https://www.cnblogs.com/syp172654682/p/7689328.html

验证git安装成功命令cmd-git



查看配置文件

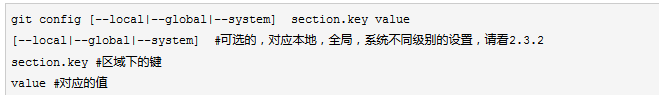
查看系统：git congif --system--list

查看当前用户：git congfig --global --list

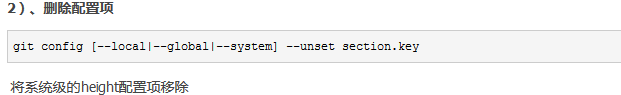
查看当前仓库（项目级）配置信息：git config --local –list

设置当前用户名与邮箱

删除/添加配置项



删除配置项



设置当前的用户名和邮箱

git config --global user.name “yinrigao”

git config –global user.email [1234@163.com](mailto:1234@163.com)

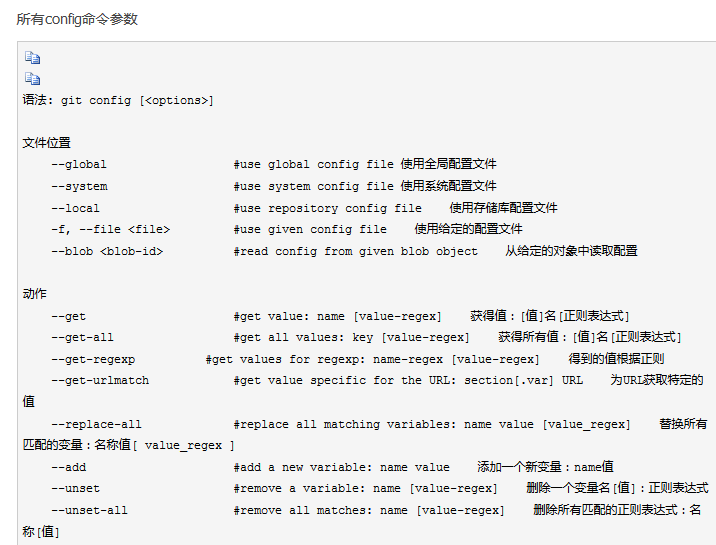
查看当前的用户名及邮箱



查看配置文件的命令是

git congfig --global +资源名称



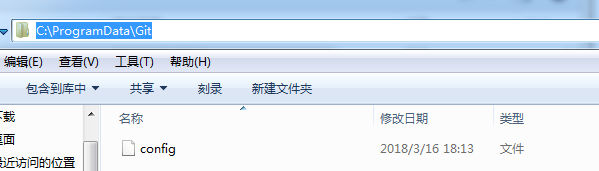


Git 三个配置文件的地址

1系统级的配置文件：system



2只适用于当前登录用户的配置：--global 全局



3位于git项目目录中的.git/config：适用于特定git项目的配置：loacal

同一配置项配置文件的优先级：1<2<3

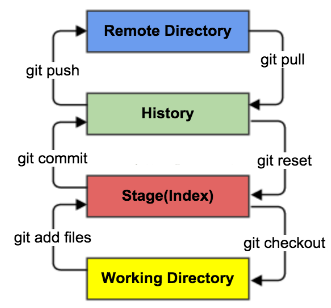
Git理论基础

工作区域

本地工作区域3：工作目录（Working Directory）、暂存区(Stage/Index)、资源库(Repository或Git Directory)

远程的git仓库(Remote Directory)1

四个工作区域的转换如下



Workspace:工作去（平时存代码的地方）

Index/stage: 暂存区，用于临时存放你的改动，事实上它只是一个文件，保存即将提交到文件列表信息

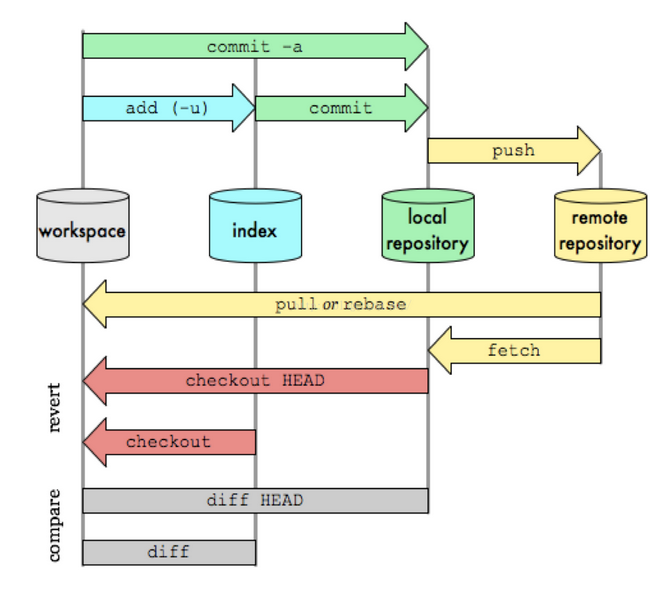
Repository：仓库区（或本地仓库），就是安全存放数据的位置，这里面有你提交到所有版本的数据。其中HEAD指向最新放入仓库的版本

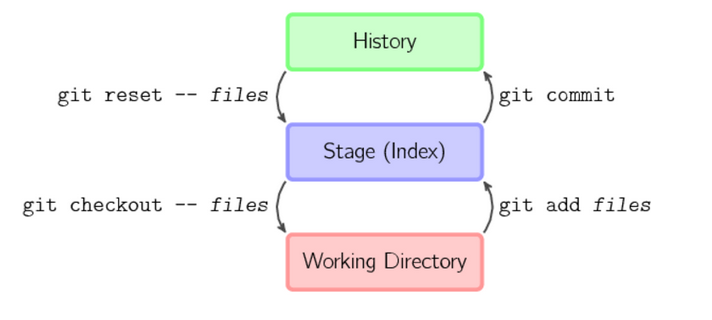
Remote：远程仓库，托管代码的服务器

Git工作流程

在workspace中添加修改文件🡺将需要版本管理的文件放入暂存区域🡺将暂存区的文件提交到git仓库

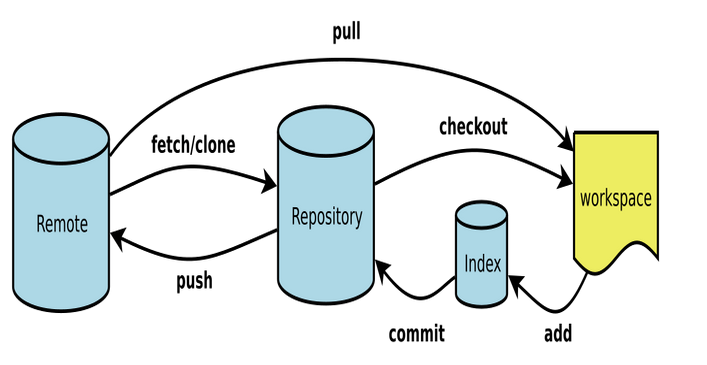
Git管理文件的三种状态：已修改（modified）,已暂存（staged）,已提交(committed)





