Git基本命令

一：git的命令的认识：与Linux的命令大同小异；

二：cd, mkdir, ll, ls 等这些基础命令都和Linux一致；

三：需求：创建一个本地的仓库，并写入一些内容，同时，将这些内容加入到本地仓库中：

（1）创建一个目录(java\_project)，用命令：mkdir java\_project;

（2）切换到该目录下，用命令：cd java\_project;

在该目录中创建一个reademe.txt的文件，然后在该文件中写入“hello world”，用命令：$ echo "public class hello {}">readme.txt（本来想写个hello world程序但是按下回车直接写入了，所以用了vi命令再次编写）结果证明和Linux命令一致；

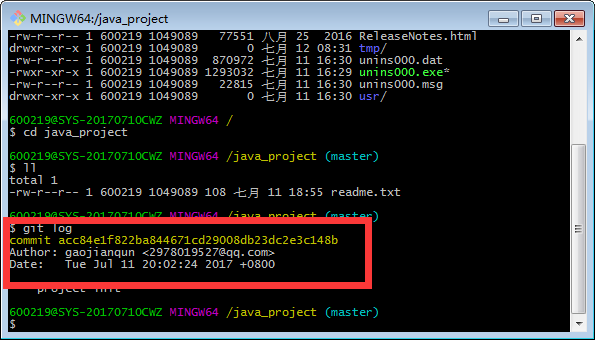
在项目目录中再次创建一个本地仓库，用命令：$ git init 产生的效果：

Reinitialized existing Git repository in D:/Git/Git/java\_project/.git/

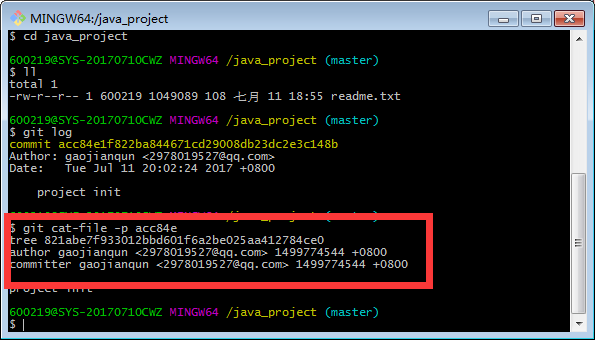
(3)将readme.txt文件添加到暂存区，用命令：$ git add readme.txt；（必须要在该git仓库所在的目录下进行）

(4)创建提交，真正的把readme.txt文件提交到本地的git仓库中，用命令：$ git commit -m 'project init'；

(5)切换到java\_project下，使用命令：$ git log可以看到如下信息，其中注意的是commit后面的信息是一个字符串，这个是一个对象（SHA签名串值，只要对象里面的内容不同，那么我们就可以认为对象的名字不会相同，反之也成立。）：

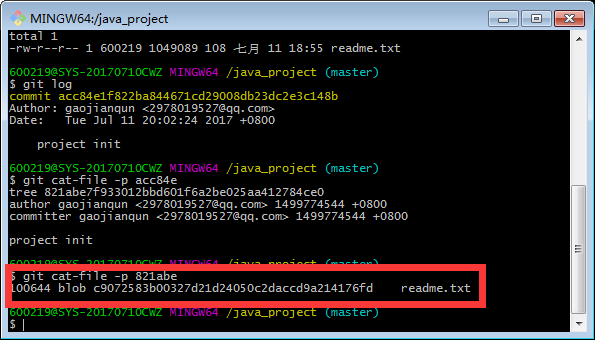


（6）具体看一下这个对象的具体内容，用命令：$ git cat-file -p acc84e(只需前六位即可)发现如下内容：



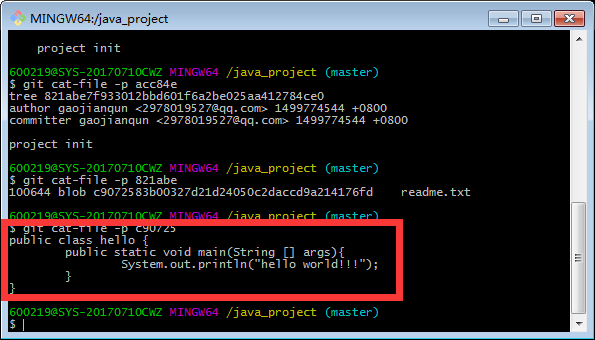
可以看到：提交“58b53c” 是引用一个名为“2bb9f0”的树对象（tree）。一个树对象（tree）可以引用一个或多个二进制对象（blob）, 每个二进制对象都对应一个文件。 更进一步, 树对象也可以引用其他的树对象，从而构成一个目录层次结构。

