目录

[一、git知识分享 2](#_Toc17107053)

[1.git clone与git pull的区别： 2](#_Toc17107054)

[2.集中式版本控制系统和分布式版本控制系统： 2](#_Toc17107055)

[3.工作区、暂存区、版本库： 2](#_Toc17107056)

[4.工作目录中文件的三种状态： 3](#_Toc17107057)

[二、git命令实操 3](#_Toc17107058)

[1.创建版本库 3](#_Toc17107059)

[2.通过git init命令将该目录变成git管理的仓库 4](#_Toc17107060)

[3.把文件添加到版本库 4](#_Toc17107061)

[4.查看文件的状态 4](#_Toc17107062)

[5.将项目push到远程仓库 5](#_Toc17107063)

[6.查看提交记录 6](#_Toc17107064)

[7.查看命令历史 6](#_Toc17107065)

[8.查看信息 6](#_Toc17107066)

[9.撤销修改 7](#_Toc17107067)

[10.版本回退 9](#_Toc17107068)

[11.删除文件 11](#_Toc17107069)

[12.分支管理 12](#_Toc17107070)

[13.标签管理 14](#_Toc17107071)

# 一、git知识分享

## 1.git clone与git pull的区别：

git clone即克隆代码，从远程服务器克隆一个一模一样的版本库到本地，是一个本地从无到有的过程。

git pull 即拉取代码，从远程仓库获取一个branch分支的更新到本地，并更新到本地库（pull是指同步一个在本地有版本的库内容更新的部分到本地库），相当于从远程获取最新版本并merge到本地，git pull = git fetch+git merge

git clone是远程操作的第一步，本地修改完代码后，在从本地仓库push到远程仓库前要先git pull,保证push 到远程下仓库时代码没有版本冲突。

## 2.集中式版本控制系统和分布式版本控制系统：

集中式版本控制系统，版本库集中存放在中央服务器，每次需要的时候都要从中央服务器上获取最新版本，用完后在将自己的版本库推送的中央服务器。集中版本控制系统需要联网才能工作。

分布式版本控制系统，不存在“中央服务器”，每个人的电脑都是一个完整的版本库，因此，工作的时候不需要联网。注：Git是用c语言开发的分布式版本控制系统。

## 3.工作区、暂存区、版本库：

* 工作区：就是所创建的文件目录；
* 暂存区：即stage或者index。一般存放在”.git”文件中的index文件，所以也可以叫暂存区为索引；
* 版本库：工作区有一个隐藏的文件夹（可通过设置取消隐藏）.git，即为git的版本库。

下图表示了工作区、暂存区、版本库之间的区别：

本机

仓库

服务器

仓库

暂存区

工作目录

git push

git pull

add

checkout

commit

reset

图中最左侧为工作区，右侧为版本库。在版本库中标记为indec(stage)的部分为暂存区，标记为master的是master分支所代表的目录树；HEAD实际是指向master的一个游标。

当对工作区修改（或新增）的文件执行“git add”命令时，暂存区的目录树会被更新；当执行提交操作”git commit”时，暂存区中的目录树或被写到版本库中，master分支会做相应的更新。

## 4.工作目录中文件的三种状态：

* modified：对原文件修改
* deleted：已提交的版本库的文件被删除
* untracked：新添加的文件，未被提交的版本库

# 二、git命令实操

## 1.创建版本库

版本库即仓库 repository，可以理解为一个目录，这个目录中的所有文件都被git管理，每个文件的修改、删除git都能跟踪。

创建版本库：

20190301511@NW-20190301511 MINGW64 /d/git学习 (master)

$ mkdirlearngit

20190301511@NW-20190301511 MINGW64 /d/git学习 (master)

