Git 教程

目 录

[一、git简介及安装 2](#_Toc531525231)

[1、下载地址 3](#_Toc531525232)

[2、linux安装git 3](#_Toc531525233)

[3、window安装git 3](#_Toc531525234)

[3、安装完成配置用户名邮箱 4](#_Toc531525235)

[二、git创建版本库并提交文件 4](#_Toc531525236)

[1、创建版本库 4](#_Toc531525237)

[2、把文件添加到版本库 5](#_Toc531525238)

[三、git版本管理 5](#_Toc531525239)

[1、版本回退 5](#_Toc531525240)

[2、工作区与暂存区 7](#_Toc531525241)

[3、管理修改 8](#_Toc531525242)

[4、撤销修改 9](#_Toc531525243)

[5、删除文件 9](#_Toc531525244)

[四、git远程仓库 10](#_Toc531525245)

[1、添加远程库 10](#_Toc531525246)

[2、从远程库克隆 15](#_Toc531525247)

[五、git分支管理 16](#_Toc531525248)

[1、git创建、合并和删除分支 16](#_Toc531525249)

[2、解决冲突 19](#_Toc531525250)

[3、分支管理及策略 20](#_Toc531525251)

[六、更新远程仓库代码 21](#_Toc531525252)

[1、Fetch + Merge命令 21](#_Toc531525253)

[2、Pull命令 21](#_Toc531525254)

[七、git开发中的两种模式 21](#_Toc531525255)

[1、分支开发模式 21](#_Toc531525256)

[2、fork开发模式 22](#_Toc531525257)

[3、分支开发模式实例命令： 22](#_Toc531525258)

[4、fork开发模式实例命令： 22](#_Toc531525259)

[八、.git/ 目录解析 23](#_Toc531525260)

[1、目录实例图 23](#_Toc531525261)

[2、各文件目录说明 23](#_Toc531525262)

[九、git本地服务器搭建 24](#_Toc531525263)

[十、eclipse git插件使用 24](#_Toc531525264)

一、git简介及安装

git是分布式版本管理系统,区别于集中式版本管理系统,例如svn等。

1、下载地址

①、<https://git-scm.com>

②、<https://github.com/git/git/releases>

③、<https://www.kernel.org/pub/software/scm/git>

2、linux安装git

2.1 yum安装方式

yum install git (缺陷：不是最新版本git,优点：快捷，简单，方便)

2.2 源码安装方式

需要5个依赖，并且保证安装了gcc编译环境:

curl-devel expat-devel gettext-devel openssl-devel zlib-devel

这些都可以通过 yum install 安装实现

yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel

安装步骤：

1.下载源码并解压:

cd /usr/local/src

wget <https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.16.0.tar.gz>

tar –zxvf git-2.16.0.tar.gz

2.编译检测安装:

cd /git-2.16.0

./configure --prefix=/usr/local/git

make && make install

3、window安装git

1.下载安装

浏览器输入下载地址: <https://git-scm.com>,进入下载界面,linux版本亦可通过该网址进行下载;

下载完成后文件名为: Git-2.18.0-64-bit.exe,点击exe文件即可执行安装。

下载界面截图如下:



3、安装完成配置用户名邮箱

$ git –-version

$ git config –-global user.name “yourname”

$ git config –-global user.email “your@email”

二、git创建版本库并提交文件

1、创建版本库

$ cd /usr/local

$ mkdir git\_repository

$ git init git\_repository

已初始化空的 Git 仓库于 /usr/local/git\_repository/.git/

$ cd git\_repository

进入git\_repository 使用ls –a 命令会发现存在了一个.git文件夹，到此证明你已经创建了一个git本地仓库。没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

2、把文件添加到版本库

Microsoft的Word格式是二进制格式，因此，版本控制系统是没法跟踪Word文件的改动的。强烈建议使用标准的UTF-8编码，所有语言使用同一种编码，既没有冲突，又被所有平台所支持。

编写一个readme.txt文件，文件内容为：

Git **is** a version control system.

Git **is** free software.

将该文件放到git\_repository目录（库）下,一定要放在该目录下,因为该目录是一个git仓库。

将文件添加到git仓库步骤：

$ git add readme.txt

$ git commit –m “first add readme.txt”

如果是多个文件一起提交可以这样：

$ git add readme.txt

$ git add file1.txt file2.txt

$ git commit –m “3 file add”

简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。所以强烈建议commit时使用-m添加有意义的说明。

三、git版本管理

1、版本回退

将readme.txt修改两次并提交，第一次修改内容：

Git **is** a version control system.

Git **is** free software.

Git update first.

第二次修改内容：

Git **is** a version control system.

Git **is** free software.

Git update first.

Git distributed under the GPL.

