**1. Git**

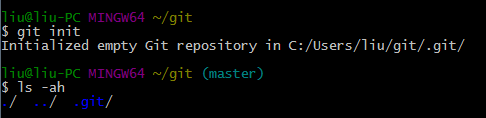
**1.1 创建版本库repository**

1.1.1首先，选择一个合适的地方，创建一个空目录：



pwd命令用于显示当前目录。在我的Windows上，这个仓库位于/Users/liu/git。

1.1.2第二步，通过git init命令把这个目录变成Git可以管理的仓库, 如果你没有看到.git目录，那是因为这个目录默认是隐藏的，用ls -ah命令就可以看见。



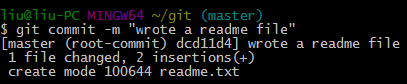
1.1.3把文件添加到版本库

1.1.3.1现在我们编写一个readme.txt文件放到git目录下

1.1.3.2用命令git add告诉Git，把文件添加到仓库：（可反复多次使用，添加多个文件；）



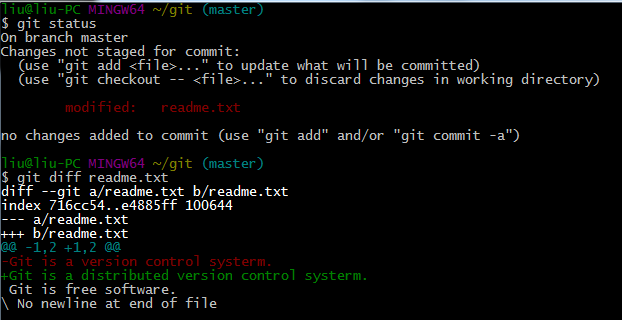
1.1.3.3第二步，用命令git commit告诉Git，把文件提交到仓库：



**1.2 时光穿梭机**

要随时掌握工作区的状态，使用git status命令。

如果git status告诉你有文件被修改过，用git diff可以查看修改内容。



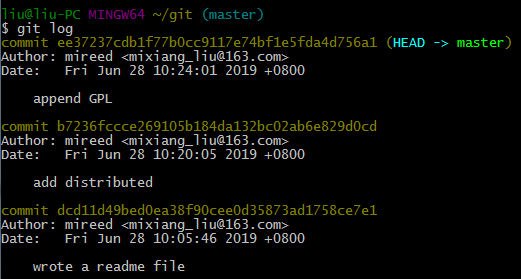
1.2.1版本回退

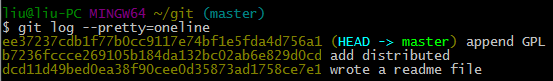
HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。（在Git中，用HEAD表示当前版本，也就是最新的提交1094adb...（注意我的提交ID和你的肯定不一样），上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。）若命令窗口没关，可以用ID找回



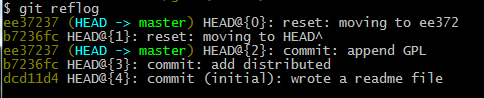


穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline参数





要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。



1.2.2工作区和暂存区

Git add 是把文件放到暂存区

Git commit -m “ ”是把暂存区的文件提交到分支

1.2.3管理修改

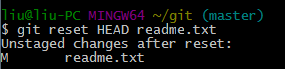
Git管理的是修改，每一次都要Git add

1.2.4撤销修改

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。



场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。





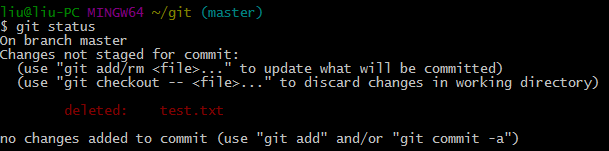
场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

1.2.5删除文件

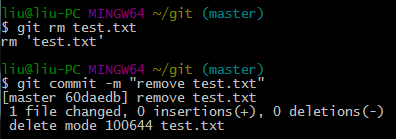
一般情况下，你通常直接在文件管理器中把没用的文件删了，或者用rm命令删了：



这个时候，Git知道你删除了文件，因此，工作区和版本库就不一致了，git status命令会立刻告诉你哪些文件被删除了：



现在你有两个选择，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：



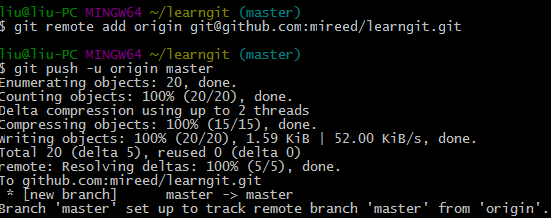
**1.3 远程仓库**

1.3.1添加远程库

要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；

关联后，使用命令git push -u origin master第一次推送master分支的所有内容；

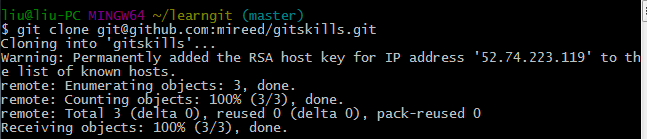
此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；



1.3.2从远程库克隆

要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。

Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。



**1.4 分支管理**

1.4.1创建与合并分支

查看分支：git branch



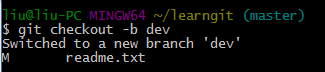
创建分支：git branch <name>



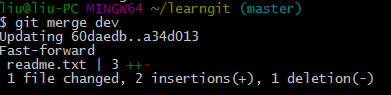
切换分支：git checkout <name>



创建+切换分支：git checkout -b <name>



合并某分支到当前分支：git merge <name>



删除分支：git branch -d <name>



1.4.2解决冲突

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。

用git log --graph命令可以看到分支合并图。

1.4.3分支管理策略

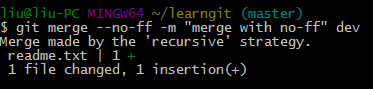
在实际开发中，我们应该按照几个基本原则进行分支管理：

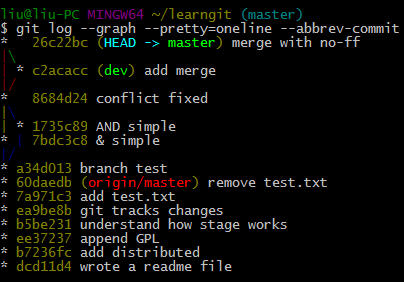
首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

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。

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。





1.4.4BUG分支

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。

1.4.5Feature分支

开发一个新feature，最好新建一个分支；

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。

1.4.6多人协作

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

多人协作的工作模式通常是这样：

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

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；

如果合并有冲突，则解决冲突，并在本地提交；

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-to <branch-name> origin/<branch-name>。

1.4.7 rebase

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。

**1.5 标签管理**

命令git tag <tagname>用于新建一个标签，默认为HEAD，也可以指定一个commit id；

命令git tag -a <tagname> -m "blablabla..."可以指定标签信息；

命令git tag可以查看所有标签。

命令git push origin <tagname>可以推送一个本地标签；

命令git push origin --tags可以推送全部未推送过的本地标签；

命令git tag -d <tagname>可以删除一个本地标签；

命令git push origin :refs/tags/<tagname>可以删除一个远程标签