Git操作

6大命令



Git仓库的创建

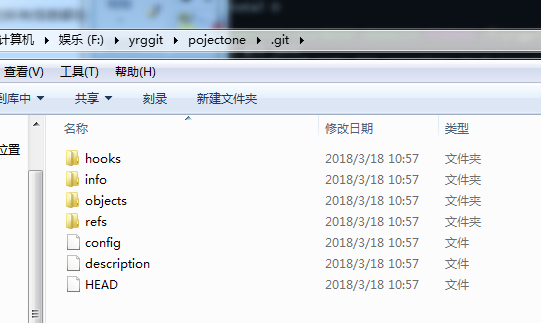
创建全新的仓库

需要用GIT管理的项目的根目录执行：需要git管理名为yrggit的项目命令执行过程

步骤：创建项目文件夹，进入文件后执行git命令

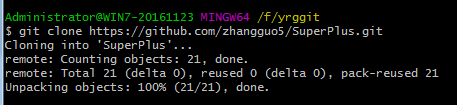


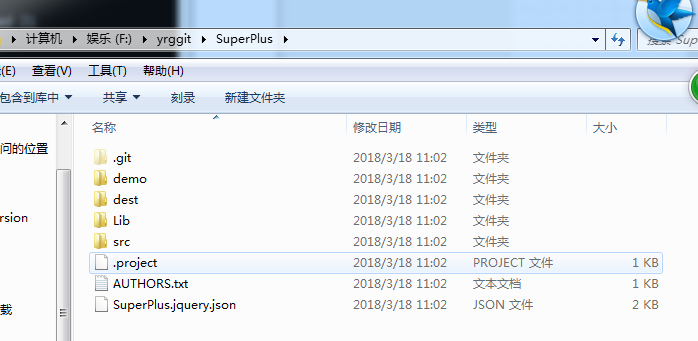
方式二：git init + 项目名称



远程仓库的克隆：将远程服务器上的仓库完全镜像一份至本地，而不是取某一个特定版本

git clone + url(远程服务器仓库地址)

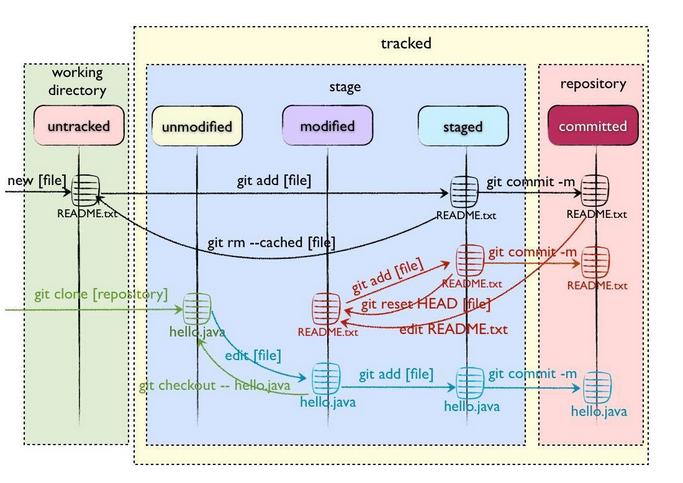




Git 文件操作

Git文件的四中状态



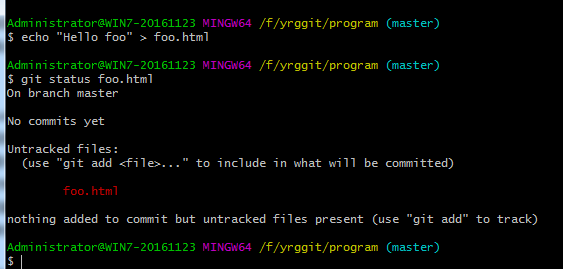


查看文件状态

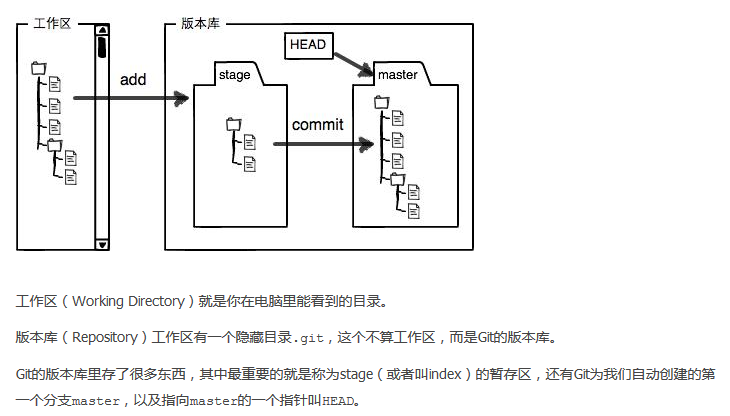
git status filename(查看单个文件状态)/git status（查看所有文件状态）

在已被git管理的文件夹下创建内容为hello foo 的foo.html 文件并查看他的状态

：foo.htm文件的状态为untracked（未跟踪），提示通过git add可以暂存



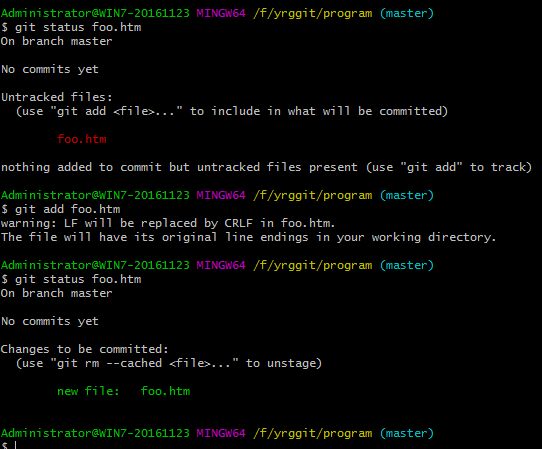
添加文件与目录



添加指定文件到暂存区 git add 文件1 文件2…

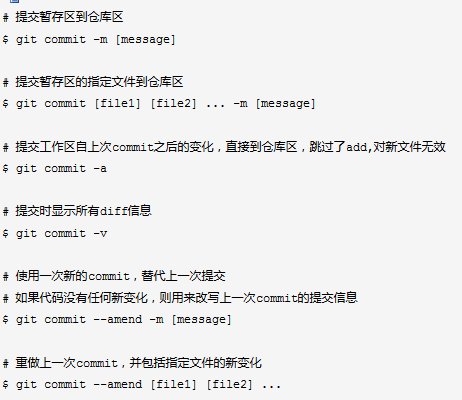
添加指定目录到暂存区包括子目录 git add dir

添加当前目录所有文件到暂存区 git add .

