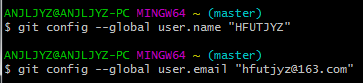
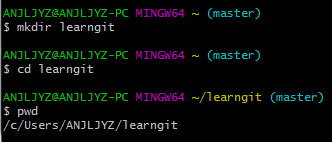
在开始菜单里找到“Git”->“Git Bash”, 自报家门：



创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：



通过git init命令把这个目录变成Git可以管理的仓库：



瞬间Git就把仓库建好了，而且告诉你是一个空的仓库（empty Git repository）

把文件添加到版本库：

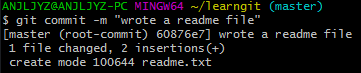
1、在learngit目录或者其子目录下创建一个文件，如readme.txt

一定要放到learngit目录下（子目录也行），因为这是一个Git仓库，放到其他地方Git再厉害也找不到这个文件。

2、用命令git add告诉Git，把文件添加到仓库：



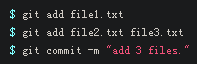
3、用命令git commit告诉Git，把文件提交到仓库：



-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。

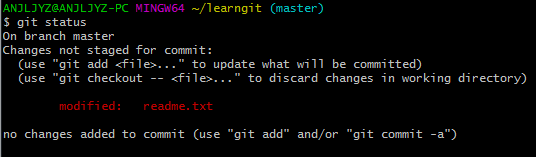
git commit命令执行成功后会告诉你，1 file changed：1个文件被改动（我们新添加的readme.txt文件）；2 insertions：插入了两行内容（readme.txt有两行内容）。

commit可以一次提交很多文件，所以你可以多次add不同的文件：



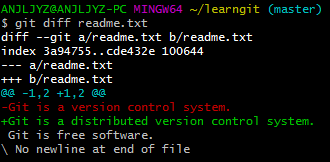
已经成功地添加并提交了一个readme.txt文件，继续修改readme.txt文件

运行git status命令看看结果：



命令显示readme.txt被修改过了，但还没有准备提交的修改

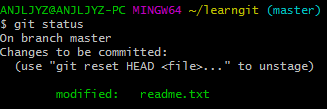
查看修改的内容用git diff <filename>：



提交修改和提交新文件是一样的两步，第一步是git add filename;



运行git status看看当前仓库的状态：



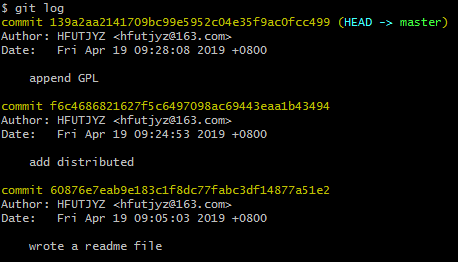
将要被提交的修改包括readme.txt

第二步使用git commit提交：





在Git中，我们用git log命令查看修改内容：



命令显示从最近到最远的提交日志，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：



版本回退：

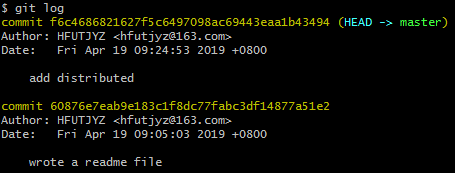
在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

回退使用git reset命令：





readme.txt的内容已经回到add distributed版本了。



这时候使用git log就看不到append GPL 的版本了，可以使用commit id进行reset：

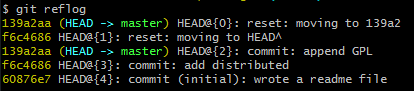




又回到了append GPL的版本



当找不到commit id 的时候，Git提供了一个命令git reflog用来记录你的每一次命令：



Git和其他版本控制系统如SVN的一个不同之处就是有暂存区的概念。

工作区（Working Directory）：就是在电脑里能看到的目录， learngit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。



我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，git commit就是往master分支上提交更改。

git diff 比较的是工作区文件与暂存区文件的区别。

git diff --cached 比较的是暂存区的文件与仓库分支里（上次git commit 后的内容）的区别。

用git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别。

每次修改，如果不用git add到暂存区，那就不会加入到commit中。

命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

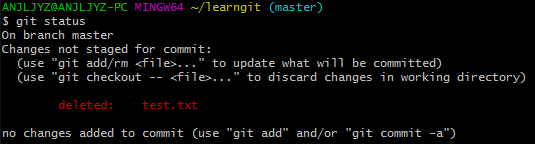
一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