提交操作：

[root@localhost git\_repository]# git add readme.txt

[root@localhost git\_repository]# git commit -m "first update"

[master 0ca9777] first update

1 file changed, 1 insertion(+)

[root@localhost git\_repository]# vim readme.txt

[root@localhost git\_repository]# git add readme.txt

[root@localhost git\_repository]# git commit -m "append GPL"

[master 44dfde6] append GPL

1 file changed, 1 insertion(+)

通过三次的增加提交，现在readme.txt在git仓库已经有了三个版本了：

版本1：first add readme.txt

版本2：first update

版本3：append GPL

在git中，我们可以使用git log 命令查看版本：

[root@localhost git\_repository]# git log

commit 44dfde6e93bcfe1415343495488389ffed10d00f (**HEAD -> master**)

Author: yument <1137804712@qq.com>

Date: Wed Aug 1 07:01:36 2018 +0800

append GPL

commit 0ca97773a48c94d10d941610be3b421a6244bc59

Author: yument <1137804712@qq.com>

Date: Wed Aug 1 07:00:24 2018 +0800

first update

commit c2331965a7f9562bb9663dd2c5eddf299e528c56

Author: yument <1137804712@qq.com>

Date: Wed Aug 1 06:47:58 2018 +0800

first add file

[root@localhost git\_repository]#

如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline参数:

[root@localhost git\_repository]# git log --pretty=oneline

44dfde6e93bcfe1415343495488389ffed10d00f (**HEAD -> master**) append GPL

0ca97773a48c94d10d941610be3b421a6244bc59 first update

c2331965a7f9562bb9663dd2c5eddf299e528c56 first add file

接下来我们准备版本回退：

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，也就是最新的提交44dfde...，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

现在，我们要把当前版本append GPL回退到上一个版本first update，就可以使用git reset命令：

[root@localhost git\_repository]# git reset --hard HEAD^

HEAD 现在位于 0ca9777 first update

查看readme.txt文件内容，确实回到了first update时的内容。

此时再通过git log再看：

[root@localhost git\_repository]# git log --pretty=oneline

0ca97773a48c94d10d941610be3b421a6244bc59 (**HEAD -> master**) first update

c2331965a7f9562bb9663dd2c5eddf299e528c56 first add file

最新的那个版本append GPL已经看不到了,如果再想回到该版本怎么办,可以通过查找之前版本的commit id回去,id为44dfde6：

[root@localhost git\_repository]# git reset --hard 44dfde6

HEAD 现在位于 44dfde6 append GPL

版本号没必要写全，前几位就可以了，Git会自动去找。当然也不能只写前一两位，因为Git可能会找到多个版本号，就无法确定是哪一个了。

2、工作区与暂存区

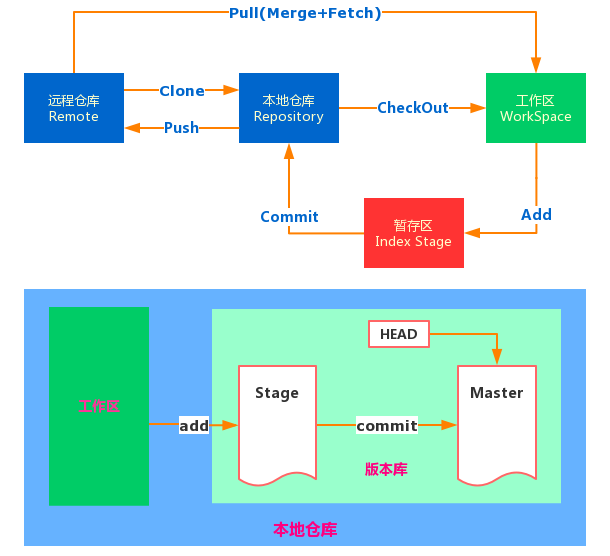
工作区（Working Directory）

就是你在电脑里能看到的目录，比如我的git\_repository文件夹就是一个工作区。

版本库（Repository）

工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



3、管理修改

工作区的文件每次修改都必须git add到暂存区，才能git commit到版本库，直接使用git commit是不能直接提交到版本库的，意思就是版本的更新都是通过暂存区的更新而更新的。通过以下实例来说明这一点。

再次编辑readme.txt，增加一行内容为:

Git **is** a version control system.

Git **is** free software.

Git update first.

git mamage is update,not file.

保存退出,然后使用git add添加到暂存区,并查看状态,git status:

[root@localhost git\_repository]# vim readme.txt

[root@localhost git\_repository]# git add readme.txt

[root@localhost git\_repository]# git status

位于分支 master

要提交的变更：

（使用 "git reset HEAD <文件>..." 以取消暂存）

修改： readme.txt

我们先不提交，再编辑一次readme.txt,内容如下：

Git **is** a version control system.

Git **is** free software.

Git update first.

git mamage is update,not file. once agin update.

