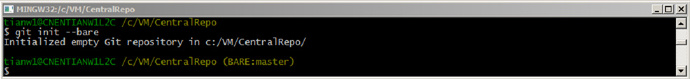
Git远程仓库

# Git远程仓库命令

## 建立中心服务器

通过前面的文章我们知道可以通过"git init"来建立一个Git仓库，这里，我们使用"git init –bare"来建立中心仓库，注意"--bare"参数，后面进行介绍。到这里，一个空的中心服务器就建好了。现在我们看看建立中心服务器时候用到的"--bare"选项。"git init --bare"方法创建的是一个裸仓库，之所以叫裸仓库是因为这个仓库只保存Git历史提交的版本信息，而不允许用户在上面进行各种git操作。

之所以有裸仓库，是因为用"git init"初始化的版本库，用户也可以在该目录下执行所有git方面的操作。但别的用户在将更新push上来的时候容易出现冲突。在这种情况下，最好就是使用" --bare"选项建立一个裸仓库。



## Clone一个仓库

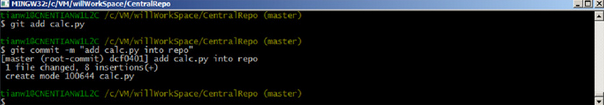
在Git中，我们有两种方式建立Git仓库：一个是通过"git init"建立一个新的仓库，另一个是通过"git clone"命令clone一个已有的Git仓库。

既然中心服务器，用户will就可以通过clone命令来复制一个Git仓库。从上面的clone操作可以看到，在clone命令中需要给出目标仓库的地址，Git支持http、ssh以及git原生协议来进行clone。这里为了演示，我们就使用本地目录。

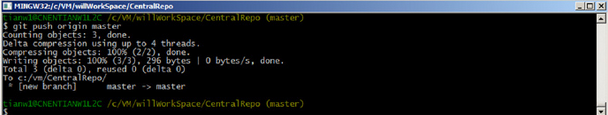


## 更新的push和pull

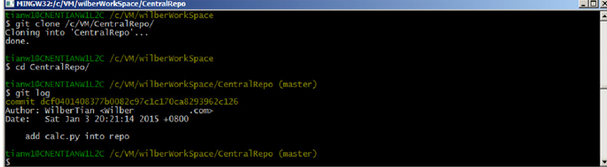
现在will在本地仓库中添加了一个"calc.py"的文件，并且提交到了本地仓库。



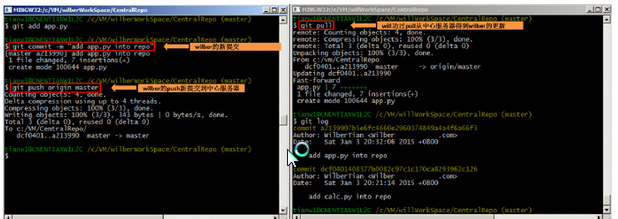
为了是其他用户可以得到这个更新，will需要把这个更新push到中心服务器上。



现在用户wilber通过clone方式获得了中心服务器上的仓库副本，通过"git log"will的更新已经自动被clone了下来。

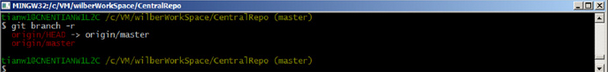


现在，wilber提交了一个新的更新，那么will就可以通过pull的方式从中心服务器得到这个更新。



# 上游仓库(upstream repository)

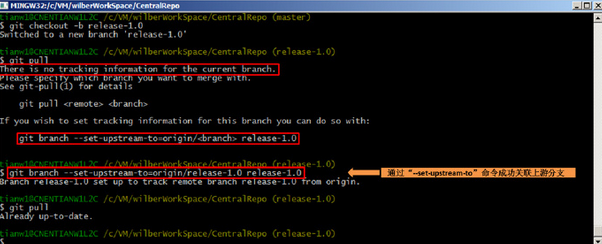
在Git系统中，通常用"origin" 来表示上有仓库。我们可以通过 "git branch -r"命令查看上游仓库里所有的分支，再用 origin/name-of-upstream-branch 的名字来抓取(fetch)远程追踪分支里的内容。



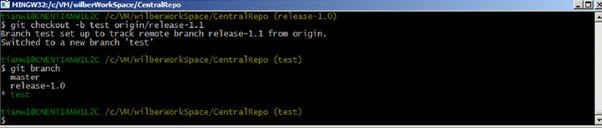
从上面可以看出，对于wilber的仓库，master分支的上游分支是中心服务器的master分支。我们可以在中心服务器上建立两个新的"release-1.0″和"release-1.1″分支，通过pull命令，用户wilber就看到了这些上游分支。