$ cd learngit

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ pwd

/d/git学习/learngit

pwd命令为显示当前目录；

## 2.通过git init命令将该目录变成git管理的仓库

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git init

Initialized empty Git repository in D:/git学习/learngit/.git/

## 3.把文件添加到版本库

首先在learngit文件夹下创建一个readme.txt文件

第一步：使用git add告诉git,把文件添加到仓库

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git add Readme.txt

第二步：使用git commit告诉git，把文件提交到仓库

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git commit -m "wrote a readme file"

[master (root-commit) 8d4c750] wrote a readme file

1 file changed, 1 insertion(+)

create mode 100644 Readme.txt

注意：git commit命令，-m后面输入的是本次提交的说明，可以是任何内容。

## 4.查看文件的状态

* 使用git status命令可以查看文件的状态，加上参数-s可以简要显示当前状态的内容：git status -s

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ 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: test.txt

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

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git status -s

M test.txt

## 5.将项目push到远程仓库

首先创建一个远程仓库，右击 给git bash，使用git init命令初始化本地git仓库配置；git add . 命令添加工作区代码到暂存区（.代表该文件夹下的所有文件）；git commit -m “”命令添加暂存区代码到本地仓库；git remote add origin [远程仓库地址]命令添加远程仓库；在push文件之前先使用git pull origin master命令更新文件；git push -u origin master命令将文件push到远程仓库。

20190301511@NW-20190301511 MINGW64 /d/projects/workspace

$ git init

Initialized empty Git repository in D:/projects/workspace/.git/

20190301511@NW-20190301511 MINGW64 /d/projects/workspace (master)

$ git add . （add . 代表添加文件夹下的所有文件）

20190301511@NW-20190301511 MINGW64 /d/projects/workspace (master)

$ git commit -m "workspace"

20190301511@NW-20190301511 MINGW64 /d/projects/workspace (master)

$ git remote add origin https://github.com/KyleWong123/workspace.git

20190301511@NW-20190301511 MINGW64 /d/projects/workspace (master)

$ git pull origin master

From https://github.com/KyleWong123/workspace

\* branch master -> FETCH\_HEAD

Already up to date.

20190301511@NW-20190301511 MINGW64 /d/projects/workspace (master)

$ git push -u origin master

Enumerating objects: 73, done.

Counting objects: 100% (73/73), done.

Delta compression using up to 6 threads

Compressing objects: 100% (62/62), done.

Writing objects: 100% (73/73), 58.15 KiB | 8.31 MiB/s, done.

Total 73 (delta 34), reused 0 (delta 0)

remote: Resolving deltas: 100% (34/34), done.

To https://github.com/KyleWong123/workspace.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

## 6.查看提交记录

* 使用git log命令查看提交日志

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git log

commit 446b380ad1aab3077c3ffbdb4bba29621642573e (HEAD ->master)

