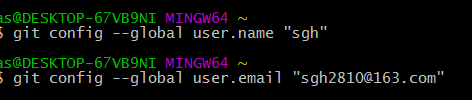
1. 申明用户名以及邮箱

$ git config --global user.name "XXXX"

$ git config --global user.email "XXXX"



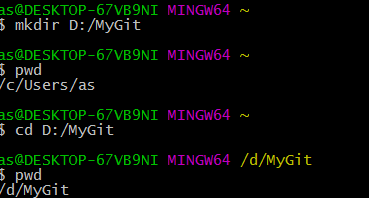
1. 创建并查看仓库

1.创建一个空的目录

$ git mkdir “路径名” //路径中不能含有汉字

$ cd “路径名” //将当前目录从默认的目录转移到目标路径（仓库路径）

$ pwd //用来显示当前路径，



2.将创建的目录变成可以使用git管理的仓库（一定要将当前目录转移到目标目录，在目标目录下执行此命令才能将目标目录变成可管理的仓库）

$ git init

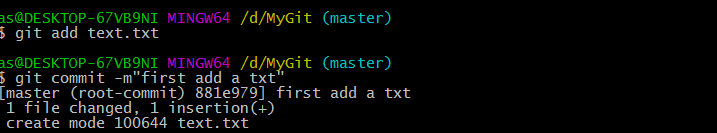
此命令结束后会在路径下创建一个.git目录，不要手动更改此目录的内容，这是用来跟踪管理仓库的，如果看不到这个路径可以通过执行 ls –ah



3.往仓库里面添加东西

$ git add 文件名 //文件一定要放在仓库的路径下，这样git才能找到，只有转移到仓库的路径后才可以进行添加等操作

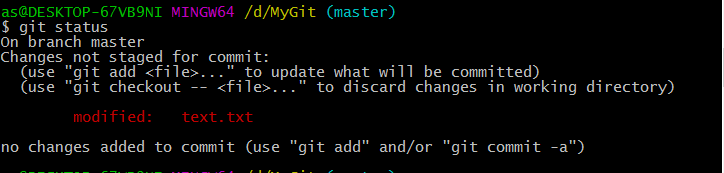
$ git commit –m “本次改动的说明“（在commit命令之前可以执行多个add命令）



1. 更改仓库里的文件的内容，提交并查看更改的内容
2. 在仓库里直接更改文件保存后执行下面命令

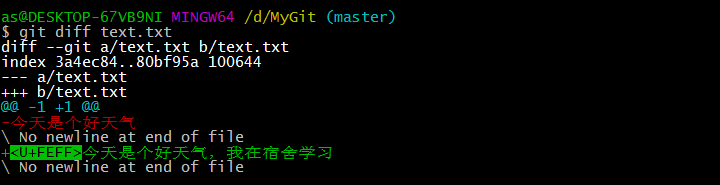
$ git status //查看工作区当前状态

执行完此命令后可以知道文件被更改过了，并不能知道改动的内容，status是工作区的状态，因为git实际上管理的是更改，也就是只有工作区有修改或者修改后只add并没有提交时才会有状态信息的显示，如果没有就显示工作区是clean干净的



1. 查看更改的内容

$ git diff 文件名 //diff就是difference的意思，执行后可以看到更改前后的句子



1. 提交

$ git add 文件名

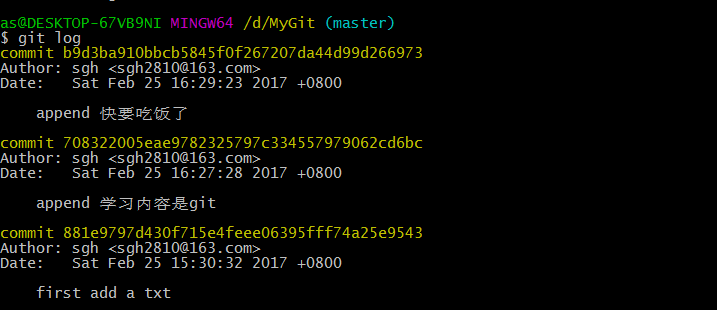
$ git commit –m ”注释内容”