# --set-upstream-to

当我们相当然的按照前一篇文章在本地仓库建立一个branch的时候，我们的pull操作会遇到以下问题，提示我们这个branch没有任何的跟踪信息。仔细想想也是，我们应该把本地仓库中的分支与上游分支关联起来。这时，我们就可以根据Git的提示，使用"--set-upstream-to"命令进行关联了。



当然，在Git中我们可以直接通过"git checkout -b localBranchName origin/remoteBranchName"来创建关联分支。建议一般把"localBranchName"名称跟"remoteBranchName"名称设置成一样。

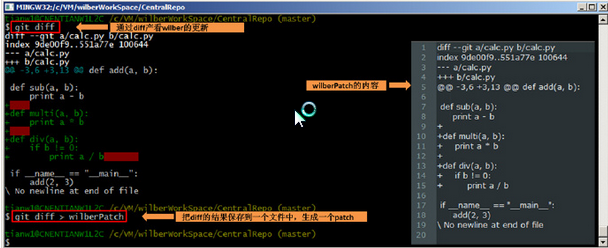


# Patch

在Git中patch绝对是一个很有用的东西。假设在一个没有网络的环境中，wilber和will还要继续工作，这时wilber有一个更新，will需要基于这个更新进行下一步的工作。如果是集中式的代码版本工具，这种情况就没有办法工作了，但是在Git中，我们就可以通过patch的方式，把wilber的更新拷贝给will。在Git中有两种patch的方式：一是通过"git diff"生成一个标准的patch，另一个是通过"git format-patch"生成一个Git专用的patch。

## 基于"git diff"的patch

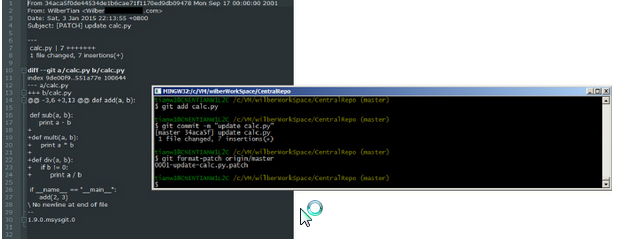
假设现在wilber更新"calc.py"文件并且通过"git diff"生成了一个patch。



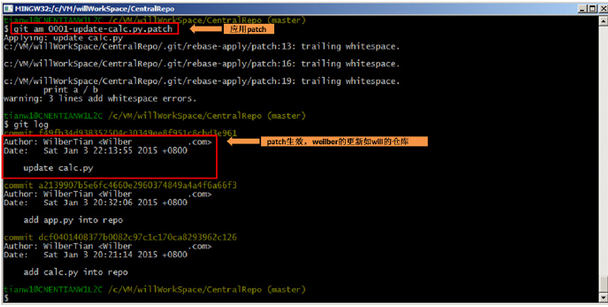
## 基于"git format-patch"的patch

同样使用上面的例子，这次我们使用"git format-patch"来生成patch。

注意"git format-patch"命令的参数"origin/master"，这个参数的含义是，找出当前master跟origin/master之间的差别，然后生成一个patch。



把patch文件拷贝到will的工作目录，则此我们通过"git am"命令来应用这个patch。



## 两种patch方式的比较

patch兼容性："git diff"生成的patch兼容性强。也就是说，如果别人的代码版本库不是Git，那么必须使用git diff生成的patch才能被别的版本控制系统接受。

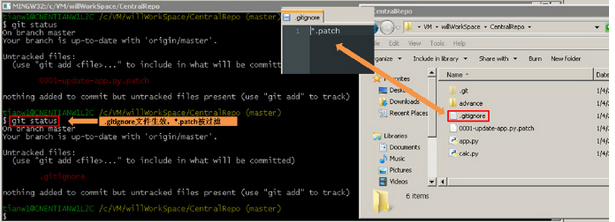
patch合并提示：对于"git diff"生成的patch，你可以用git apply –check 查看patch是否能够干净顺利地应用到当前分支中；如果"git format-patch" 生成的patch不能打到当前分支，git am会给出提示，帮你解决冲突，两者都有较好的提示功能。

patch信息管理：通过"git format-patch"生成的patch中有很多信息，比如开发者的名字，因此在应用patch时，这个名字会被记录进版本库，这样做是比较合理的。

# .gitignore

项目中可能会经常生成一些Git系统不需要追踪(track)的文件，在编译生成过程中 产生的文件或是编程器生成的临时备份文件。我们可以在使用"git add"是避免添加这些文件到暂存区中，但是每次都这么做会比较麻烦。