（7）具体再看一下这个树对象下的数据结构，用命令：$ git cat-file -p 821abe；



不难看出，2bb9f0”这个树对象（tree）包括了了一个二进制对象（blob），对应于我们在前面创建的那个叫 ”readme.txt”的文件。

（8）再次具体看一下这个二进制文件中的具体内容，用命令：$ git cat-file -p c90725



四：需求，如何clone一个远程项目（下载[linux](http://lib.csdn.net/base/linux" \t "_blank" \o "Linux知识库)-2.6的最新版源代码）：

（1）在Java\_project目录下创建一个目录为temp的目录，用命令：$ mkdir temp

（2）切换到temp目录下，开始clone项目，用命令：$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git（由于系统下载的项目名称为linux-2.6.git，所以在temp目录下会产生一个文件为linux-2.6的文件夹）；

五：现象，实际上我们必须要先把已经修改的文件全部存储到暂存区，然后再合并提交。

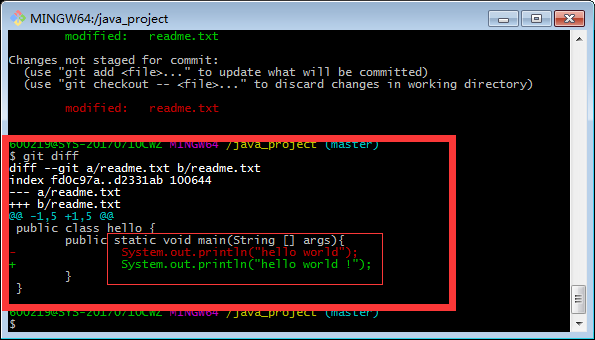
（1）当我们修改一个文件的时候，将readme.txt文件修改(用编辑命令 vi readme.txt)，然后暂存，使用命令：$ git add readme.txt;这个时候会提示：

Changes to be committed:

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

modified: readme.txt

这就说明该文件修改之后已经存储在了暂存区；

（2）如何查看同一份文件存在暂存区和没有存在暂存区的差异，用命令：$ git diff;效果如下，其中红色标记部分为已经存储在暂存区中的部分，标记为绿色的部分为未存储在暂存区中的部分，二者的差异已经用不同颜色标记出来：

六:操作分支

(1)创建一个目录名称为java\_project ,切换到java\_project目录下,用命令：mkdir java\_project ,cd java\_project;

(2)创建一个文件myfile.txt,写入hello world；

(3)初始化仓库，用命令：$ git init;

(4)将myfile.txt文件添加到暂存区，用命令：$ git add myfile.txt;

(5)将文件提交到本地仓库中，用命令：$ git commit –m “first commit”;

(6)创建一个分支，用命令：$ git branch issue1;

(7)将原有的内容拷贝到新创建的分支上，用命令：$ git branch issue1;此时原来在master上的内容拷贝到了issue1上；此时咱们所在的领域是在issue1分支上而非master分支上。

七：并行操作

1. 创建两个分支，issue2,issue3,用命令：$ git branch issue2;$ git branch issue3;

八：连接远程仓库，创建分支，合并分支，合并到开发环境中

(1)从远程仓库中下载程序，用命令：$ git clone <http://172.28.4.12/600219/java_project.git>;将程序下载到本地的work\_code文件夹下；

(2)此时查看一下本地的work\_code文件夹下就可以看到java\_project项目，在该项目下写入文件，切换到此文件夹下，用命令：$ echo public class hello{} >myfile.txt；

(3)写入到暂存区，用命令：$git add myfile.txt;

(4)将myfile文件提交，用命令：$git commit –m “first commit”;