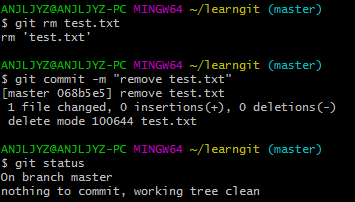
命令git reset HEAD <file>可以把暂存区的修改撤销掉（unstage），重新放回工作区：



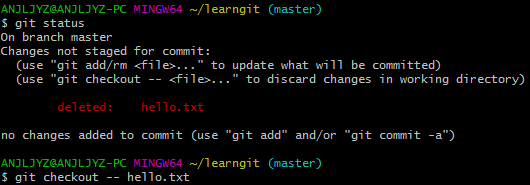
在Git中，删除也是一个修改操作：



现在有两个选择，一是确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit：



另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：



远程仓库

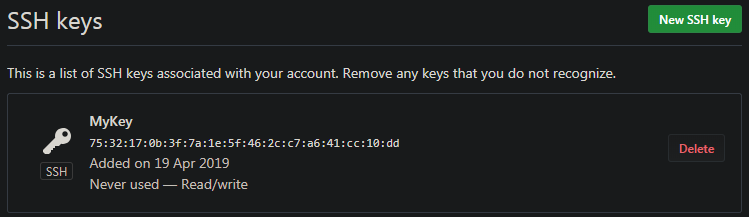
注册GitHub账号，由于本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以，需要一点设置：

第1步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

ssh-keygen -t rsa -C [hfutjyz@163.com](mailto:hfutjyz@163.com)

id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

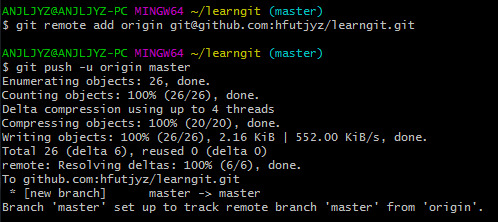
第2步：登陆GitHub，打开“Account settings”，“SSH Keys”页面，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容，然后ADD SSH Key：



创建新的仓库learngit，在GitHub上的这个learngit仓库还是空的，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。

git remote add origin git@github.com:hfutjyz/learngit.git

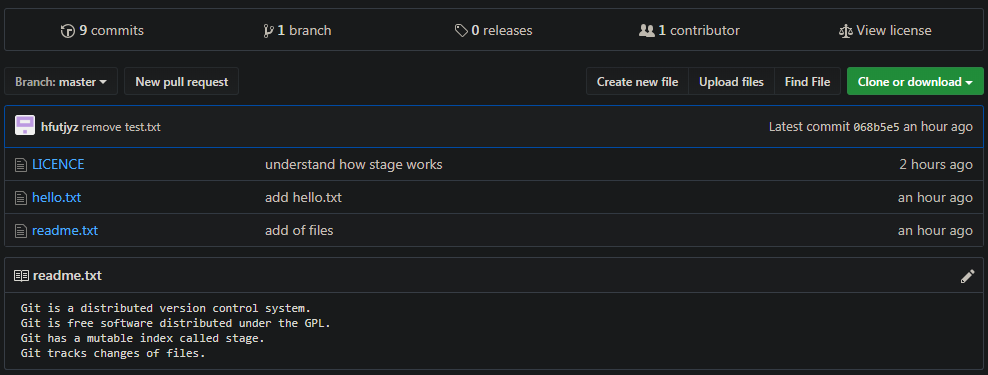
git push -u origin master



把本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程。

由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

推送成功后，可以立刻在GitHub页面中看到远程库的内容已经和本地一模一样：



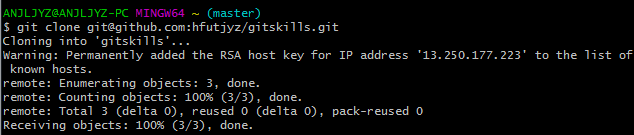
从现在起，只要本地作了提交，就可以通过命令：

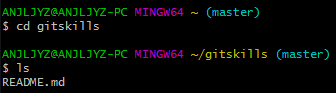
git push origin master

把本地master分支的最新修改推送至GitHub，现在，你就拥有了真正的分布式版本库！

在GitHub创建仓库，然后克隆到本地：

git clone [git@github.com:hfutjyz/gitskills.git](mailto:git@github.com:hfutjyz/gitskills.git)