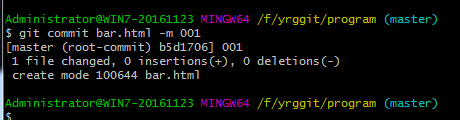


提交文件到本地仓库

通过add只是将文件或目录添加到了index暂存区，使用commit可以实现将暂存区的文件提交到本地仓库



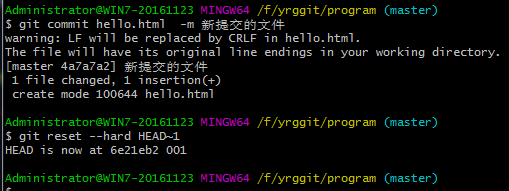
注意meaaage是必须要注明提交信息的





修订提交

撤销上次提交



撤销提交git revert <commit-id>：需要日志文件配合

这条命令会把指定的提交的所有提交回滚，同时形成一个新的提交

日志与历史



查看指定文件的日志



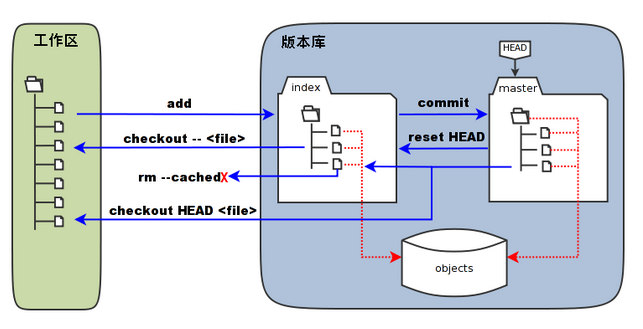
git log --graph"以图形化的方式显示提交历史的关系

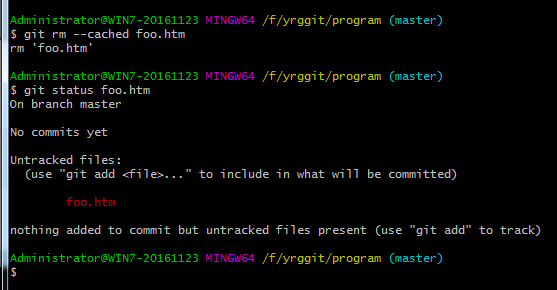
git log -1"则表示显示1行

history可以查看您在bash下输入过的指令

git reflog"中会记录这个仓库中所有的分支的所有更新记录，包括已经撤销的更新

移除文件与目录（撤销add操作，从暂存区删除文件，工作区文件不变）



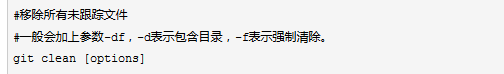
git rm –cached <file> 

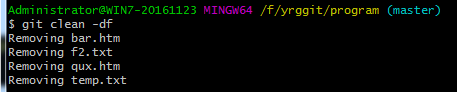
通过重写目录树移除add文件：

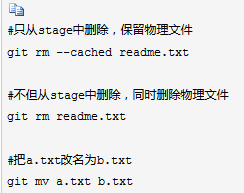
测试没有提交文件不通过git reset HEAD <file>：只有commit的文件才可以执行此命令

当执行 “git reset HEAD” 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

移除所有未跟踪文件：







git rm readme.txt ：删除已经提交的文件

总结

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树

当执行对提交过的文件执行 “git reset HEAD” 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响

只针对暂存区未提交文件

git rm –cached <file>” 命令时，会直接从暂存区删除文件，工作区则不做出改变

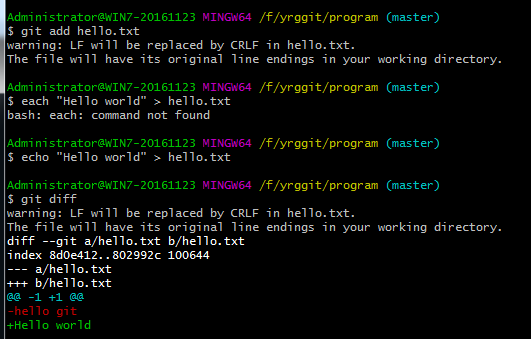
当执行 “git checkout .” 或者 “git checkout — <file>” 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 “git checkout HEAD .” 或者 “git checkout HEAD <file>” 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

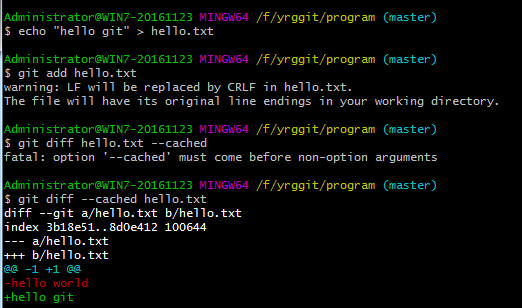
查看文件修改前后的差异

比较工作空间和暂存区文件的差异

git diff + 文件名 用于显示WorkSpace中的文件和暂存区文件的差异



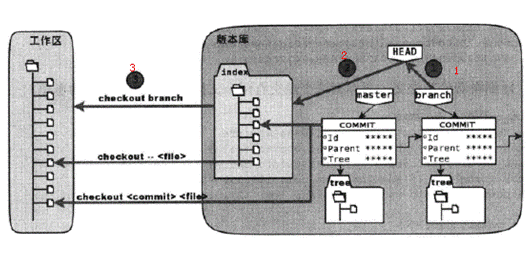
 ---a表示修改之前的文件，+++b表示修改后的文件

比较暂存区的文件与之前提交过文件的差异

比较repo与工作空间中的文件差异

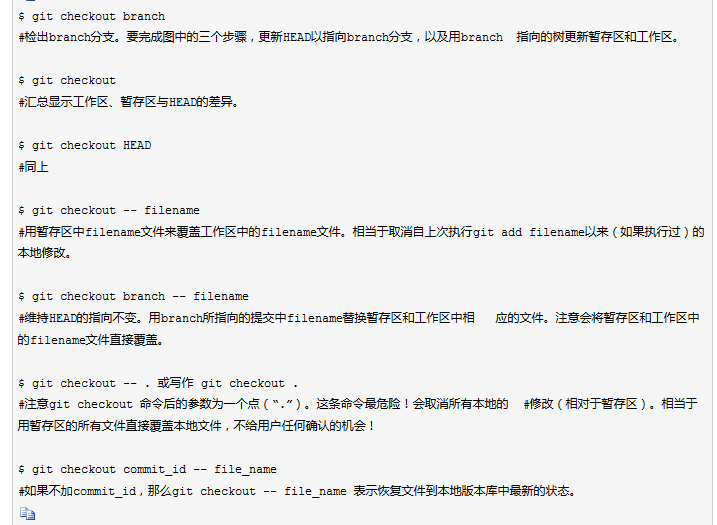
git diff HEAD~n

checkout命令



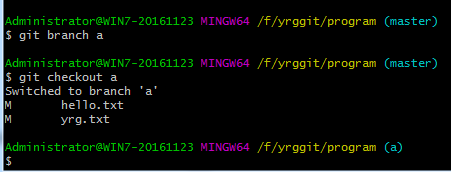
get checkout + 文件名 【get checkout HEAD+文件名】：用暂存区的文件覆盖工作空间的文件（不保留工作空间修改的地方）

git checkout branch + 文件名要完成图中的三个步骤，更新HEAD以指向branch分支，以及用branch 指向的树更新暂存区和工作区即更新所有未commit的



/////////////////////////////////////////////////////////////////////////////////////////////

创建分支 ：git branch 分支名称 切换到新分支：git checkout 分支名称

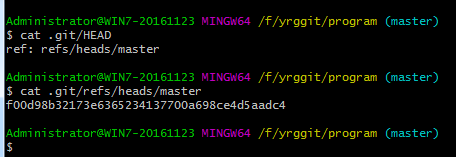


HEAD是checkout的灵魂

当我们身处master（某个）分支的时候，当前的HEAD的内容是什么？

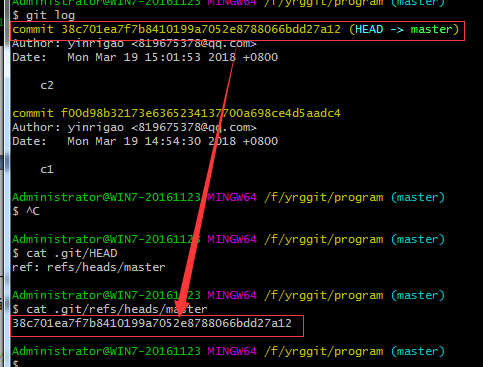
新建一个文件a做一次commit

f00d98b32173e6365234137700a698ce4d5aadc4：第一次提交的hash值



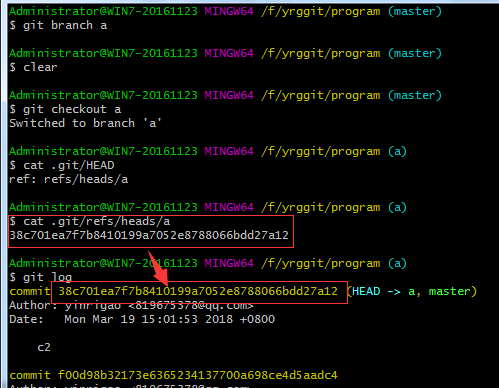
我们看到c1的提交hash值和HEAD对应分支master的当前hash值是一样的。也就是说，HEAD指向的是当前分支名master，而master又对应了当前的最新的一次提交ID