Author: KyleWong123 <15029378233@163.com>

Date: Sat Jul 27 15:08:18 2019 +0800

modify

commit 8d4c7507e1ed76cec823836f749748bd00c68320

Author: KyleWong123 <15029378233@163.com>

Date: Sat Jul 27 11:46:42 2019 +0800

wrote a readme file

* 使用 git log –pretty=oneline可以美化log信息

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git log --pretty=oneline

446b380ad1aab3077c3ffbdb4bba29621642573e (HEAD ->master) modify

8d4c7507e1ed76cec823836f749748bd00c68320 wrote a readme file

## 7.查看命令历史

* Git提供的git reflog可以查看命令历史：

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git reflog

446b380 (HEAD ->master) HEAD@{0}: reset: moving to 446b

8d4c750 HEAD@{1}: reset: moving to HEAD^

ad84041 HEAD@{2}: commit (amend): modify

446b380 (HEAD ->master) HEAD@{3}: commit: modify

8d4c750 HEAD@{4}: commit (initial): wrote a readme file

## 8.查看信息

* Git提供 git diff 可以查看工作区和暂存区的区别，

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git diff

**diff --git a/test.txt b/test.txt**

**index 425d28e..ed57f99 100644**

**--- a/test.txt**

**+++ b/test.txt**

@@ -1,2 +1,2 @@

123456

-foolish

\ No newline at end of file

+78910

* git diff HEAD 查看工作区与当前分支最新commit的差异

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git diff HEAD -- test.txt

**diff --git a/test.txt b/test.txt**

**index 425d28e..ed57f99 100644**

**--- a/test.txt**

**+++ b/test.txt**

@@ -1,2 +1,2 @@

123456

-foolish

\ No newline at end of file

+78910

\ No newline at end of file

* git diff [first-branch]…[second-branch] 查看两次commit之间的差异

## 9.撤销修改

* git checkout – file

如果修改了本地文件，但还没add 到暂存区。主要用来丢弃工作区的修改，以下例子说明：原本工作区文件的内容为：123456；我修改文本内容为：123456789后查看文件的状态为：modified，且查看文本内容已经是：123456789，然后使用命令：git checkout – test.txt撤销工作区的修改，查看文本内容又回到了原来的：123456.

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ cat test.txt

123456

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git status

On branch master

Changes to be committed:

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

modified: test.txt

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: test.txt

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ cat test.txt

123456

789

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git checkout -- test.txt

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ cat test.txt

123456

* git reset HEAD file

如果修改了本地文件，并且已经add到了暂存区，使用 git reset HEAD file就可以把暂存区的修改（即add操作）撤销掉，重新放回工作区。以下示例说明：将本地文件修改后，查看当前文件状态为：modified，并且等到add及提交；然后执行git add将修改的文件添加到暂存区；然后使用: git reset HEAD file撤销上一步的add操作，查看文件状态为：modified，并且等待add 以及commit，即回到了工作区状态

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ 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: test.txt

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

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git add .

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git status

On branch master

Changes to be committed:

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

modified: test.txt

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git reset HEAD test.txt

Unstaged changes after reset:

M test.txt

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ 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: test.txt

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

* 撤销提交—版本回退

如果已经将add到暂存区的问道进行了commit，要想撤销，就需要使用版本回退（见下章）

## 10.版本回退

* 在git中用HEAD表示当前版本，也就是最新提交的版本，上一个版本就是HEAD^,上上一个版本就是HEAD^^(也可表示为HEAD~数字)，现在利用git reset命令将版本退回到上个版本，使用cat命令查看当前版本的内容，使用git log查看提交的版本日志：

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git reset --hard HEAD^

HEAD is now at 8d4c750 wrote a readme file

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ cat Readme.txt

learn git

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git log

commit 8d4c7507e1ed76cec823836f749748bd00c68320 (HEAD ->master)

Author: KyleWong123 <15029378233@163.com>

Date: Sat Jul 27 11:46:42 2019 +0800

wrote a readme file

* 通过log可以看到当前版本已经回去了，而最新提交的版本已经不在了，要想找到最新提交的版本就需要使用commit id,然后使用git reset –-hard 446b命令(446b为最新提交版本的commit id的前四位)，就可以找到最新版本，通过git log --pretty=oneline查看提交日志，就可以看到两次记录,通过cat Readme.txt查看内容可以看到为最新修改过的内容：

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git reset --hard 446b

HEAD is now at 446b380 modify

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ git log --pretty=oneline

446b380ad1aab3077c3ffbdb4bba29621642573e (HEAD ->master) modify

8d4c7507e1ed76cec823836f749748bd00c68320 wrote a readme file

20190301511@NW-20190301511 MINGW64 /d/git学习/learngit (master)