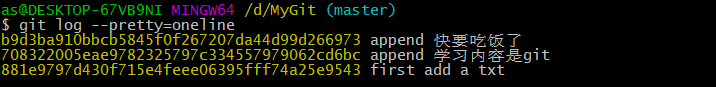
不执行这两个命令的话仓库并不能记录你的更改

1. 查看更改文件的次数，并恢复到某一个版本的文件
2. 查看都有过哪些改动

$ git log

直接执行此操作后可以清楚的看到更改的注释（就是执行commit –m “”时添加的注释）以及更改的时间，如果嫌显示的信息让人眼花缭乱，可以在命令的后面加上 --pretty=oneline 参数，顾名思义每次记录改动的信息都在一行显示出来（执行完此命令后会有一大串数字，这些数字是commit id（版本号），这个版本号在恢复到某个版本时会用到）

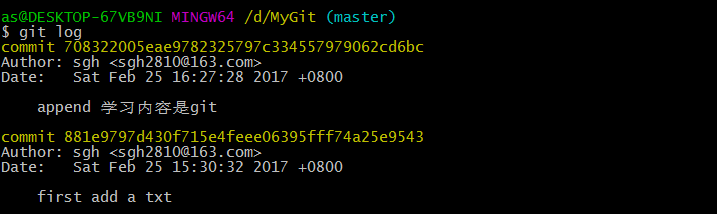




1. 将文件恢复到历史版本

$ git reset –-hard 参数 //参数处可填：HEAD代表当前版本，也就是最新提交的版本；HEAD^代表上一个版本；HEAD^^代表上两个版本；HEAD~n代表之前的第n个版本。执行完此命令后再执行一下git log命令会发现，恢复的版本之后的版本都没了，只有它之前的历史版本，如果还想恢复到那些被删除的版本中的某个可以将参数填成那个版本的commit id（版本号），可以不用写全commit id，毕竟太长了，具体写几位，只要能锁定到那个版本号就行了【如果忘记了版本号commit id，可以通过git reflog 命令查看之前执行的命令信息（包含了在哪个版本下执行的），通过这些信息就可以知道版本号了】





可以看到此时少了一个版本



我们通过commit id恢复了删除了的版本

$ cat 文件名 //此命令可以查看当前版本的文件的内容



五．工作区和暂存区

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。

先来看名词解释。

工作区（Working Directory）

就是你在电脑里能看到的目录，比如我的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。



分支和HEAD的概念我们以后再讲。

前面讲了我们把文件往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。

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

你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

俗话说，实践出真知。现在，我们再练习一遍，先对readme.txt做个修改，比如加上一行内容：

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

然后，在工作区新增一个LICENSE文本文件（内容随便写）。

先用git status查看一下状态：

$ 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: readme.txt*

*#*

*# Untracked files:*

*# (use "git add <file>..." to include in what will be committed)*

*#*

*# LICENSE*

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

Git非常清楚地告诉我们，readme.txt被修改了，而LICENSE还从来没有被添加过，所以它的状态是Untracked。

现在，使用两次命令git add，把readme.txt和LICENSE都添加后，用git status再查看一下：

$ git status

*# On branch master*

*# Changes to be committed:*

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

*#*

*# new file: LICENSE*

*# modified: readme.txt*

*#*

现在，暂存区的状态就变成这样了：



所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。

$ git **commit** -m "understand how stage works"

[master 27c9860] understand how stage works

2 files changed, 675 insertions(+)

**create** mode 100644 LICENSE

一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是“干净”的：

$ git status

*# On branch master*

nothing to commit (working directory clean)