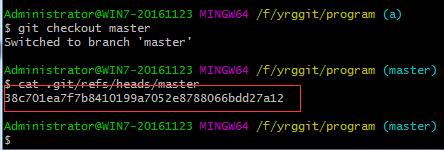
再做一次提交，看看master对应的hash值有无变化



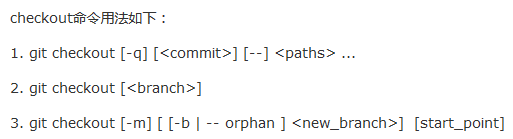
我们可以不难看出，HEAD对应的ref没有变化，还是master，但是master对应的commit ID却变成了c2对应的commit ID，即更新为最后一次提交的ID咯

新建分支a，并切换到a分支



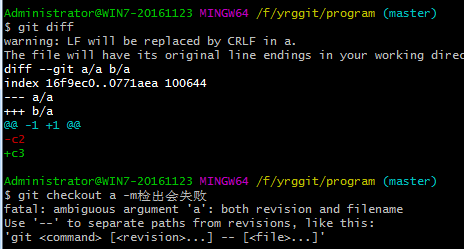


从上图分析，在master分支上的时候，HEAD指向的是master，对应的是c2的commit ID。而切换到a分支的时候，HEAD也相应的指向了a，同时a对应的是a分支上的最新commit ID。因此，我们可以得出结论，在切换分支的时候，HEAD也会相应的指向对应的分支引用



用法2比用法1的区别在于，用法1包含了路径。为了避免路径和引用（或提交ID）同名而发生冲突，可以在<paths>前用两个连续的连字符作为分隔。用法1的<commit>是可选项，如果省略，则相当于从暂存区进行检出。

现在我们处于master分支下，然后我们修改了文件a，输入“c3”文本到a中，这时候，暂存区中的内容是没有"c3"的，通过git diff可以比较。现在我们从当前分支暂存区中检出文件报错



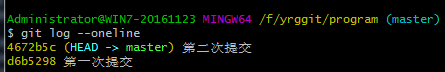
git以为我们想检出仓库a，由于仓库中还存在分支a，同时当前分支中又存在文件a，于是git分不清楚了。这时候怎么办？有两个办法，第一，我们在命名分支的时候要注意语义性，分支名要具有一定的意义；第二，参照用法1，使用两个连字符来分隔

第一种方法不省略commit

在不省略<commit>的时候，<commit>既可以是某一个具体的commit hash值，也可以是某个分支名称，tag名称。不论分支也好，tag也好，它们本质上对应的都是一个commit hash值

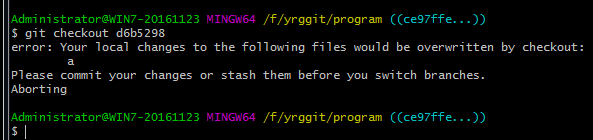
**第1种用法（包含<paths>的用法）不会改变HEAD头指针，主要使用于指定版本的文件覆盖工作区中对应的文件。如果省略<commit>，则会用暂存区的文件覆盖工作区中的文件，否则用指定提交中的文件覆盖暂存区和工作区中的对应文件**

**将文件恢复到最初提交状态**

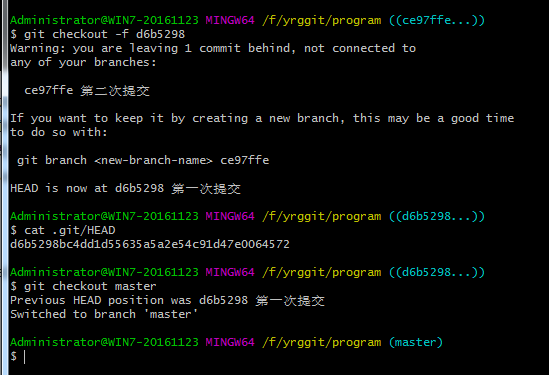




文件在暂存区没有commit检出历史版本报错

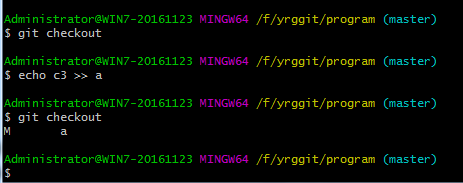


目前我们处于master分支上，且已经有了两次提交，分别是c1和c2，然后我们修改a，给a文件添加内容"c3"，并add到暂存区，随即使用checkout到c1的commit 上。注意，刚开始checkout的时候，git不会允许你直接切换，因为你修改了暂存区的内容， 它会提醒你提交后再切换，这时候，你可以使用-f 强行切换。再查看状态的时候，git提示我们已经不在任何分支上，HEAD指针也是指向具体的c1的commit值，进入了“分离头指针”状态。这个状态下，要回到master上面，只需要git checkout master即可，也可以在这个状态上新建分支



checkou后面不跟任何参数，则就是对工作区进行检查，请看例子。

我们身处master分支上，并且没有任何改动，这时候git checkout没有任何输出。然后，我们给a文件添加内容“c3”，然后再git checkout一下，git就会提示a文件有修改



**总结：对于第2种用法，不是检出某个具体文件的的时候，即不指定<paths>的时候，单纯的检出某个commit或分支，是会改变HEAD头指针的。而且只有当HEAD切换到某个分支的时候才可以对提交进行跟踪，否则就会进入“分离头指针”的状态。如果省略用法2后面的<branch>，则默认对工作区进行状态检查。**

**git branch <branch> <start point>**

**以某个commit创建新分支。 在通常情况下，我们都会在当前分支的基础上，创建新分支。比如git branch new\_branch**

**我们还可以基于当前分支的某一次commit来创建分支**

****

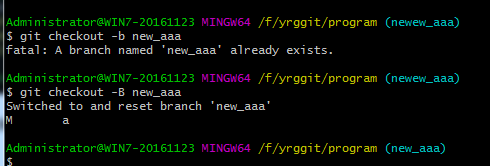
**git checkout –detach <branch>**

**切换到分支的游离状态，默认以该分支下的最后一次提交ID，请看下面的例子。**

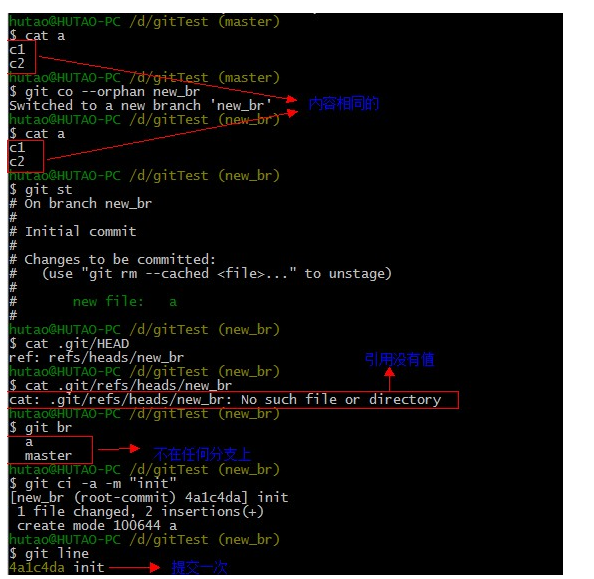
**当前分支为a，然后使用git checkout --detach master，那么HEAD就会切换到master的最后一次commit值的状态下**