所以，为了满足上面的需求，Git系统中有一个忽略特定文件的功能。我们可以在工作目录中添加一个叫".gitignore"的文件，来告诉Git系统要忽略哪些文件。



可以看到，当我们使用添加过".gitignore"文件后，文件中的过滤规则就生效了。在windows环境中不支持文件名为".gitignore"，所以可以把文件命名为".gitignore."；.gitignore"文件只会对当前目录以及所有当前目录的子目录生效；也就是说如果我们把".gitignore"文件移到"advance"文件夹中，那么过滤规则就是会对"advance"及其子目录生效了。把".gitignore"文件提交到仓库里，这样其他的开发人员也可以共享这套过滤规则。

过滤语法：

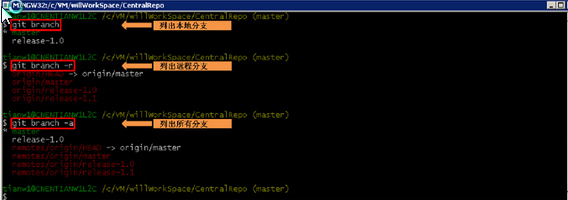
* 斜杠"/"开头表示目录
* 星号"\*"通配多个字符
* 问号"?"通配单个字符
* 方括号"[]"包含单个字符的匹配列表
* 叹号"!"表示不忽略匹配到的文件或目录

exclude文件：在Git仓库中有一个".git/info/exclude"文件，当我们指向对特定的仓库使用特定的过滤规则时，我们可以把过滤语句写在exclude文件中。

|  |
| --- |
| #java项目中常用的配置 |

# 远程仓库命令详解

## git branch



## git remote

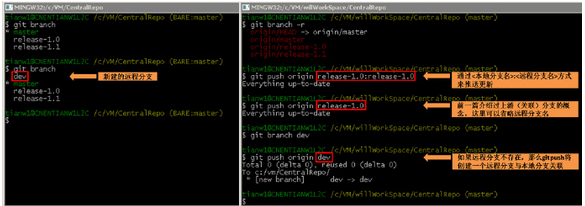


## git push

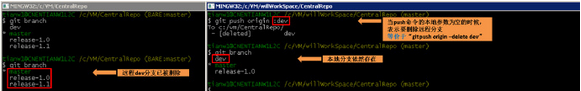
push命令用来将本地分支的更新推送的远程仓库，该命令的格式如下：

git push <远程主机名> <本地分支名>:<远程分支名>

### 通过"git push"更新、创建远程分支



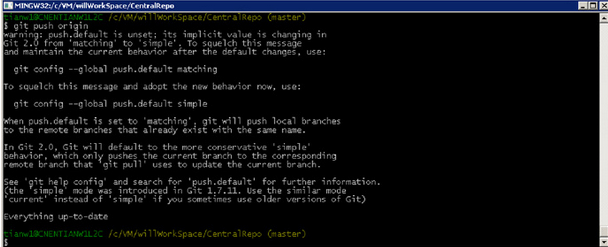
### 通过"git push"删除远程分支



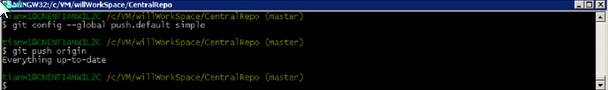
### 省略分支信息的"git push origin"

通过这种方式push的时候，报出了一个警告，提示"push.default"没有设置。在Git中push有两种设置：

* simple方式：只是推送当前分支的更新到对应的远程分支；在Git 2.0以后就默认使用这种方式
* matching方式：会推送所有有对应的远程分支的本地分支



根据Git的提示，我们可以通过"git config --global push.default simple"来设置push方式。



## git pull

pull命令的作用是取回远程某个分支的更新，然后合并到指定的本地分支，pull命令格式如下：git pull <远程主机名> <远程分支名>:<本地分支名>

|  |
| --- |
| #取回origin主机release-1.0分支的更新，与本地的release-1.0分支合并。  一般来说，pull命令都是在关联的本地分支和远程分支之间进行；当然，你可以使用不关联的本地分支和远程分支进行pull操作，但是不建议这么做。  #如果真的需要别的远程分支上的更新，建议使用"cherry-pick"把这个更新拿到关联的远程分支上，然后在关联分支上进行pull操作。 git pull origin release-1.0:release1.0 #表示取回origin/next远程分支的更新，然后合并到当前分支  省略本地分支名：git pull origin release-1.0  #如果当前分支存在上游（关联）分支，可以直接使用git pull origin  #表示本地的当前分支自动与关联的origin主机分支进行合并 |

## git fetch

fetch命令比较简单，作用就是将远程的更新取回到本地。

|  |
| --- |
| #该命令表示将远程origin主机的所有分支上的更新取回本地  git fetch origin  #该命令只取回远程origin主机上master分支上的更新  git fetch origin master |