没有被提交到缓冲区用该命令进行还原

git checkout -- test.txt

#将远程github上的版本库通本地进行关联，冒号后面为github上

#的用户名和版本库，这是建立在将本地公钥添加到github的前提下

#origin2为定义的名称，可以自定义

git remote add origin2 git@github.com:xingmin860818/mypython.git

#将本地建立的版本库与github同步,该步骤要进入本地的版本库中执行

#master为命令一部分，非自定义,本地的版本库中必须存在已提交过的

#文件

git push -u origin2 master

#经过上面的第一次推送后，今后有推送内容可直接执行以下命令

git push origin2 master

#团队从0开发，就需要先建立远程库，然后各个开发端克隆该库

git clone git@github.com:xingmin860818/mypython.git

git remote add origin2 git@github.com:xingmin860818/mypython.git

修改，本地commit，推送远端

git push -u origin2 master

#分支管理

分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作

创建并切换到分支

[root@ansible-agent git]# git checkout -b dev

Switched to a new branch 'dev'

相当于：

[root@ansible-agent git]# git branch dev

[root@ansible-agent git]# git checkout dev

查看分支：

[root@ansible-agent git]# git branch （省略\*）

\* dev

master

分支中修改提交文件readme.txt

vim readme.txt

git add readme.txt

git commit -m 'branch test'

[dev fec145a] branch test

1 file changed, 1 insertion(+)

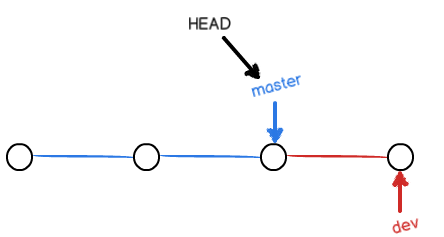
Dev分支的工作完成，切换回master分支

git checkout master

Switched to branch 'master'

ls

发现刚才便器的内容不见了，那是因为刚刚提交的是在Dev分支上的，而master分支此刻的提交点没有变



现在吧dev分支id工作合并到master分支上：

git merge dev

Updating c465d28..27a1a27

Fast-forward

readme.txt | 2 +-

1 files changed, 1 insertions(+), 1 deletions(-cat readme.txt

这是在看readme.txt的内容就更新了

注意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的当前指向dev的当前提交，所以合并速度非常快。当然也不是每次合并都能Fast-forward。

合并完成后就可以放心的删除dev分支了：

#git branch -d dev

Deleted branch dev (was 27a1a27).

#git branch

\* master

解决冲突：

下面在创建新分支，新分支更改文件后，切换回老分支再更改文件，然后合并两个分支，就会出现冲突问题

[root@ansible-agent git]# git checkout -b featurel

Switched to a new branch 'featurel'

[root@ansible-agent git]# ls

abc readme.txt

[root@ansible-agent git]# vim readme.txt

[root@ansible-agent git]# git add readme.txt

[root@ansible-agent git]# git commit -m 'and simple’

[featurel c4fa083] and simple

Committer: root <root@ansible-agent.(none)>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly:

git config --global user.name "Your Name"

git config --global user.email you@example.com

If the identity used for this commit is wrong, you can fix it with:

git commit --amend --author='Your Name <you@example.com>'

1 files changed, 1 insertions(+), 0 deletions(-)

[root@ansible-agent git]# git checkout master

Switched to branch 'master'

[root@ansible-agent git]# vim readme.txt

[root@ansible-agent git]# git add readme.txt

[root@ansible-agent git]# git commit -m 'simpol'

[master 6ed2f44] simpol

Committer: root <root@ansible-agent.(none)>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly:

git config --global user.name "Your Name"

git config --global user.email you@example.com

If the identity used for this commit is wrong, you can fix it with:

git commit --amend --author='Your Name <you@example.com>'

1 files changed, 1 insertions(+), 0 deletions(-)

[root@ansible-agent git]# git merge featurel

Auto-merging readme.txt

CONFLICT (content): Merge conflict in readme.txt

Automatic merge failed; fix conflicts and then commit the result.