然后直接git commit,并使用git status查看状态：

[root@localhost git\_repository]# vim readme.txt

[root@localhost git\_repository]# git commit -m "git tracks changes"

[master 1dae6af] git tracks changes

1 file changed, 1 insertion(+)

[root@localhost git\_repository]# git status

位于分支 master

尚未暂存以备提交的变更：

（使用 "git add <文件>..." 更新要提交的内容）

（使用 "git checkout -- <文件>..." 丢弃工作区的改动）

修改： readme.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

可以看到第二次编辑的内容并未有提交，因为第二次修改的内容没有添加到暂存区。使用git diff命名查看当前工作区与版本库中最新版本的区别：

[root@localhost git\_repository]# git diff HEAD -- readme.txt

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

**index 70945fc..a1e607a 100644**

**--- a/readme.txt**

**+++ b/readme.txt**

@@ -2,4 +2,4 @@ git is a version control system.

git is a free software.

git update first.

git distributed under the GPL.

-git manage is update ,not file.

+git manage is update ,not file. once agin update.

4、撤销修改

撤销修改主要是在还没有完成提交操作时发生。即有两种情况：

1.工作区内容已经被修改保存，还没有添加到暂存区;

2.工作区内容修改后，已经添加到暂存区了，然后又做了修改。

通过先前讲述的管理修改，我们在git status时会发现提示，让我们用git checkout -- file可以丢弃工作区的修改。

git checkout对应上述第一种情况，撤销修改就回到和版本库一模一样的状态；

git checkout对应上述第二种情况，撤销修改就回到添加到暂存区后的状态；

实例：

[root@localhost git\_repository]# git checkout -- readme.txt

[root@localhost git\_repository]# git status

位于分支 master

无文件要提交，干净的工作区

将管理修改时，第二次修改我们并没有添加到暂存区，撤销修改，查看文件内容，会发现还原了。

5、删除文件

工作区的文件修改并提交到了版本库，现在我们直接在工作区将该文件删掉了，但是版本库的还没有删除，这时候造成版本不一致,git status会告诉我们那些文件被删了。这时候我们有两种选择：

1.这个文件确实没有用了，所以需要同时删除版本库对应的；

2.这个文件是我们不小心误删的，所以可以通过版本库再还原回来；

实例:

[root@localhost git\_repository]# vim test.txt

[root@localhost git\_repository]# git add test.txt

[root@localhost git\_repository]# git commit -m "test git rm"

[master 70d133c] test git rm

1 file changed, 2 insertions(+)

create mode 100644 test.txt

[root@localhost git\_repository]# ll

-rw-r--r--. 1 root root 138 8月 1 22:47 readme.txt

-rw-r--r--. 1 root root 32 8月 1 22:58 test.txt

恢复test.txt：

[root@localhost git\_repository]# rm -rf test.txt

[root@localhost git\_repository]# ll

-rw-r--r--. 1 root root 138 8月 1 22:47 readme.txt

[root@localhost git\_repository]# git status

位于分支 master

尚未暂存以备提交的变更：

（使用 "git add/rm <文件>..." 更新要提交的内容）

（使用 "git checkout -- <文件>..." 丢弃工作区的改动）

删除： test.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

[root@localhost git\_repository]# git checkout -- test.txt

[root@localhost git\_repository]# ll

-rw-r--r--. 1 root root 138 8月 1 22:47 readme.txt

-rw-r--r--. 1 root root 32 8月 1 23:01 test.txt

git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

同版本库中一起删除，用命令git rm删掉，并且git commit:

[root@localhost git\_repository]# rm -rf test.txt

[root@localhost git\_repository]# ll

-rw-r--r--. 1 root root 138 8月 1 22:47 readme.txt

[root@localhost git\_repository]# git status

位于分支 master

尚未暂存以备提交的变更：

（使用 "git add/rm <文件>..." 更新要提交的内容）

（使用 "git checkout -- <文件>..." 丢弃工作区的改动）

删除： test.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

[root@localhost git\_repository]# git rm test.txt

rm 'test.txt'

[root@localhost git\_repository]# git commit -m "rm test.txt"

[master 9877122] rm test.txt

1 file changed, 2 deletions(-)