成功克隆到本地

分支管理

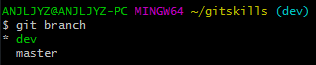
创建dev分支，然后切换到dev分支：



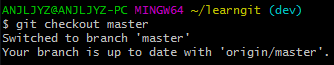
git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：



git branch命令会列出所有分支，当前分支前面会标一个\*号。

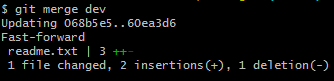


在dev分支上正常提交，然后切换到master分支查看

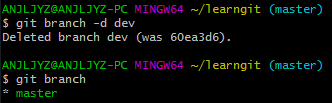


dev分支改变的内容在master中不显示，需要合并：

git merge命令用于合并指定分支到当前分支：



合并完成后，就可以放心地删除dev分支，删除后只剩下master分支：



创建新分支，改变内容后提交，然后切换到master分支，再修改同一份文件，然后提交，由于两次修改不一样，在合并时会出现错误：









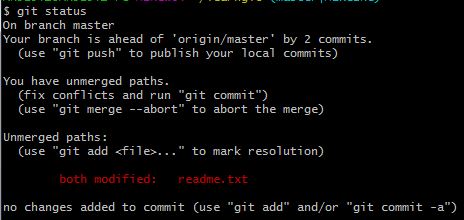
现在，master分支和feature1分支各自都分别有新的提交，变成了这样：



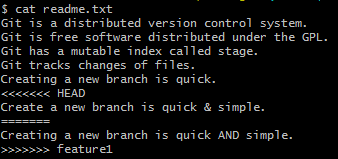
这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突：



git status 会显示冲突的内容：



查看文件内容



Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容

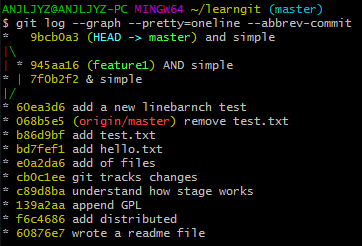
修改之后再提交，



现在，master分支和feature1分支变成了下图所示：



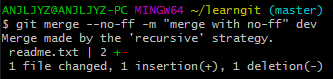
用git log --graph命令可以看到分支合并图。



最后删除feature1分支



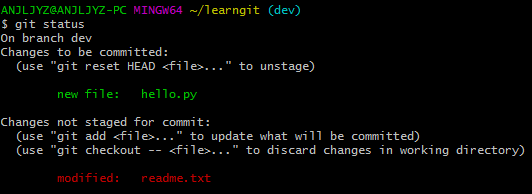
分支合并时使用 –no-ff参数，表示禁用Fast forward:



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。

Bug分支

当前工作的分支没有提交，需要去处理别的Bug时



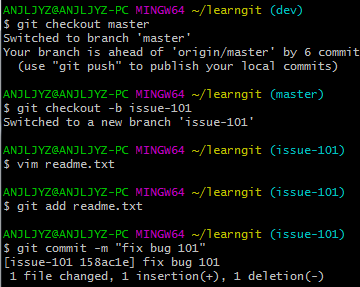
Git提供了一个stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作：



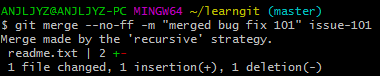
现在，用git status查看工作区，就是干净的（除非有没有被Git管理的文件），因此可以放心地创建分支来修复bug。



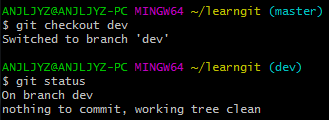
首先确定要在哪个分支上修复bug，假定需要在master分支上修复，就从master创建临时分支：



修复完成后，切换到master分支，并完成合并，最后删除issue-101分支：



回到dev分支：



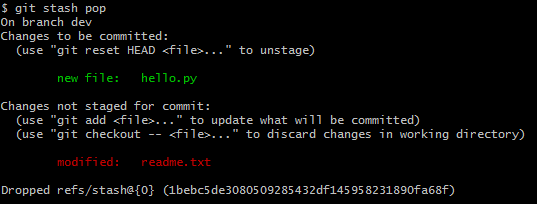
工作区是干净的，刚才的工作现场存到哪去了？用git stash list命令看看：



工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；

另一种方式是用git stash pop，恢复的同时把stash内容也删了：



再用git stash list查看，就看不到任何stash内容了：

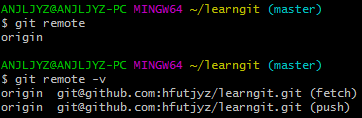


可以多次stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令：

git stash apply stash@{0}

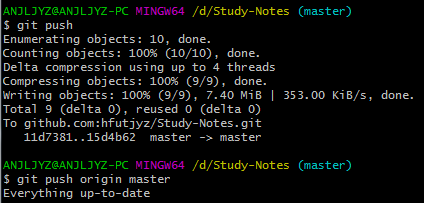
如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

当你从远程仓库克隆时，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。或者用git remote -v显示更详细的信息：