$ cat Readme.txt

learn git

append

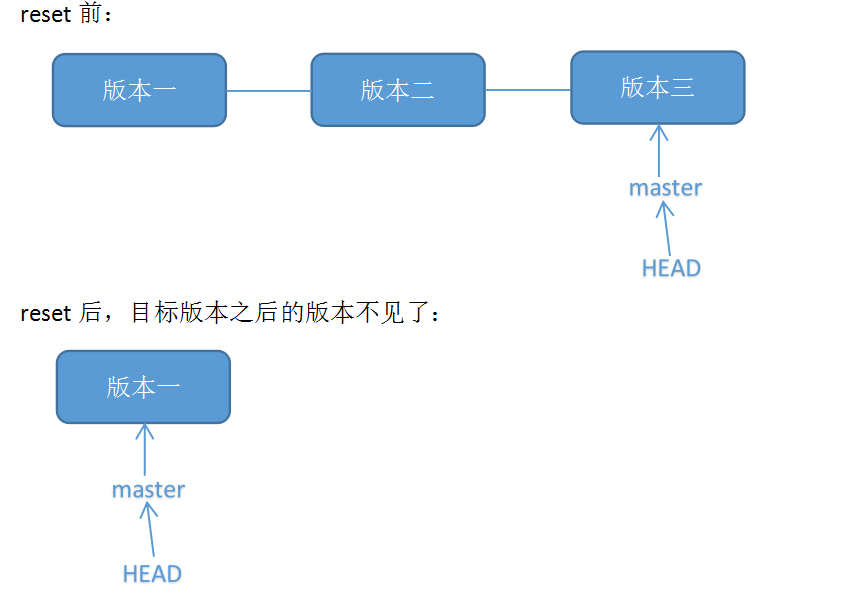
* git revert

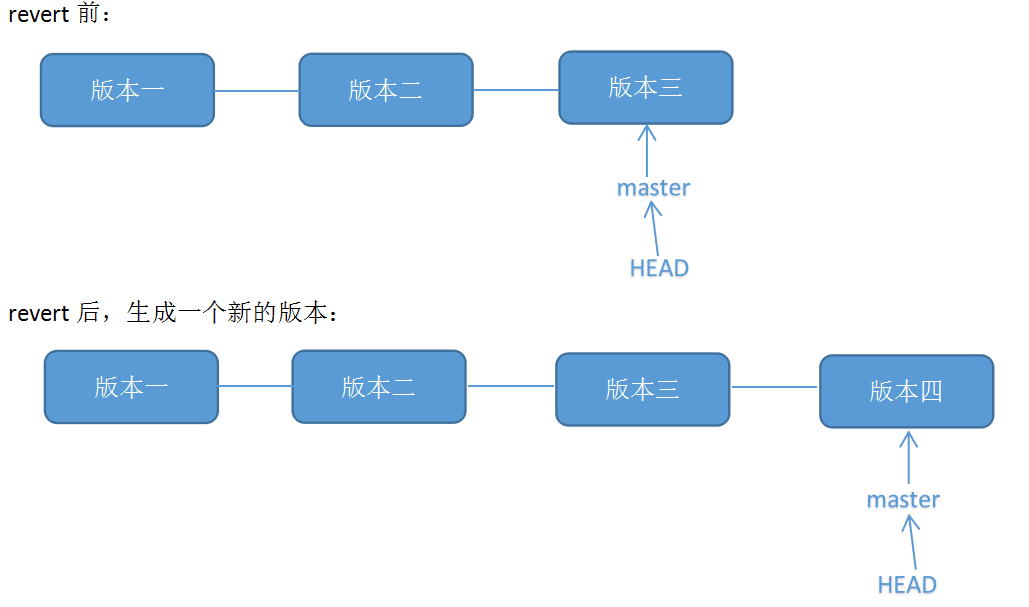
git revert是提交一个新的版本，对于之前的commit等都会保留，常用命令为：git revert HEAD 撤销前一次commit;

git revert HEAD^ 撤销倒数第二次commit;

git revert commit-id :撤销指定的commit

以下附上git reset与git revert的区别图：





由两张图可以清楚的看到git reset的原理：改变HEAD指向的位置，就是将HEAD指向要回退到的把版本，reset后，目标版本之后的本版就不在了；

git revert的原理：git revert是用于“反做”某一个版本，以达到撤销该版本的修改的目的。比如，我们commit了三个版本（版本一、版本二、版本三），突然发现版本二不行（如：有bug），想要撤销版本二，但又不想影响撤销版本三的提交，就可以用 git revert 命令来反做版本二，生成新的版本四，这个版本四里会保留版本三的东西，但撤销了版本二的东西。