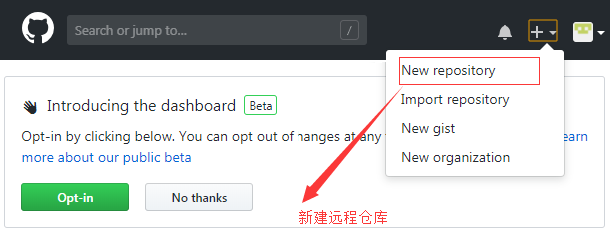
delete mode 100644 test.txt

四、git远程仓库

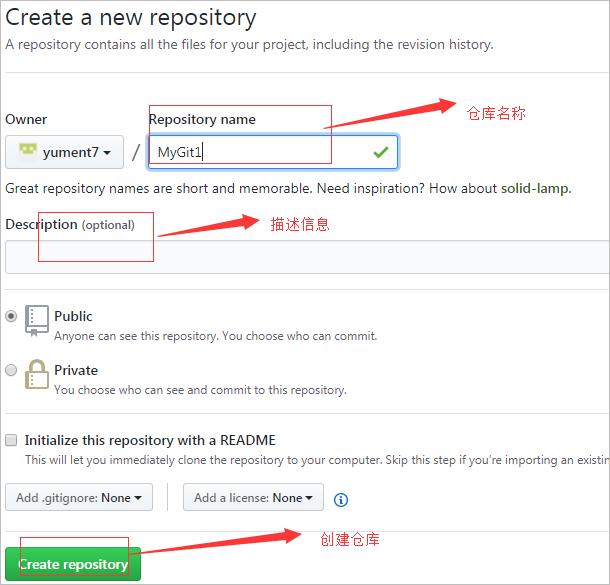
1、添加远程库

登录github,没有账号可以先注册github账号。

创建仓库：



点击New repository到具体页面，如下：

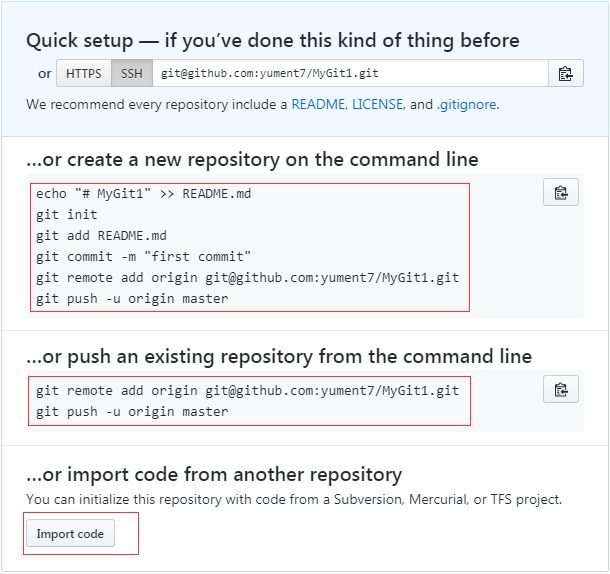


描述信息可写可不写;

该仓库可以设为公共和私有;

初始化仓库并产生一个readme文件,可选可不选，在此我们不选择。

点击Create Repository生成仓库，如下，给出提示信息：



GitHub告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

根据GitHub的提示，在本地的仓库下运行命令：

$ git remote add origin git@github.com:yument7/MyGit.git

添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

下一步，就可以把本地库的所有内容推送到远程库上：

$ git push -u origin master

The authenticity of host 'github.com (52.74.223.119)' 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)? yes

我们是第一次使用Git的clone或者push命令连接GitHub，所以会得到一个警告,这个警告只会出现一次，后面的操作就不会任何警告了。

我们按提示直接输入yes得到如下信息：

Warning: Permanently added 'github.com,52.74.223.119' (RSA) to the list of known hosts.

Permission denied (publickey).

fatal: 无法读取远程仓库。

请确认您有正确的访问权限并且仓库存在。

这有由于没有生成ssh key，且没有上传到github，ssh连接的时候是需要秘钥的，端到端建立安全连接。

**SSH key生成及将本机公钥上传到github**

1.使用命令：ssh-keygen –t rsa –C “xxx@xxx.com”;

说明：邮箱为你在github账号上绑定的邮箱。

之后会要求确认路径和输入密码，我们这使用默认的一路回车就行。成功的话会在提示的路径下生成.ssh文件夹，进去，打开id\_rsa.pub，复制里面的key，该key就是rsa加密产生的公钥。

[root@localhost local]# ssh-keygen -t rsa -C "1137804712@qq.com"

Generating public/private rsa key pair.

Enter file in which to save the key (/root/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /root/.ssh/id\_rsa.

Your public key has been saved in /root/.ssh/id\_rsa.pub.