****

**git checkout -B <branchname>：强制创建一个可以覆盖已存在的分支**

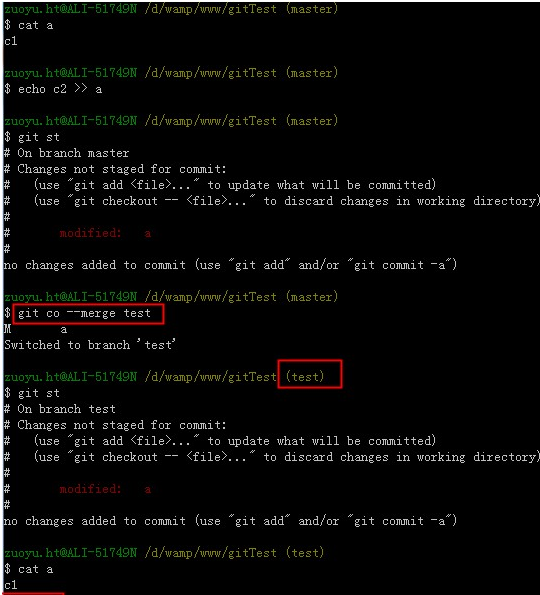


**git checkout --orphan <branch>**

假如你的某个分支上，积累了无数次的提交，你也懒得去打理，打印出的log也让你无力吐槽，那么这个命令将是你的神器，它会基于当前所在分支新建一个赤裸裸的分支，没有任何的提交历史，但是当前分支的内容一一俱全。新建的分支，严格意义上说，还不是一个分支，因为HEAD指向的引用中没有commit值，只有在进行一次提交后，它才算得上真正的分支

**5. git checkout --merge <branch>**

这个命令适用于在切换分支的时候，将当前分支修改的内容一起打包带走，同步到切换的分支下。



有两个需要注意的问题。

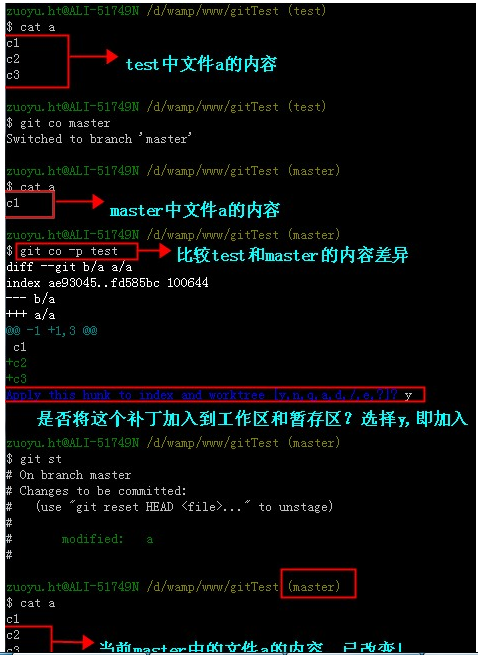
第一，如果当前分支和切换分支间的内容不同的话，容易造成冲突。

第二，切换到新分支后，当前分支修改过的内容就丢失了。

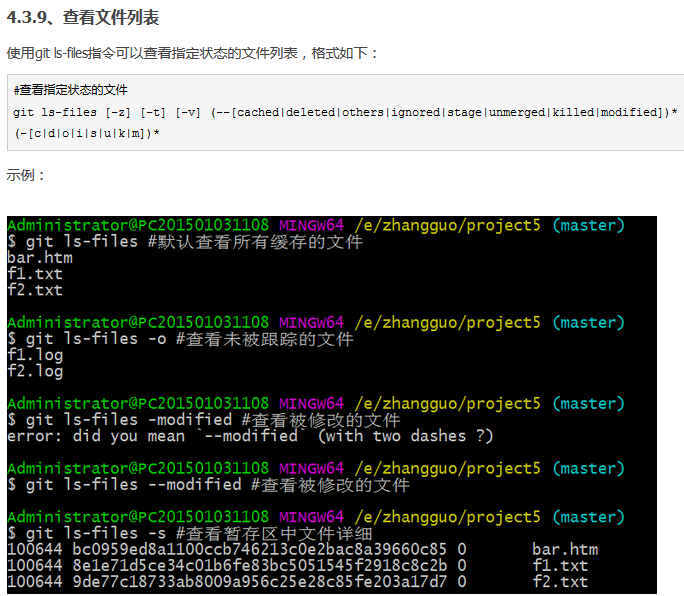
所以这个命令，慎用！

**6. git checkout -p <branch>**

这个命令可以用来打补丁。这个命令主要用来比较两个分支间的差异内容，并提供交互式的界面来选择进一步的操作。这个命令不仅可以比较两个分支间的差异，还可以比较单个文件的差异哦！



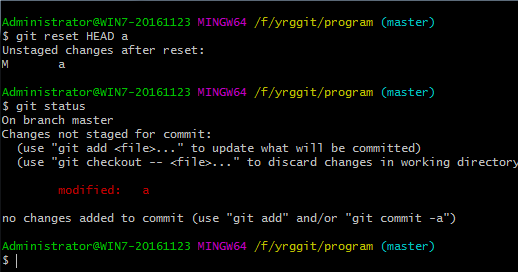




撤销提交

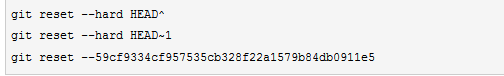
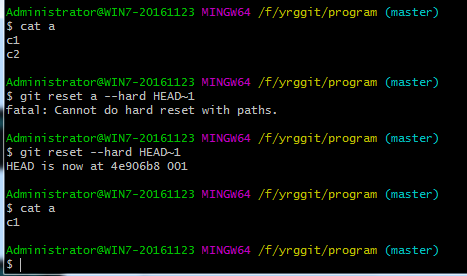
**撤销暂存区更新（文件必须已经commit然后又修改执行add）：**

git reset HEAD <file>



**撤销本地仓库更新**

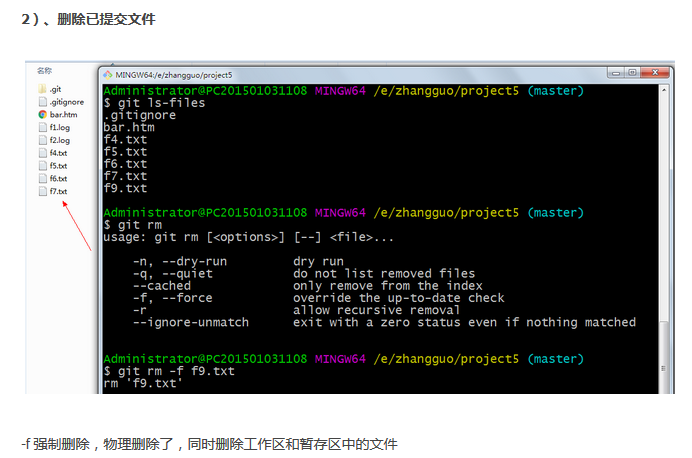
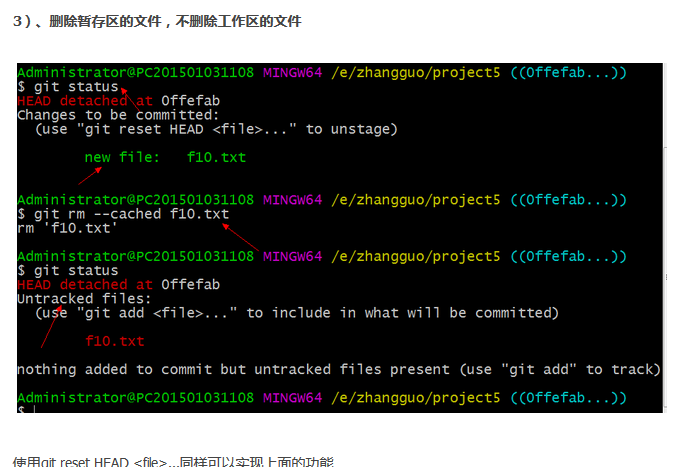
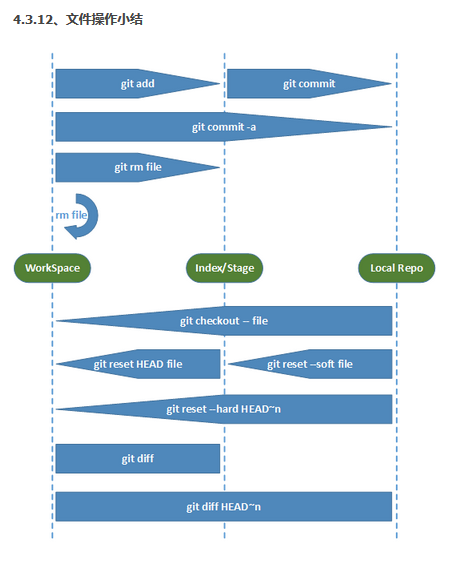
**使用HEAD指针**和**使用commit id**