[root@ansible-agent git]# git status

# On branch master

# Unmerged paths:

# (use "git add/rm <file>..." as appropriate to mark resolution)

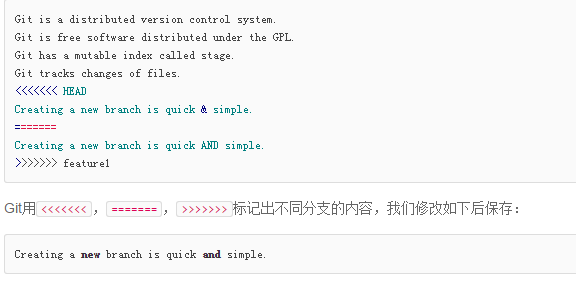
#

# both modified: readme.txt

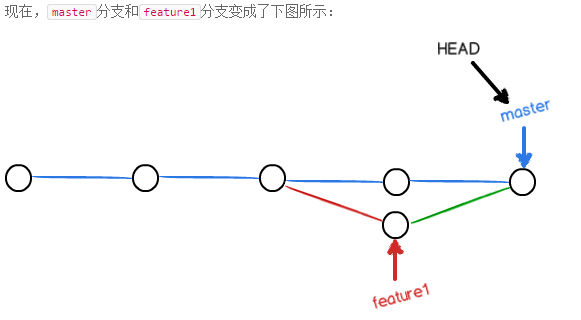
#

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

查看readme.txt的内容



在提交：



用带参数的git log也可以看到分支的合并情况：

[root@ansible-agent git]# git log --graph --pretty=oneline --abbrev-commit

\* abc816e conflit fixed

|\

| \* c4fa083 and simple

\* | 6ed2f44 simpol

|/

\* 27a1a27 b

\* c465d28 branch test

\* 02e6f05 msg

最后可以删除新建的分支

[root@ansible-agent git]# git branch -d featurel

Deleted branch featurel (was c4fa083).

分支管理策略

通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

下面我们实战一下--no-ff方式的git merge：

首先，仍然创建并切换dev分支

[root@ansible-agent git]# git checkout -b dev

'Switched to a new branch 'dev'

[root@ansible-agent git]# vim readme.txt

[root@ansible-agent git]# cat readme.txt

Creating a new branch is quick

ksksksksk

ooooooooo

dddddddddddddeeeeeeeeeeeeeeeeevvvvvvvvvvvvvvvvv 新增内容

[root@ansible-agent git]# git add readme.txt

[root@ansible-agent git]# git commit -m 'add merge'

[dev 86d6cf0] add merge

[root@ansible-agent git]# git checkout master

Switched to branch 'master'

准备合并dev分支请注意—no-ff参数，表示禁用Fast-forward

[root@ansible-agent git]# git merge --no-ff -m 'merge with no-ff' dev

Merge made by recursive.

readme.txt | 1 +

1 files changed, 1 insertions(+), 0 deletions(-)

因为本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去，合并后用git log看看分支历史

[root@ansible-agent git]# git log --graph --pretty=oneline --abbrev-commit

\* 7bca23f merge with no-ff

|\

| \* 86d6cf0 add merge

|/

\* abc816e conflit fixed

|\

| \* c4fa083 and simple

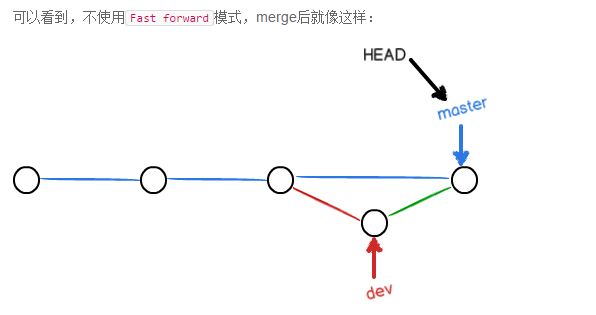
\* | 6ed2f44 simpol

|/

\* 27a1a27 b

\* c465d28 branch test

\* 02e6f05 msg



[http://img.alicdn.com/tps/TB1DmcoJXXXXXavXpXXXXXXXXXX-26-26.png](http://pro.taobao.com/index.htm?spm=a2320.7388781.1998051528.5.JOvNkN)[http://img.alicdn.com/tps/TB1rRQpJXXXXXcvXXXXXXXXXXXX-40-26.png](javascript:;)

分支管理策略

阅读: 73909

通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

下面我们实战一下--no-ff方式的git merge：

首先，仍然创建并切换dev分支：

$ git checkout -b dev

Switched to a new branch 'dev'

修改readme.txt文件，并提交一个新的commit：

$ git add readme.txt

$ git **commit** -m "add merge"

[dev 6224937] **add** merge

1 file changed, 1 insertion(+)

现在，我们切换回master：

$ git checkout master

Switched to branch 'master'

准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward：

$ git merge --no-ff -m "merge with no-ff" dev

Merge made by the 'recursive' strategy.

readme.txt | 1 +

1 file changed, 1 insertion(+)

因为本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去。

合并后，我们用git log看看分支历史：

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

\* 7825a50 merge **with** no-ff

|\

| \* 6224937 add merge

|/

\* 59bc1cb conflict fixed

...