The key fingerprint is:

79:6c:ee:ff:d5:9e:18:34:d9:be:a1:05:9d:49:ab:fd 1137804712@qq.com

The key's randomart image is:

+--[ RSA 2048]----+

| . |

| o = +|

| S + = \* |

| + . \* .|

| . o =o|

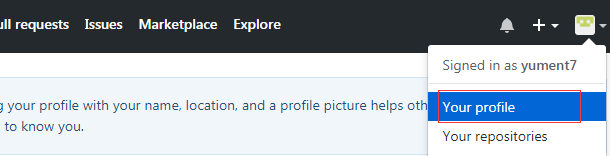
| . =o=|

| ...+.oE|

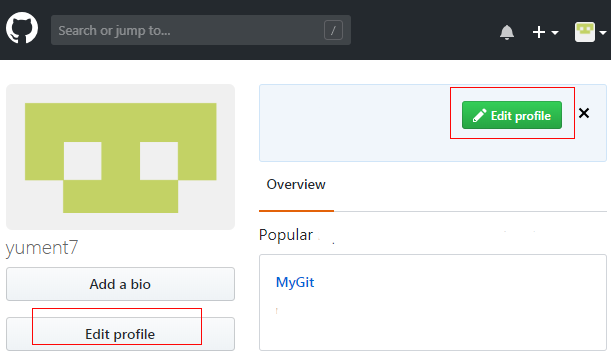
+-----------------+

从执行的结果中我们看到，我们的公钥在 /root/.ssh目录下，文件为：id\_rsa.pub

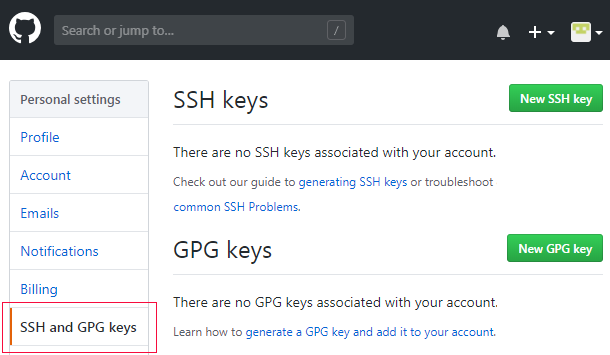
2.回到github网站，将复制的公钥添加到github上进行关联：



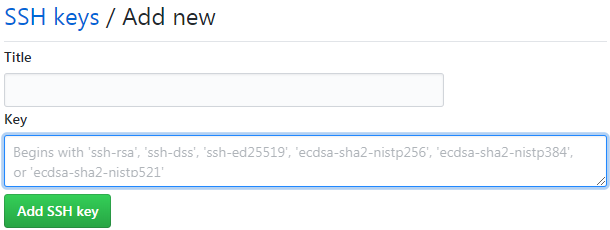
点击`Your profile`进入：



点击 `Edit profile` 进入，找到`SSH and GPG keys`选项，点击，右侧会出现相应内容：



然后点击`New SSH key`,title随便填写，将复制的key粘贴进去。



添加成功之后，切换到shell终端，进行测试：

[root@localhost .ssh]# ssh -T git@github.com

Hi yument7! You've successfully authenticated, but GitHub does not provide shell access.

表示关联成功了。我们回到工作区目录,先解除之前的关联，重新关联远程仓库：

[root@localhost ]# git remote remove origin

[root@localhost ]# git remote add origin git@github.com:yument7/MyGit1.git

[root@localhost ]# git push -u origin master

对象计数中: 16, 完成.

Delta compression using up to 4 threads.

压缩对象中: 100% (11/11), 完成.

写入对象中: 100% (16/16), 1.27 KiB | 1.27 MiB/s, 完成.

Total 16 (delta 4), reused 0 (delta 0)

remote: Resolving deltas: 100% (4/4), done.

To github.com:yument7/MyGit1.git

\* [new branch] master -> master

分支 'master' 设置为跟踪来自 'origin' 的远程分支 'master'。

可以看到我们提交的东西已经成功了。然后切换到github网页，查看也是成功的。就不在进行截图演示了。

2、从远程库克隆

先前我们是先创建本地仓库，然后去创建远程仓库并进行关联。

现在我们先创建远程仓库，然后克隆远程仓库到本地。

1.使用自己的github账号登录github官网，找到`New repository`菜单单击,进入创建页面。

（注：截图在上面有）

2.填写仓库名称，选择公有私有，勾选initial this repository with a README选项，会为我们自动生成一个readme.md文件，填写好之后，点击`create repoistory`。

3.使用git clone克隆远程仓库：

[root@localhost local]# git clone git@github.com:yument7/MyGit2.git

正克隆到 'MyGit2'...

Warning: Permanently added the RSA host key for IP address '13.229.188.59' to the list of known hosts.

remote: Counting objects: 3, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

接收对象中: 100% (3/3), 完成.

the list of known hosts.

remote: Counting objects: 3, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

接收对象中: 100% (3/3), 完成.