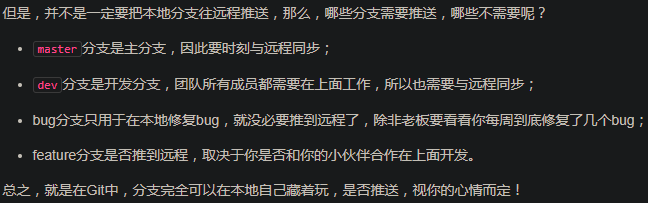


上面显示了可以抓取和推送的origin的地址。如果没有推送权限，就看不到push的地址。

推送分支，就是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就会把该分支推送到远程库对应的远程分支上：

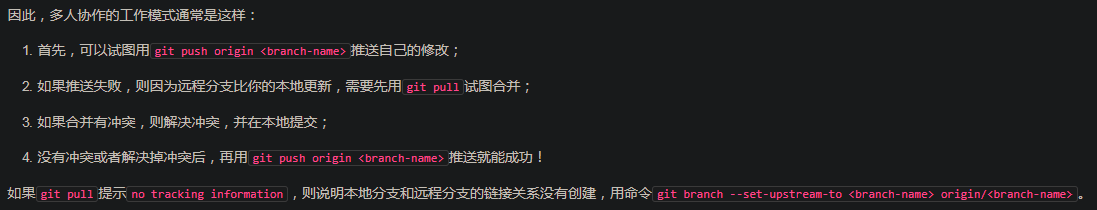


如果要推送其他分支，比如dev，就改成：git push origin dev



创建远程origin的dev分支到本地: git checkout -b dev origin/dev

这样就可以在dev上继续修改，然后，时不时地把dev分支push到远程

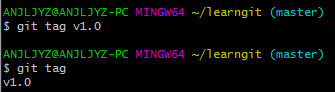


rebase操作可以把本地未push的分叉提交历史整理成直线；

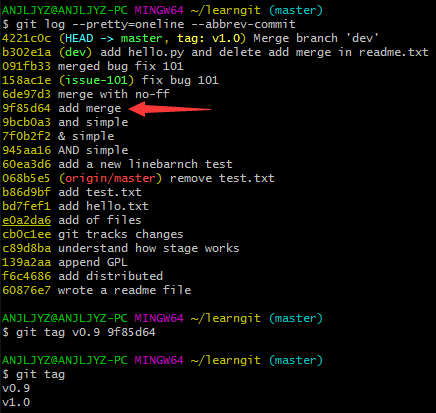
rebase的目的是使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比。

标签可以方面的区分版本：

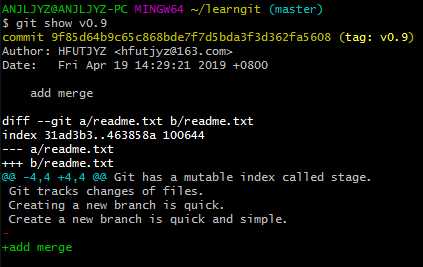
命令git tag <name>就可以打一个新标签：



可以给之前的commit打标签，只需要找到对应的commit id即可

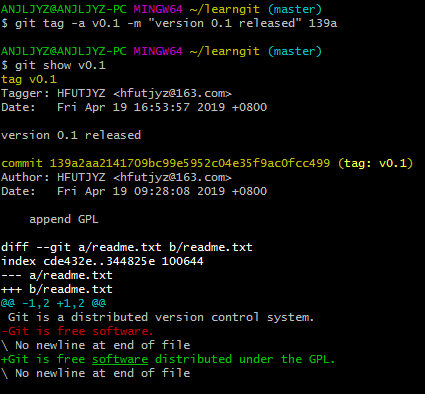


可以用git show <tagname>查看标签信息：



可以看到，v0.9确实打在add merge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：



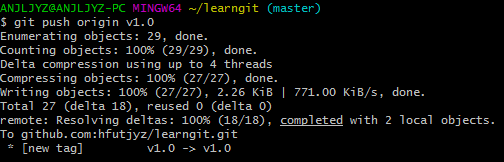
标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签。

删除标签

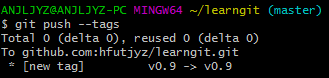


因为创建的标签都只存储在本地，不会自动推送到远程。所以，打错的标签可以在本地安全删除。

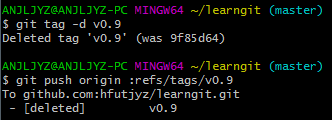
如果要推送某个标签到远程，使用命令git push origin <tagname>：



一次性推送全部尚未推送到远程的本地标签：



如果标签已经推送到远程，要删除远程标签就麻烦一点，先从本地删除，然后，从远程删除：



自定义Git