在Git中，有一个HEAD指针指向当前分支中最新的提交。当前版本，我们使用"HEAD^"，那么再钱一个版本可以使用"HEAD^^"，如果想回退到更早的提交，可以使用"HEAD~n"。（也就是，HEAD^=HEAD~1，HEAD^^=HEAD~2） 

删除文件

删除未跟踪的文件 rm + 文件名

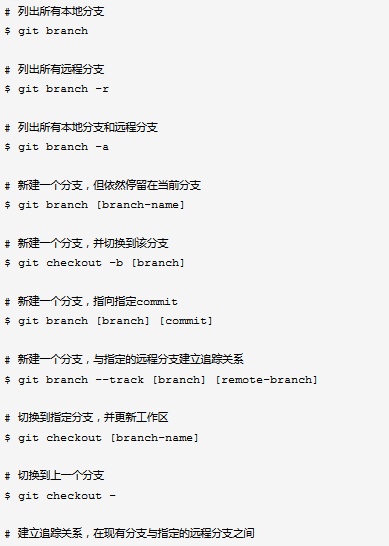
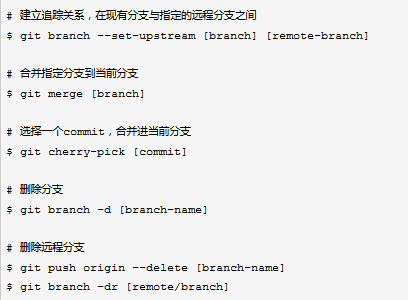
删除已提交文件 git rm +文件名（同时删除仓库 暂存区 及本地文件）

GIT分支：分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作

截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支

查看本地分支：git branch

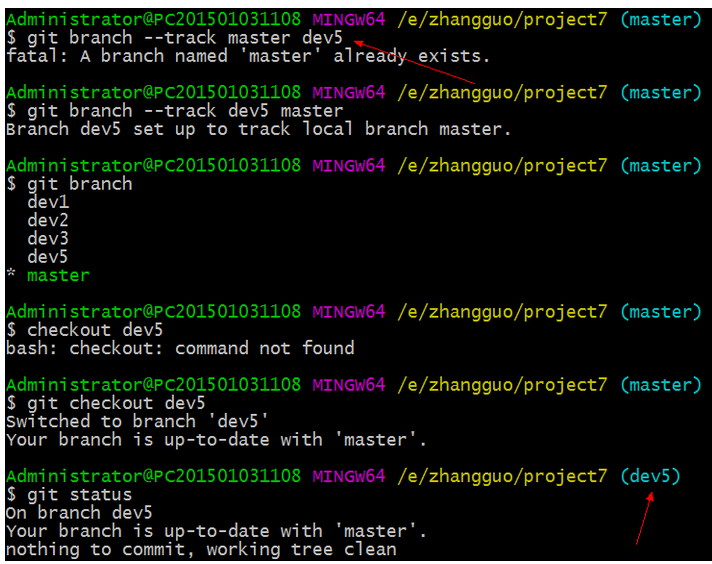
查看远程分支本地使用：git branch -r

新建分支：git branch [branch-name]

新建一个分子并且换到该分支：git checkout -b [branch]



新建一个分支，与指定的远程分支建立追踪关系使用命令：$ git branch --track [branch] [remote-branch]



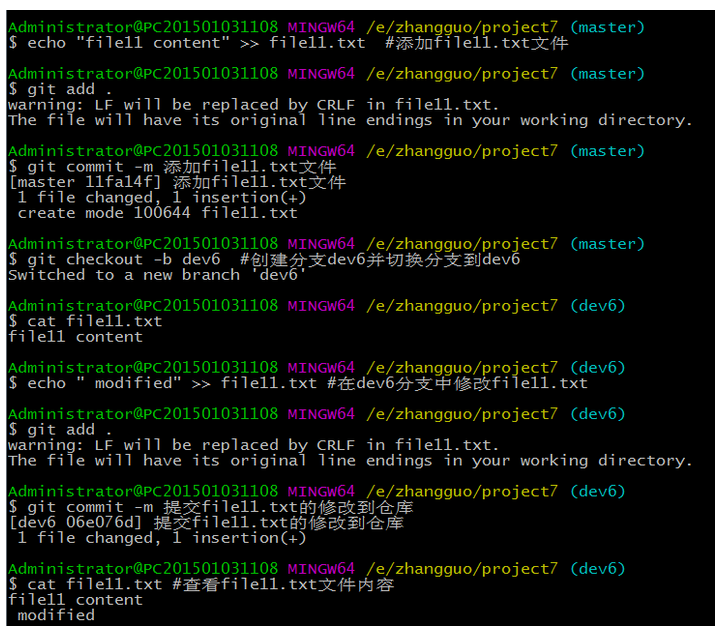
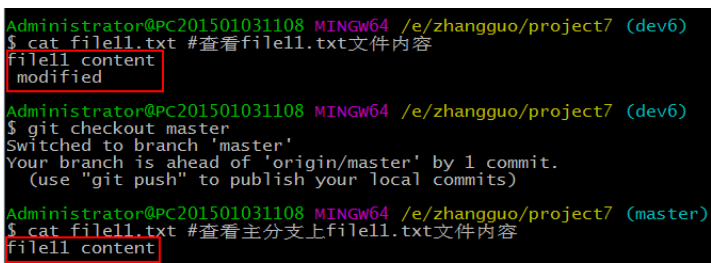
切换分支：git checkout [branch-name]

切换上一个分支：git branch –

合并分支

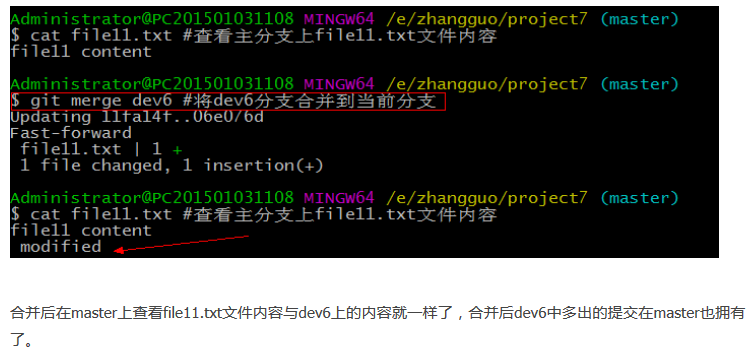
合并当前分支到主分支（合并分支就是对分支的指针操作）

$ git merge [branch]