[root@localhost local]# ll

总用量 96

drwxr-xr-x. 3 root root 4096 8月 2 05:18 MyGit2

[root@localhost local]# cd MyGit2/

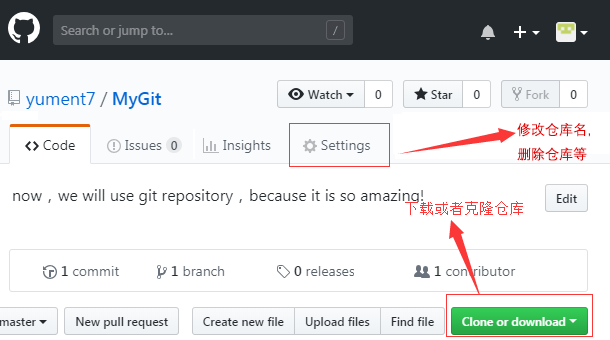
[root@localhost MyGit2]# ll

总用量 4

-rw-r--r--. 1 root root 70 8月 2 05:18 README.md

这样我们就成功的从远程仓库clone一个仓库到本地。

如果我们想删除没有用的仓库，我们可以通过setting选项进入操作：



五、git分支管理

1、git创建、合并和删除分支

创建分支: git branch brhName

切换分支: git checkout brhName

创建并切换分支: git checkout -b brhName

创建并切换分支会让Head指针指向新创建的分支，此时，从主干拉取的分支和主干在同一个版本层次上。

分支创建实例：

[git@localhost git\_repository]$ git branch pay

[git@localhost git\_repository]$ git checkout pay

切换到分支 'pay'

[git@localhost git\_repository]$ git checkout master

切换到分支 'master'

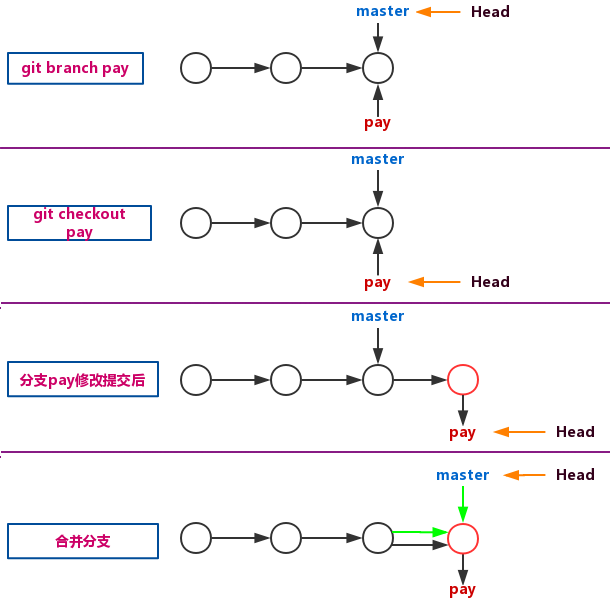
您的分支与上游分支 'origin/master' 一致。

[git@localhost git\_repository]$ git checkout -b paypay

切换到一个新分支 'paypay'

合并分支: git merge brhName

合并分支首先一定要弄清楚主次之分，当我们在分支上对文件内容做了修改，然后想合并到另一个分支上去。首先我们需要添加到暂存区并提交到版本库，此时Head会指向分支当前版本的位置，由于此版本超前了，所以需要切换到想要合并的分支上去，切换之后，Head会指向切换分支当前版本的位置。由于切换后分支文件内容未做任何修改，所以较拉取得的分支修改后的版本会落后一个版本。所以在此基础上合并会直接把修改的内容直接合并过来，然后让当前分支前进一个版本。如图所示：



实例：

[git@localhost git\_repository]$ git checkout pay

切换到分支 'pay'

[git@localhost git\_repository]$ ll

-rw-r--r--. 1 root root 138 8月 1 22:47 readme.txt

[git@localhost git\_repository]$ vim readme.txt

[git@localhost git\_repository]$ git status

位于分支 pay

尚未暂存以备提交的变更：

（使用 "git add <文件>..." 更新要提交的内容）

（使用 "git checkout -- <文件>..." 丢弃工作区的改动）

修改： readme.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

[git@localhost git\_repository]$ git add readme.txt

[git@localhost git\_repository]$ git commit -m "branch first commit"

[pay 7218d1a] branch first commit

1 file changed, 1 insertion(+), 1 deletion(-)

[git@localhost git\_repository]$ git checkout master

切换到分支 'master'

您的分支与上游分支 'origin/master' 一致。

[git@localhost git\_repository]$ git merge pay

更新 9877122..7218d1a

Fast-forward

readme.txt | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

[git@localhost git\_repository]$ git push -u origin master

对象计数中: 3, 完成.

Delta compression using up to 4 threads.

压缩对象中: 100% (2/2), 完成.

写入对象中: 100% (3/3), 295 bytes | 295.00 KiB/s, 完成.