可以看到，不使用Fast forward模式，merge后就像这样：



分支策略

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



小结

Git分支十分强大，在团队开发中应该充分应用。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

bug管理

软件开发中，bug就像家常便饭一样。有了bug就需要修复，在Git中，由于分支是如此的强大，所以，每个bug都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除。

当你接到一个修复一个代号101的bug的任务时，很自然地，你想创建一个分支issue-101来修复它，但是，等等，当前正在dev上进行的工作还没有提交：

$ git status

*# On branch dev*

*# Changes to be committed:*

*# (use "git reset HEAD <file>..." to unstage)*

*#*

*# new file: hello.py*

*#*

*# 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*

*#*

并不是你不想提交，而是工作只进行到一半，还没法提交，预计完成还需1天时间。但是，必须在两个小时内修复该bug，怎么办？

幸好，Git还提供了一个stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作：

[root@ansible-agent git]# git stash

Saved working directory and index state WIP on dev: 86d6cf0 add merge

HEAD is now at 86d6cf0 add merge

[root@ansible-agent git]# git status

# On branch dev

nothing to commit (working directory clean)

再用git status 查看工作区，就是干净的了，这是可以放心的创建分支来修复bug

[root@ansible-agent git]# git checkout master

Switched to branch 'master'

[root@ansible-agent git]# git checkout -b issue-101

Switched to a new branch 'issue-101'

现在修复bug，然后提交

[root@ansible-agent git]# vim readme.txt

[root@ansible-agent git]# git add readme.txt

[root@ansible-agent git]# git commit -m 'fix bug 101'

[issue-101 6a7b0b9] fix bug 101

Committer: root <root@ansible-agent.(none)>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly:

git config --global user.name "Your Name"

git config --global user.email you@example.com

If the identity used for this commit is wrong, you can fix it with:

git commit --amend --author='Your Name <you@example.com>'

1 files changed, 1 insertions(+), 1 deletions(-)

[root@ansible-agent git]# git checkout master

Switched to branch 'master'

[root@ansible-agent git]# git merge --no-ff -m 'merged bug fix 101' issue-101

Merge made by recursive.

readme.txt | 2 +-

1 files changed, 1 insertions(+), 1 deletions(-)

[root@ansible-agent git]# git branch -d issue-101

Deleted branch issue-101 (was 6a7b0b9).

工作区是干净的，刚才的工作现场保存到哪里去了？用git stash list命令看看：

[root@ansible-agent git]# git stash list

stash@{0}: WIP on dev: 86d6cf0 add merge

工作现场还在，git吧stash内容存在某个地方了，现在要恢复一下，有两个办法：

一是用git stash apply恢复，但是回复后，stash内容并不删除，你需要用git stash drop来删除；二是用git stash pop,恢复是吧stash内容也删除了：

[root@ansible-agent git]# git stash pop

Auto-merging readme.txt

CONFLICT (content): Merge conflict in readme.txt

再用git stash list查看就看不到任何stash内容了。

可以多次stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令

git stash apply stash@{0}

小结:

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场。

Feature 分支

软件开发中，总有无穷无尽的新的功能要不断添加进来。

添加一个新功能时，你肯定不希望因为一些实验性质的代码，把主分支搞乱了，所以，每添加一个新功能，最好新建一个feature分支，在上面开发，完成后，合并，最后，删除该feature分支。

现在，你终于接到了一个新任务：开发代号为Vulcan的新功能，该功能计划用于下一代星际飞船。

于是准备开发：

[root@ansible-agent git]# git checkout -b feature-vulcan

Switched to a new branch 'feature-vulcan'

开始开发:

[root@ansible-agent git]# vim vulcan.py

开发完毕后

[root@ansible-agent git]# git add vulcan.py

[root@ansible-agent git]# git status

# On branch feature-vulcan

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: vulcan.py

#

[root@ansible-agent git]# git commit -m 'add feature vulcan'

[feature-vulcan a0c68da] add feature vulcan

Committer: root <root@ansible-agent.(none)>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly:

git config --global user.name "Your Name"

git config --global user.email you@example.com

If the identity used for this commit is wrong, you can fix it with:

git commit --amend --author='Your Name <you@example.com>'

1 files changed, 1 insertions(+), 0 deletions(-)

create mode 100644 vulcan.py

切回dev

[root@ansible-agent git]# git checkout dev

Switched to branch 'dev'

一切顺利的话，feature分支和bug分支是类似的，合并然后删除

但是，

此时接到上级命令，新功能必须取消，要销毁这个分支

[root@ansible-agent git]# git branch -d feature-vulcan

error: The branch 'feature-vulcan' is not fully merged.

If you are sure you want to delete it, run 'git branch -D feature-vulcan'.

提示销毁失败，如果确定要删除，用后面的命令强行删除：

[root@ansible-agent git]# git branch -D feature-vulcan

Deleted branch feature-vulcan (was a0c68da).

多人协作:

当从远程参股克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且远程仓库的默认名称是origin。

要查看远程库的信息用git remote：

[root@ansible-agent git]# git remote

origin

显示更详细的信息

[root@ansible-agent git]# git remote -v

origin git@github.com:xingmin860818/test.git (fetch)

origin git@github.com:xingmin860818/test.git (push)

推送分支

推送分支，就是把该分支上所有的本地提交推送到远程库，推送是，要指定本地分支，这样，git就会把该分支推送到远程库对应的分支上

[root@ansible-agent git]# git push origin3 master

Counting objects: 28, done.

Compressing objects: 100% (23/23), done.

Writing objects: 100% (28/28), 2.34 KiB, done.

Total 28 (delta 8), reused 0 (delta 0)

To git@github.com:xingmin860818/git.git

\* [new branch] master -> master

[root@ansible-agent git]# git push origin3 dev

Total 0 (delta 0), reused 0 (delta 0)

To git@github.com:xingmin860818/git.git

\* [new branch] dev -> dev

但是，并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；
* feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

总之，就是在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：

多人协作时，大家都会往maser和dev分支上推送各自的修改。

现在模拟一个你的朋友，可以在另一台电脑或同一电脑的其他目录下克隆：

[root@ansible-agent test5]# git clone git@github.com:xingmin860818/git.

git

默认情况下，你的朋友只能看到本地的master分支，可以使用git branch查看

现在你的朋友要在dev分支上开发，就必须创建远程origin的dev分支到本地，于是，他用这个命令创建本地dev分支：

[root@ansible-agent git]# git checkout -b dev origin/dev

Branch dev set up to track remote branch dev from origin.

Switched to a new branch 'dev'

接下来就可以进行编程了，修改完成后可以push到远程

[root@ansible-agent git]# git add readme.txt

[root@ansible-agent git]# git commit -m 'add'

[dev dbf3fc5] add

Committer: root <root@ansible-agent.(none)>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly:

git config --global user.name "Your Name"

git config --global user.email you@example.com

If the identity used for this commit is wrong, you can fix it with:

git commit --amend --author='Your Name <you@example.com>'

1 files changed, 1 insertions(+), 0 deletions(-)

[root@ansible-agent git]# git push origin dev 推送到远程

Counting objects: 5, done.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 293 bytes, done.

Total 3 (delta 1), reused 0 (delta 0)

To git@github.com:xingmin860818/git.git

86d6cf0..dbf3fc5 dev -> dev

如果你的朋友已经向orgin/dev分支推送了他得提交，而碰巧你也对同意的文件做了修改，并试图推送会提示推送失败，因为你的小伙伴的最新提交和你试图推送的提交有冲突，解决办法也很简单，git已经提示我们先用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：

[root@ansible-agent git]# git pull

git pull也失败了，原因是没有指定本地dev分支与远程origin/dev分支的链接，根据提示，设置dev和origin/dev的链接：

[root@ansible-agent git]# git branch --set-upstream dev origin/dev

Branch dev set up to track remote branch dev from origin.

[root@ansible-agent git]# git pull

Already up-to-date.