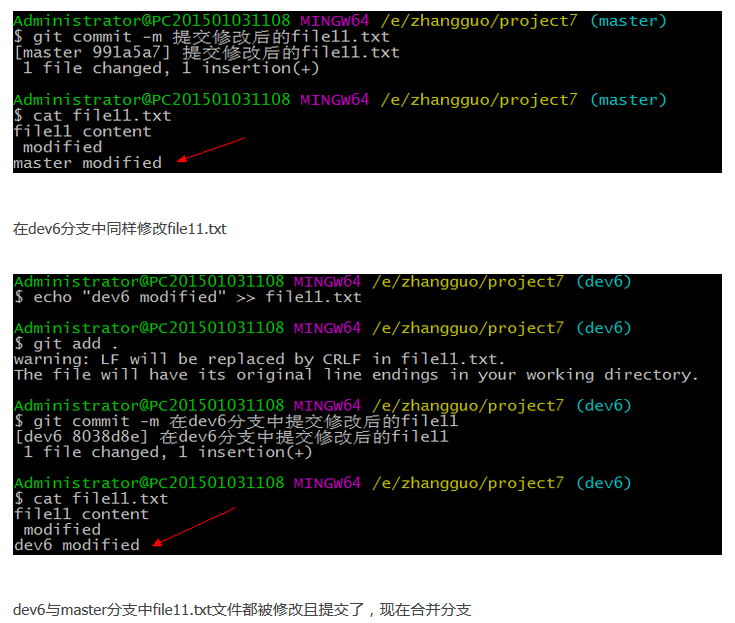
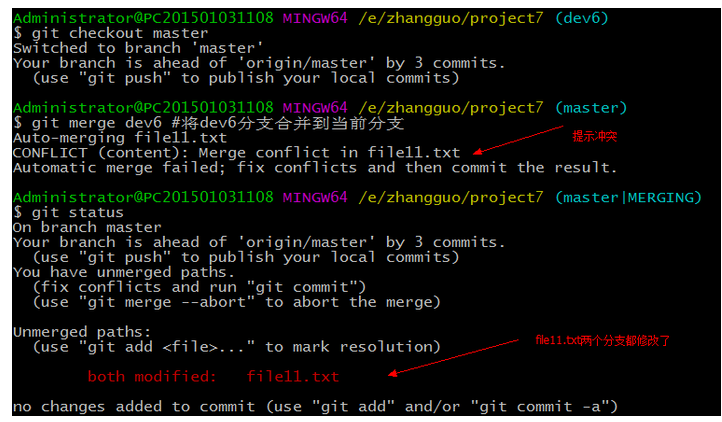
这里的file11.txt主分支与dev6的内容现在是不同的，因为在dev6中已被修改过

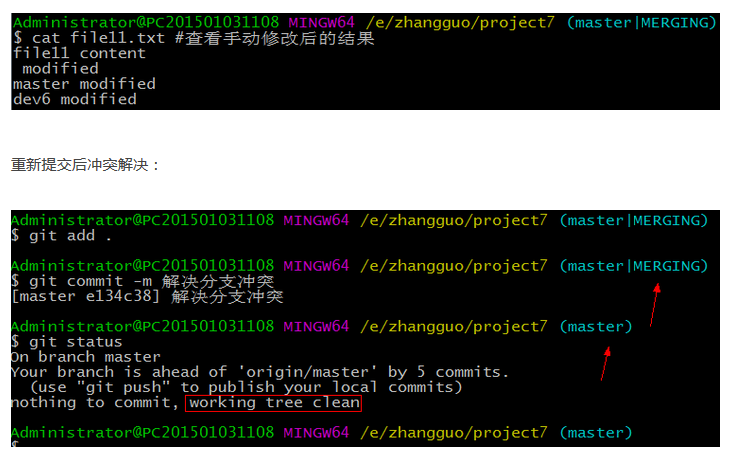
现在我们将dev6合并到主分支中去



冲突解决

如果同一个文件在合并分支时都被修改了则会引起冲突



手动解决完冲突后就可以把此文件添 加到索引(index)中去，用git commit命令来提交，就像平时修改了一个文件 一样。

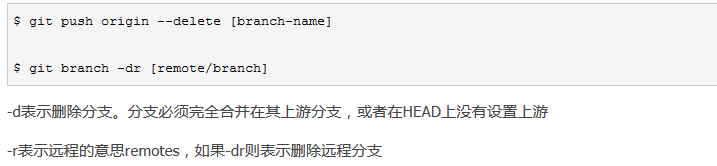
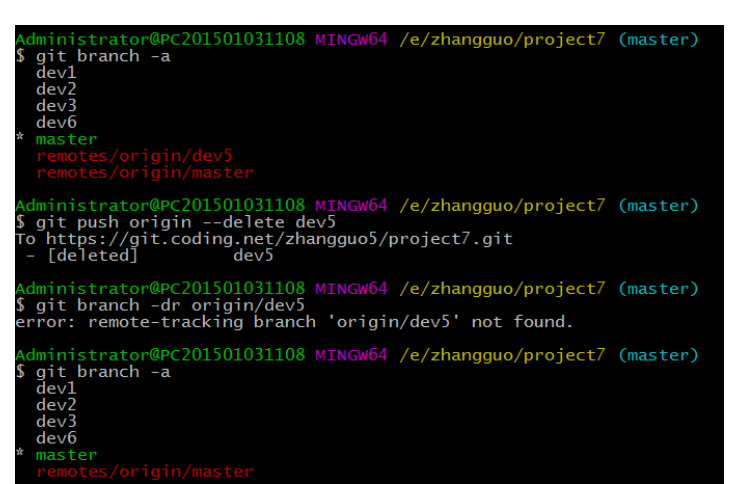
用git log --graph命令可以看到分支合并图



master主分支应该非常稳定，用来发布新版本，一般情况下不允许在上面工作，工作一般情况下在新建的dev分支上工作，工作完后，比如上要发布，或者说dev分支代码稳定后可以合并到主分支master上来

删除分支

删除本地分支可以使用命令：$ git branch -d [branch-name]，-D（大写）强制删除

删除远程分支可以使用如下指令： 

Git 远程仓库(Github)

添加新的远程仓库: shortname缩略名以便引用

git remote add [shortname] [url]

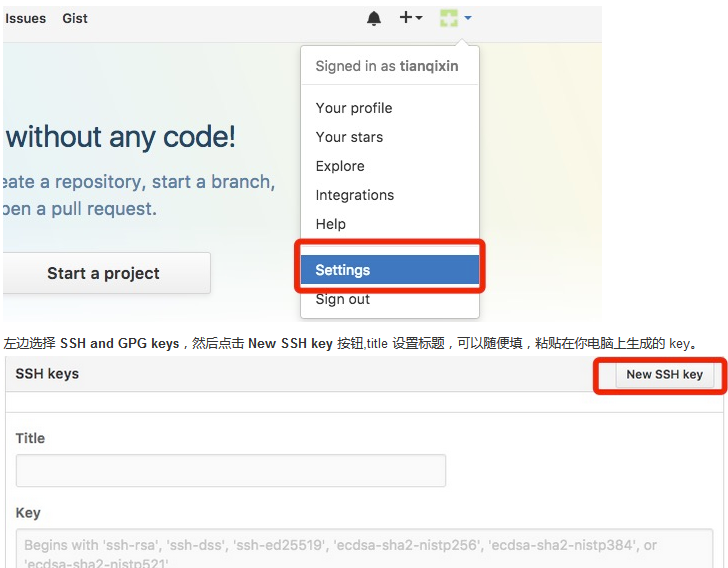
如果本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以我们需要配置验证信息