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

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To github.com:yument7/MyGit1.git

9877122..7218d1a master -> master

分支 'master' 设置为跟踪来自 'origin' 的远程分支 'master'。

查看分支: git branch

[git@localhost git\_repository]$ git branch

\* master //\*表示目前正处于master分支

pay

paypay

删除分支: git branch -d brhName

[git@localhost git\_repository]$ git branch -d paypay

已删除分支 paypay（曾为 9877122）。

2、解决冲突

如下情景就会产生冲突：

①.先修改pay分支上readme.txt内容，添加暂存区提交版本库。

②.接着修改master主干上readme.txt内容，添加暂存区提交版本库。

③.合并pay分支。

如下所示：

[git@localhost git\_repository]$ git checkout pay

切换到分支 'pay'

[git@localhost git\_repository]$ vim readme.txt

[git@localhost git\_repository]$ git add readme.txt

[git@localhost git\_repository]$ git commit -m "pay branch second update"

[pay b2aff5e] pay branch second update

1 file changed, 1 insertion(+)

[git@localhost git\_repository]$ git checkout master

切换到分支 'master'

您的分支与上游分支 'origin/master' 一致。

[git@localhost git\_repository]$ vim readme.txt

[git@localhost git\_repository]$ git add readme.txt

[git@localhost git\_repository]$ git commit -m "master test conflication"

[master 2e57468] master test conflication

1 file changed, 1 insertion(+)

[git@localhost git\_repository]$ git merge pay

自动合并 readme.txt

冲突（内容）：合并冲突于 readme.txt

自动合并失败，修正冲突然后提交修正的结果。

[git@localhost git\_repository]$ git status

位于分支 master

您的分支领先 'origin/master' 共 1 个提交。

（使用 "git push" 来发布您的本地提交）

您有尚未合并的路径。

（解决冲突并运行 "git commit"）（使用 "git merge --abort" 终止合并）

未合并的路径：

（使用 "git add <文件>..." 标记解决方案）

双方修改： readme.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

查看readme.txt文件：

Git **is** a version control system.

git update first.

git distributed under the GPL.

git created first branch pay.

<<<<<<< HEAD

we test master updated what will come out!

=======

we let git product confictio，haha!

>>>>>>> pay

Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，我们修改如下后保存：

如何解决冲突，直接修改文件内容如下：

Git **is** a version control system.

git update first.

git distributed under the GPL.

git created first branch pay.

we test master updated what will come out!

we let git product confictio，haha!

然后保存，提交,查看提交记录：

[git@localhost git\_repository]$ git add

[git@localhost git\_repository]$ git add readme.txt

[git@localhost git\_repository]$ git commit -m "conflict fixed"

[master fe83cb1] conflict fixed

[git@localhost git\_repository]$ git log

commit fe83cb10bc2a5fa6cc949725769703613872d6ad (HEAD -> master)

Merge: 2e57468 b2aff5e

Author: yument <1137804712@qq.com>

Date: Sat Aug 4 06:04:56 2018 +0800

conflict fixed

3、分支管理及策略

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。

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

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；

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：



六、更新远程仓库代码

1、Fetch + Merge命令

命令语法详解：

官方解释：**git-fetch - Download objects and refs from another repository** ,意思是说使用该命令会从另一个远程存储库中下载对象和参考，即下载更新文件到版本库，使用fetch命令并不会立即更新到工作区，需要使用Merge命令合并当前更新的远程分支到本地工作区分支。

命令使用：

# 方式一

$ git fetch origin master # 从远程origin仓库的master分支下载代码到本地master

$ git log –p FETCH\_HEAD # 比较本地仓库与远程仓库代码异同

$ git merge origin/master # 合并远程分支代码到本地分支master

# 方式二

$ git fetch origin master:temp # 更新到本地并新建一个分支temp

$ git diff temp # 比较master分支和temp分支的不同

$ git merge temp # 合并temp分支到master分支

$ git branch -d temp # 删除temp

2、Pull命令

命令详解：

pull 的过程可以看做是fetch和 merge的合并，使用pull 名会更新远程代码到本地库并合并到当前工作区分支。

命令使用：

# 完整格式可表示为：

$ git pull <远程主机名> <远程分支名>:<本地分支名>

# 如果远程分支是与当前分支合并，则冒号后面的部分可以省略, 例如：

$ git pull origin master # 更新代码并合并到本地master

七、git开发中的两种模式

1、分支开发模式

分支开发模式的情形是这样：

1.所有开发者拥有修改主干的权限，即只有一个远程仓库

2.所有开发者关联远程仓地址并拉取代码到本地，然后创建分支，在分支进行开发

3.一个版本开发完成后，切回主干，合并分支并推送到远程仓库

可能产生的问题：

1.在合并推送的时候，主干代码可能被其他人提交了一个版本，所以每次合并时记得先更新主干代码;

2.如果已经合并推送到远程仓，恰巧其他人于你之前推送一个版本，并且你们修改了同样的文件，此时就会产生冲突，正确做法，是版本回退，回退到你之前的版本，此时趁机更新主干，然后再合并