这回git pull成功，但是合并有冲突，需要手动解决，解决的方法和分支管理中的[解决冲突](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001375840202368c74be33fbd884e71b570f2cc3c0d1dcf000)完全一样。解决后，提交，再push：

$ git **commit** -m "merge & fix hello.py"

[dev adca45d] merge & fix hello.py

$ git push origin dev

Counting objects: 10, done.

Delta compression **using** up **to** 4 threads.

Compressing objects: 100% (5/5), done.

Writing objects: 100% (6/6), 747 bytes, done.

Total 6 (delta 0), reused 0 (delta 0)

**To** git@github.com:michaelliao/learngit.git

291bea8..adca45d 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git branch --set-upstream 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。

### 小结

* 查看远程库信息，使用git remote -v；
* 本地新建的分支如果不推送到远程，对其他人就是不可见的；
* 从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；
* 在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；
* 建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；
* 从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

创建标签：

git中打标签非常简单，首先切换到需要打标签的分支上：

[root@ansible-agent git]# git branch

\* dev

master

[root@ansible-agent git]# git checkout master

Switched to branch 'master'

然后用命令git tag <name>就可以打一个标签

[root@ansible-agent git]# git tag v1.3

[root@ansible-agent git]# git tag

v1.1

v1.3

默认标签是打在最新提交的commit上的，有时候，如果忘了打标签，比如现在是周五了，但是应该在周一打的标签没有打怎么办？

方法是找到历史提交的commit id，然后打上就可以了：

[root@ansible-agent git]# git log --pretty=oneline --abbrev-commit

7fcdd6f msg2

4d984bc msg

74d4fb3 merged bug fix 101

6a7b0b9 fix bug 101

7bca23f merge with no-ff

86d6cf0 add merge

abc816e conflit fixed

6ed2f44 simpol

c4fa083 and simple

27a1a27 b

c465d28 branch test

02e6f05 msg

比方说给最后一个打标签：

[root@ansible-agent git]# git tag v1.0 02e6f05

[root@ansible-agent git]# git tag

v1.0

v1.1

v1.3

注意，标签不是按时间顺序列出的，而是按字母排序的，可以用git show <tag name>查看标签信息：

[root@ansible-agent git]# git show v1.0

commit 02e6f05e45a63c45d8e99bd057955f9247a62616

Author: root <root@ansible-agent.(none)>

Date: Mon Nov 30 21:54:04 2015 +0800

msg

可以看到确实打在 msg这次提交上

还可以创建带有说明的标签，用-a指定标签名，-m指定说明文字

[root@ansible-agent git]# git tag -a v2.0 -m 'aaaaa' 27a1a27

[root@ansible-agent git]# git show v2.0

tag v2.0

Tagger: root <root@ansible-agent.(none)>

Date: Tue Dec 1 04:07:03 2015 +0800

aaaaa

commit 27a1a2769a3f7b462544341b9605f14a6179ee72

Author: root <root@ansible-agent.(none)>

Date: Tue Dec 1 00:33:05 2015 +0800

b

操作标签：

如果标签打错了，也可以删除：

[root@ansible-agent git]# git tag -d v1.0

Deleted tag 'v1.0' (was 02e6f05)

因为创建的标签纸存储在本地，不会自动推送到远程，所以，打错的标签可以在本地安全的删除。

如果要推送某个标签到远程，使用命令 git push origin <tagname>

[root@ansible-agent git]# git push origin v2.0

Counting objects: 1, done.

Writing objects: 100% (1/1), 155 bytes, done.

Total 1 (delta 0), reused 0 (delta 0)

To git@github.com:xingmin860818/git.git

\* [new tag] v2.0 -> v2.0

或者一次推送全部尚未推送到远程的本地标签：

[root@ansible-agent git]# git push origin --tags

Total 0 (delta 0), reused 0 (delta 0)

To git@github.com:xingmin860818/git.git

\* [new tag] v1.1 -> v1.1

\* [new tag] v1.3 -> v1.3

如果标签已推送到远程，要删除远程标签，先删除本地

[root@ansible-agent git]# git tag -d v1.1

Deleted tag 'v1.1' (was 7fcdd6f)

再删除远程

[root@ansible-agent git]# git push origin :refs/tags/v1.1

To git@github.com:xingmin860818/git.git

- [deleted] v1.1