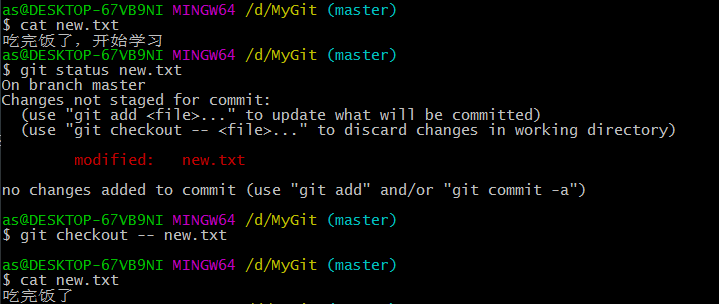
现在版本库变成了这样，暂存区就没有任何内容了：



六．撤销修改

$ git checkout – 文件名 //总是回到上一次add或commit的版本

有两种情况（1）自修后还没有add将其添加到缓存区，执行此命令后回到改之前的状态

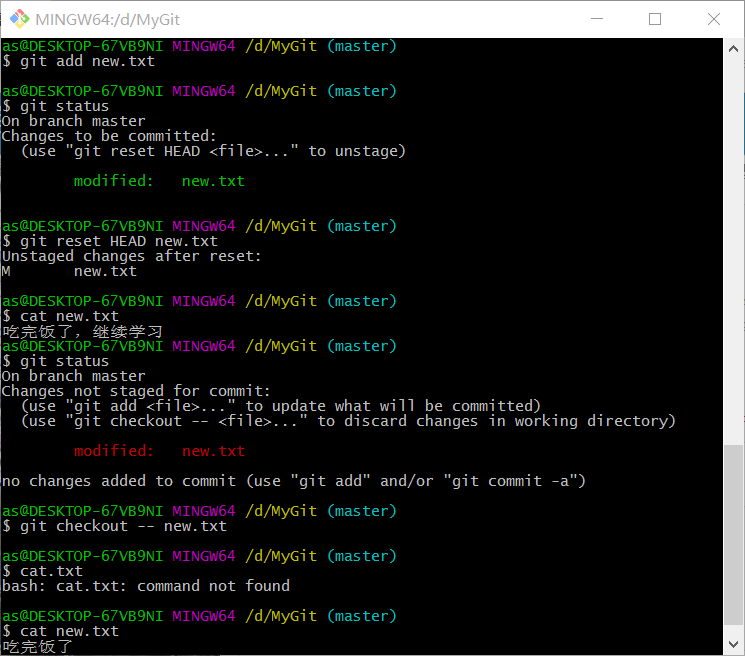


（2）add添加到缓存区后又做了修改，执行后回到刚刚添加到缓存区时的版本。

(3)add添加到缓存区，此时需要先回到添加之前的工作区（此时内容依然是修改了的），然后checkout命令撤销修改

从缓存区回到工作区：

$ git reset HEAD 文件名



当然如果已经提交了修改commit了，那就只能恢复之前版本了reset – hard HEAD^

七．删除操作

1.rm file //该方法删除后无法commit提交，也就是无法添加纪录（实际上也可提交注释，删除后$ git commit –am ““）

2.git rm file

Git commit –m“” //可以通过commit提交添加注释

八．远程库操作

1.创建完远程库以及本地的路径后

$ git remote add origin [git@github.com:yourgithubName/yourRepository(github](mailto:git@github.com:yourgithubName/yourRepository(github)上创建的仓库名).git // 此命令主要是把当前路径与特定的远程仓库连接起来

2.

$ git clone <https://github.com/yourGithubName/Repository>(仓库名).git

执行完此命令后再本地仓库会产生一个和远程仓库名一样的路径(含有一个RANDOM.dm的文件)，之后便可在此路径下传东西了

注：每次add添加完(如果添加的是文件夹在add后加一个点，如 add . 表示将所有子文件都add了)，并且commit提交后

$ git push origin master //执行后上传到服务器