2、fork开发模式

fork开发模式情形：

1.每个开发者拥有一个远程服务器账号，通过账号登录远程web服务器，好比git、gitee等，然后每个开发者通过fork拷贝公共代码仓到自己的账户下作为自己私有的代码仓；

2.克隆远程私有代码仓地址到本地，关联公共代码仓地址；

3.本地无需进行分支创建，直接主干开发，新版本先推送到自己私有的远程仓，然后通过web服务器页面向公共仓发合并请求（一般公共仓由管理员维护，管理员会对代码进行审核，提出建议），审核通过将我们的版本合并到公共代码仓

4.更新代码需更新公共代码仓，公共代码仓汇聚了所有开发者开发的版本。

3、分支开发模式实例命令：

[git@lvao git\_project]$ git clone git@gitee.com:public/yument.git

[git@lvao git\_project]$ cd yument

[git@lvao yument]$ git checkout –b lvao-branch

[git@lvao yument]$ git status

[git@lvao yument]$ git add .

[git@lvao yument]$ git commit – m “comment-new testing version”

[git@lvao yument]$ git checkout master

[git@lvao yument]$ git pull upstream master

[git@lvao yument]$ git merge lvao-branch

[git@lvao yument]$ git push upstream master

当有两个分支修改了同一个文件时，合并时就会产生冲突，此时需要我们手动去解决冲突文件，然后再提交。

4、fork开发模式实例命令：

[git@lvao git\_project]$ git clone git@gitee.com:public/lvao/yument.git

[git@lvao git\_project]$ cd yument

[git@lvao yument]$ git remote -v

[git@lvao yument]$ git remote add upstream git@gitee.com/public/yument.git

[git@lvao yument]$ git status

[git@lvao yument]$ git add .

[git@lvao yument]$ git commit – m “comment-new testing version”

[git@lvao yument]$ git push origin master

注意：

冲突的产生：在发送合并请求时，其他人已先发送了合并请求并合并到主干，并且和我们修改了同一个文件。这时就需要这样来处理冲突：

[git@lvao yument]$ git log --pretty=oneline

[git@lvao yument]$ git reset HEAD^ #或者 git reset want-reset-commit-id

[git@lvao yument]$ git status

[git@lvao yument]$ git add .

[git@lvao yument]$ git stash

[git@lvao yument]$ git pull upstream master

[git@lvao yument]$ git stash pop

[git@lvao yument]$ git diff

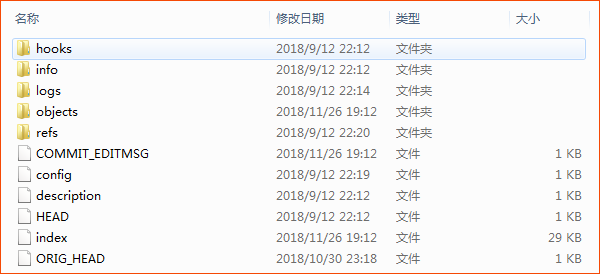
[git@lvao yument]$ git add .

[git@lvao yument]$ git commit – m “comment-new testing version”

[git@lvao yument]$ git push origin master

八、.git/ 目录解析

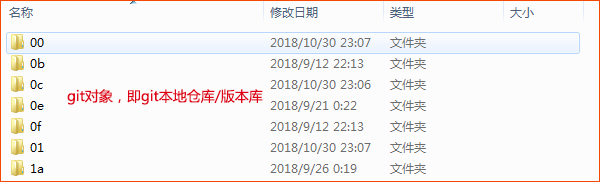
1、目录实例图



2、各文件目录说明

|  |  |
| --- | --- |
| 目录 | 说 明 |
| hooks | 该目录为钩子目录，文件夹下存放着一些shell脚本 |
| info | exclude：存放仓库的一些信息，可以在该文件中添加需要忽略提交的文件 |
| logs | 保存所有更新的引用记录，分 HEAD | (refs: heads, remote) |
| objects | 存放所有的git对象, 版本库所在目录 |
| refs | heads：保存当前最新的一次提交的哈希值 ，分(heads,remotes, tags) |
| COMMIT\_EDITMSG | 最新提交的一次Commit Message，git系统不会用到，给用户一个参考 |
| config | git仓库配置文件 |
| description | 库的描述信息，主要给gitweb等git托管系统使用 |
| HEAD | 映射到ref引用，能够找到下一次commit的前一次哈希值 |
| index | 暂存区（stage），一个二进制文件 |
| ORIG\_HEAD | HEAD指针的前一个状态，在logs/HEAD中，最后一条提交记录的当前hash |

Object目录图：



refs目录图：



九、git本地服务器搭建

十、eclipse git插件使用