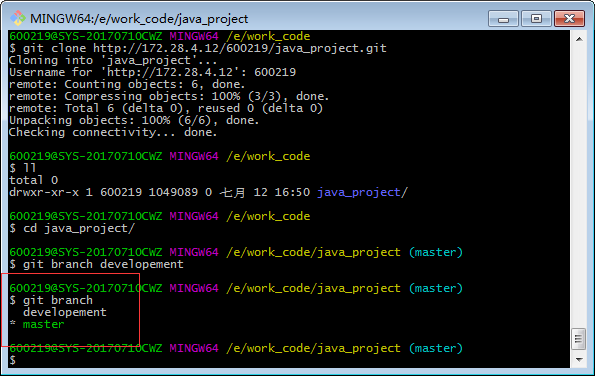
(5)先同步到本地，用命令：$git pull,之后输入用户名和密码；

(6)将myfile.txt文件推送到远程仓库中，用命令：$git push；

(7)重新再从远程仓库中克隆项目，切换到工作空间上，用命令：$git clone <http://172.28.4.12/600219/java_project.git>;将程序下载到本地的work\_code文件夹下；

(8)下载完毕后，要真正的在上面开发了，所以要先创建一个分支development，用命令：$git branch development；

(9)查看一下分支的状态，用命令：$git branch；

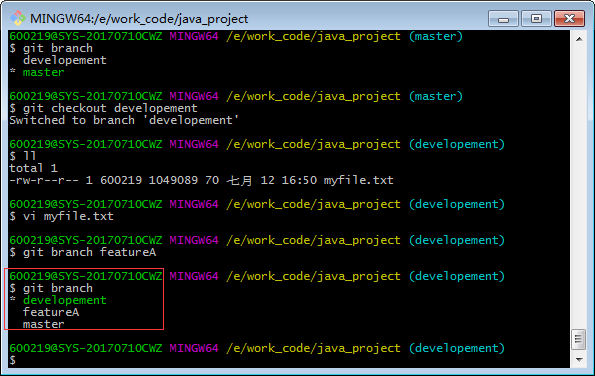


此时有两个分支，一个是主分支，master，一个是development分支；

（9）切换到development分支上，用命令：$git checkout developement；

(10)在创建一个分支，用命令：$git branch featureA；(真正的在上面开发一个功能)

(11)查看一下分支的状态，用命令：$git branch；



(12)切换到featureA分支上，用命令：$git checkout featureA；

(13)这个时候可以真正的开发一项功能了，修改myfile.txt文件中的内容，开始合并分支（即将featureA中的内容合并到development分支上）:首先现将featureA上的文件存到暂存区，用命令：$git add myfile.txt；然后提交，用命令：$git commit –m “featureA功能开发完毕”;

(14)合并分支featureA ,先切换到development分支，用命令：$git checkout developement；

(15)正式开始合并featureA分支，然后用命令：$git merge featureA；

(16)查看一下featureA的内容是否合并到developement分支上,vi编辑一下即可；

(17)直接将内容存储到暂存区即可，用命令：$git add myfile.txt；

(18)继续向上切换到master分支上，然后进行合并，用命令：$git checkout master；

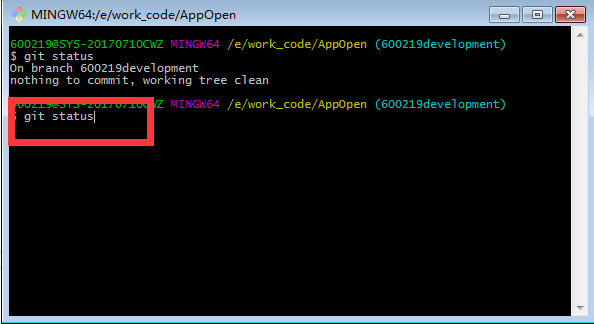
(19)合并development分支，用命令：$git merge developement；