## 11.删除文件

* git rm file

用git rm命令可以删除工作区的文件，（注意：git rm删除文件既删除了工作区的文件并add到了暂存区）然后确实要从版本库中删除，就可以执行：git commit命令进行提交；若是误删了工作区的文档想要恢复，首先使用：git reset HEAD file撤销add操作，再使用git checkout – file撤销此次删除操作（删除也是一次修改操作）

* rm file

使用rm 命令删除工作区文件，可以使用：git checkout – file找回

## 12.分支管理

* 浅谈分支

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。Git的分支是与众不同的，无论创建、切换和删除分支，Git在1秒钟之内就能完成！无论你的版本库是1个文件还是1万个文件。

* 创建分支

使用git checkout –b 分支名 来创建分支并立即切换到该分支下；等同于：

git branch 分支名 + git checkout 分支名，然后使用git branch查看分支，该命令会列出所有分支，\*代表当前分支.

示例1说明：我用git checkout –b branch1命令创建了branch1分支并立刻切换到了该分支下，用git branch可以查看到当前有 branch1、master两个分支，并且\*指向当前的分支branch1

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (master)

$ git checkout -b branch1

Switched to a new branch 'branch1'

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ git branch

\* branch1

Master

示例2说明：我用git branch branch2创建了branch2分支，使用git branch查看分支，看到branch2分支已创建成功，但目前并未指向该分支；然后使用 git checkout branch2将分支切换到branch2，再使用git branch查看分支，看到\*已经指向branch2这条分支。

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ git branch branch2

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ git branch

\* branch1

branch2

master

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ git checkout branch2

Switched to branch 'branch2'

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch2)

$ git branch

branch1

\* branch2

master

* 合并分支

使用git merge 指定分支名 命令将指定分支与当前分支进行合并

示例说明：首先我创建了test.txt文件并输入了相关内容，然后在branch1分支上进行了提交（具体命令操作如下）

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ touch test.txt

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ git add test.txt

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ git commit -m "test branch"

[branch1 743a77a] test branch

1 file changed, 1 insertion(+)

create mode 100644 test.txt

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ cat test.txt

This is a test about Branch.

然后将分支切换到master主分支上，查看test.txt的内发现并没有添加的信息，是因为刚才的提交是在branch1分支上的，

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (branch1)

$ git checkout master

Switched to branch 'master'

Your branch is ahead of 'origin/master' by 1 commit.

(use "git push" to publish your local commits)

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (master)

$ cat test.txt

所以使用git merge branch1命令将branch分支与master分支进行合并，再查看test.txt就发现文档内容已经存在（具体命令操作如下）

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (master)

$ git merge branch1

Auto-merging test.txt

Merge made by the 'recursive' strategy.

test.txt | 1 +

1 file changed, 1 insertion(+)

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (master)

$ cat test.txt

This is a test about Branch.

* 删除分支

使用 git branch –d 分支名可以删除分支

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (master)

$ git branch -d branch2

Deleted branch branch2 (was b2136b9).

Administrator@DXS2TL2FJB14HBG MINGW64 ~/Desktop/ppt (master)

$ git branch

branch1

\* master

使用 git push origin delete [branch-name]删除远程分支

## 13.标签管理

发布一个版本时，我们通常先在版本库中打一个标签（tag），这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来。所以，标签也是版本库的一个快照。

Git的标签虽然是版本库的快照，但其实它就是指向某个commit的指针，创建和删除标签都是瞬间完成的。

Git有commit，为什么还要引入tag？

“请把上周一的那个版本打包发布，commit号是6a5819e...”

“一串乱七八糟的数字不好找！”

如果换一个办法：

“请把上周一的那个版本打包发布，版本号是v1.2”

“好的，按照tag v1.2查找commit就行！”

所以，tag就是一个让人容易记住的有意义的名字，它跟某个commit绑在一起。

* 创建标签

使用 git tag [tag-name] 命令打标签，标签默认是打在最新commit上的，使用 git tag查看所有标签

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git tag v1.0

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git tag

v1.0

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git log

commit 1ae3f655eae32e3ccabaa780ae8e68e4bf7c0f0b (HEAD -> master, tag: v1.0)

Merge: 4fd7fc6 0688b48

Author: KyleWong123 <15029378233@163.com>

Date: Mon Aug 19 09:31:24 2019 +0800

merge with --no-ff

commit 0688b481c5c2f0d44fb1775fcacfd20a57d1d97f (dev)

Author: KyleWong123 <15029378233@163.com>

Date: Mon Aug 19 09:30:37 2019 +0800

branch dev

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，现在已经是周五了，但应该在周一打的标签没有打，方法是找到历史提交的commit id，然后打上就可以了：

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git log

commit 1ae3f655eae32e3ccabaa780ae8e68e4bf7c0f0b (HEAD -> master, tag: v1.0)

Merge: 4fd7fc6 0688b48

Author: KyleWong123 <15029378233@163.com>

Date: Mon Aug 19 09:31:24 2019 +0800

merge with --no-ff

commit 0688b481c5c2f0d44fb1775fcacfd20a57d1d97f (dev)

Author: KyleWong123 <15029378233@163.com>

Date: Mon Aug 19 09:30:37 2019 +0800

branch dev

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git tag v2.0 0688

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git tag

v1.0

v2.0

还可以为tag加上一定的注释，使用:

git tag -a tag-name -m “description” commit-id 来创建带有注释的tag，使用 git show tag-name 来查看对应tag的详细信息

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git tag -a v3.0 -m "this is a test about fixing comflic" 4fd7

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git tag

v1.0

v2.0

v3.0

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git show v3.0

tag v3.0

Tagger: KyleWong123 <15029378233@163.com>

Date: Mon Aug 19 11:10:42 2019 +0800

this is a test about fixing comflic

commit 4fd7fc6d30f2a1495da26657aaecd02cf86516b0 (tag: v3.0)

Merge: 45eecd2 ba4b585

Author: KyleWong123 <15029378233@163.com>

…

如果要把 tag 推到远程仓库，则使用：tag push origin tag-name命令 也可以使用： tag push origin –tags 将所有未推送到远程仓库的标签一次性推送

20190301511@NW-20190301511 MINGW64 /d/ppt (master)

$ git push origin v1.0

Enumerating objects: 22, done.

Counting objects: 100% (22/22), done.

Delta compression using up to 6 threads

Compressing objects: 100% (21/21), done.

Writing objects: 100% (22/22), 5.71 MiB | 834.00 KiB/s, done.

Total 22 (delta 5), reused 0 (delta 0)

remote: Resolving deltas: 100% (5/5), done.

To https://github.com/KyleWong123/ppt.git

\* [new tag] v1.0 -> v1.0

* 删除标签

使用 git tag -d tag-name 删除本地创建的tag

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git tag -d v11

Deleted tag 'v11' (was ddf23c8)

20190301511@NW-20190301511 MINGW64 ~/Desktop/test (master)

$ git tag

v1.0

v2.0

v3.0

如果已将tag 推送到了远程仓库，然年想要删除远程仓库的tag，使用： git push origin :refs/tags/tag-name 删除

20190301511@NW-20190301511 MINGW64 /d/ppt (master)

$ git push origin :refs/tags/v1.0

To https://github.com/KyleWong123/ppt.git

- [deleted] v1.0