(20)存储到缓存区，用命令：$git add myfile.txt；

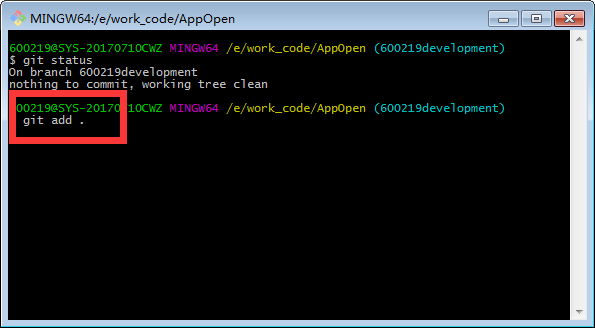
(21)之后开始推送到远程仓库中，用命令：$git push；

提交程序

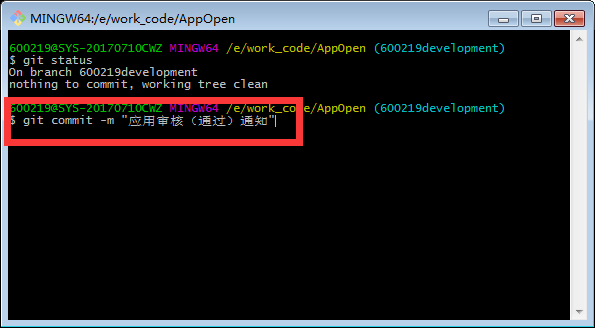
1. 查询有没有可提交文件：$git status



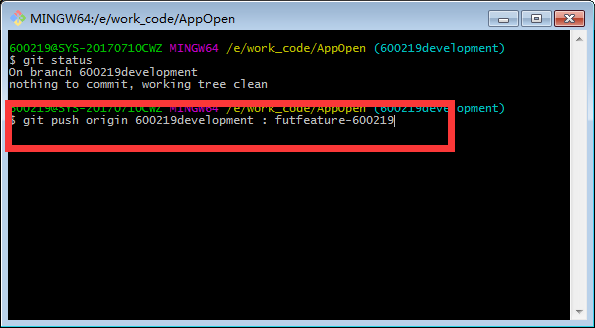
1. 没有冲突直接将所有文件添加到暂存区：$git add .



1. 然后提交：$git commit –m “说明”;



1. 将本地分支合并到远程分支上：$git origin 本地分支名:远程分支名



**重新git命令大全**

1. **使用git命令删除本地分支：**

**git branch –d branchName**

1. **删除远程分支：**

**git push origin :branchName**

1. **查看远程分支：**

**git branch –r**

1. **从远程上拉取代码**
2. **切换到你所在的那个分支(2)git pull**
3. **远程覆盖本地文件**

**git fetch --all**

**git reset --hard origin/develop**

1. **如果push到远程分支之后，如果回滚需要如下两个命令：**

**本地回滚到所有提交之前**

git revert 215d7c24a101ff3b51d638b38d768f05920fc848(版本号)

**强制将本地回滚到提交的那个状态**

git reset --hard 215d7c24a101ff3b51d638b38d768f05920fc848(版本号)

**7.**假如现在的Bug还没有解决，而又出现了一个新的Bug结果切换另一个分支**当执行 $ git checkout master 命令的时候，将提示出错：**

**error: Your local changes to the following files would be overwritten by checkout:**

**readme.txt**

**Please commit your changes or stash them before you switch branches.**

1. ****将当前分支藏起来：****

****$ git stash****

1. ****之后切换其他分支：git checkout ......即可；****
2. ****修改完bug以后，解除分支“封印”，切回到原来分支；执行命令：****

****$git stash pop****

# **8.将远程git仓库里的指定分支拉取到本地（本地不存在的分支）**

****(1)git checkout -b 本地分支名 origin/远程分支名****

****$ git checkout -b local/dev2 origin/dev2****

****(2)若成功，将会在本地创建新分支dev2,并自动切到dev2上。****

****如果出现提示：****

****fatal: Cannot update paths and switch to branch 'dev2' at the same time.****

****Did you intend to checkout 'origin/dev2' which can not be resolved as commit?****

****表示拉取不成功。我们需要先执行****

****git fetch****

****(3)然后再执行****

****git checkout -b 本地分支名 origin/远程分支名****

****$ git checkout -b local/dev2 origin/dev2****