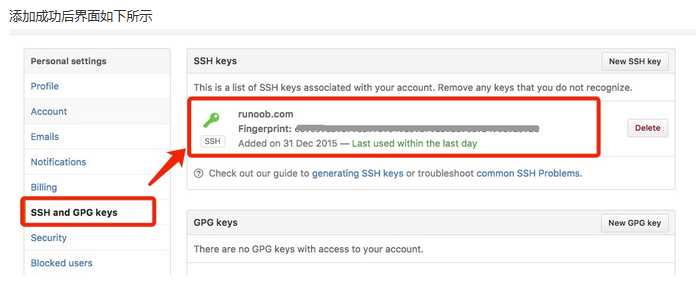
本地生成ssh密匙

$ ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)

**your\_email@youremail.com** 改为你在 github 上注册的邮箱，之后会要求确认路径和输入密码，我们这使用默认的一路回车就行，成功的话会在~/下生成.ssh文件夹，进去，打开 id\_rsa.pub，复制里面的 key

回到 github 上，进入 Account => Settings（账户配置）。





命令行验证添加成功



查看当前的远程仓库（执行时加上 -v 参数，你还可以看到每个别名的实际链接地址）

$ git remote

origin

$ git remote -v

origin    git@github.com:tianqixin/runoob-git-test.git (fetch)

origin    git@github.com:tianqixin/runoob-git-test.git (push)

提取远程仓库(该命令执行完后需要执行git merge 远程分支到你所在的分支)

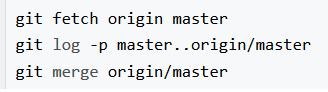
从远程仓库下载新分支与数据：git fetch

从远端仓库提取数据并尝试合并到当前分支:git merge

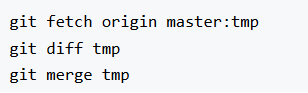
该命令就是在执行 git fetch 之后紧接着执行 git merge 远程分支到你所在的任意分支。

假设你配置好了一个远程仓库，并且你想要提取更新的数据，你可以首先执行 **git fetch [alias]** 告诉 Git 去获取它有你没有的数据，然后你可以执行 **git merge [alias]/[branch]** 以将服务器上的任何更新（假设有人这时候推送到服务器了）合并到你的当前分支。

git fetch：相当于是从远程获取最新版本到本地，不会自动merge



以上命令的含义：   
首先从远程的origin的master主分支下载最新的版本到origin/master分支上   
然后比较本地的master分支和origin/master分支的差别   
最后进行合并

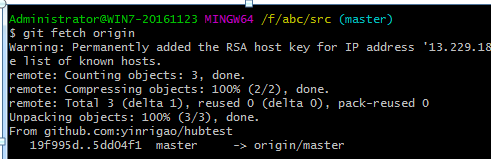


从远程获取最新的版本到本地的test分支上   
之后再进行比较合并

git pull：相当于是从远程获取最新版本并merge到本地

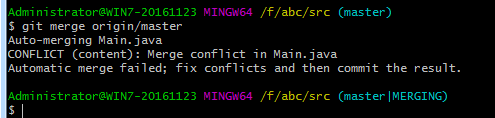
git pull origin master

远程仓库修改文件，本地更新



以上信息"mmmmmmmmmmm -> origin/master" 说明 master 分支已被更新，我们可以使用以下命令将更新同步到本地

$ git merge origin/master



推送你的新分支与数据到某个远端仓库命令: git push [alias] [branch]

以上命令将你的 [branch] 分支推送成为 [alias] 远程仓库上的 [branch] 分支



删除远程仓库

git remote rm [别名]

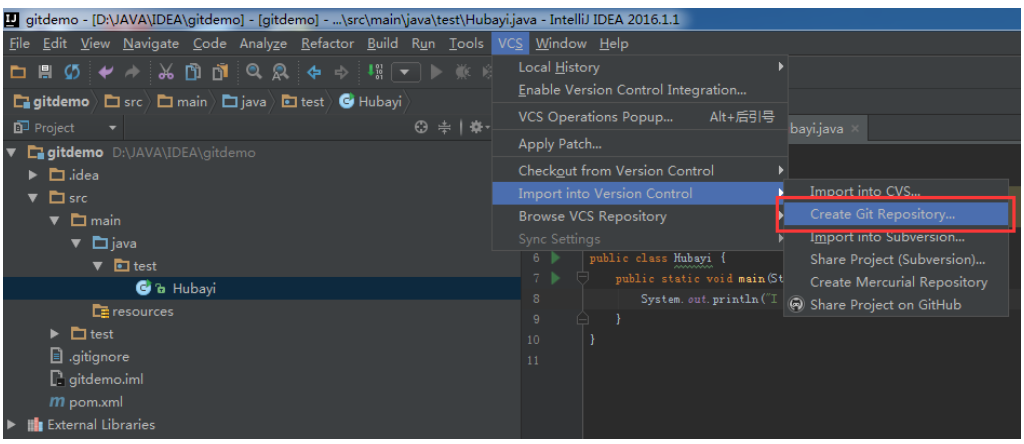
Git远程仓库：<https://github.com/yinrigao/yrgideagit.git>

https：//github.com

账号：yinrigao/yinrigao1022

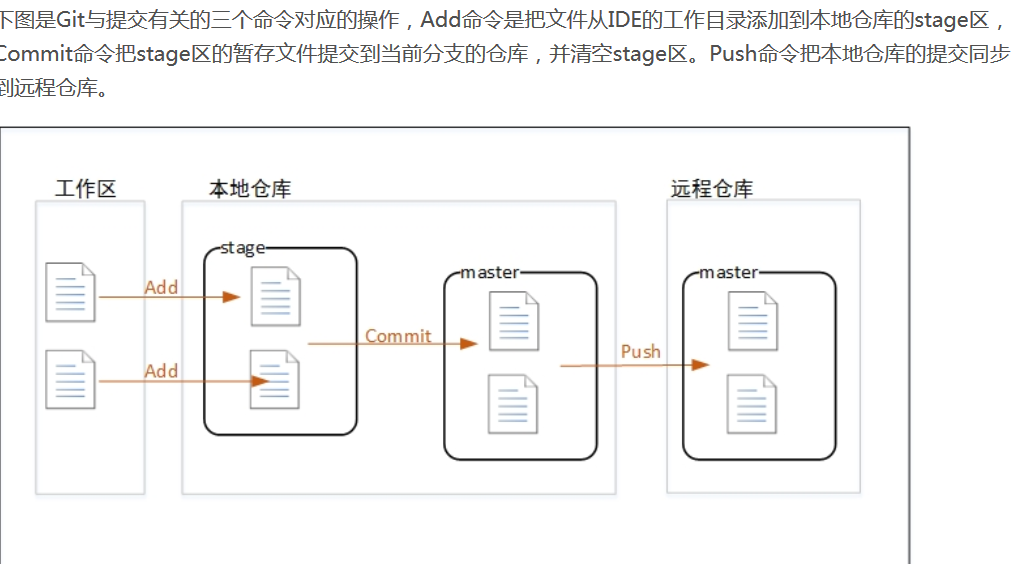
创建项目并提交到远程git仓库

创建好项目，选择VCS - > Import into Version Control -> Create Git Repository

****

接下来指定本地仓库的位置，按个人习惯指定即可，例如这里选择了项目源代码同目录

点击OK后创建完成本地仓库，注意，这里仅仅是本地的。下面把项目源码添加到本地仓库（**以上所有步骤相当在指定目录于创建了一个本地仓库=git init命令**）



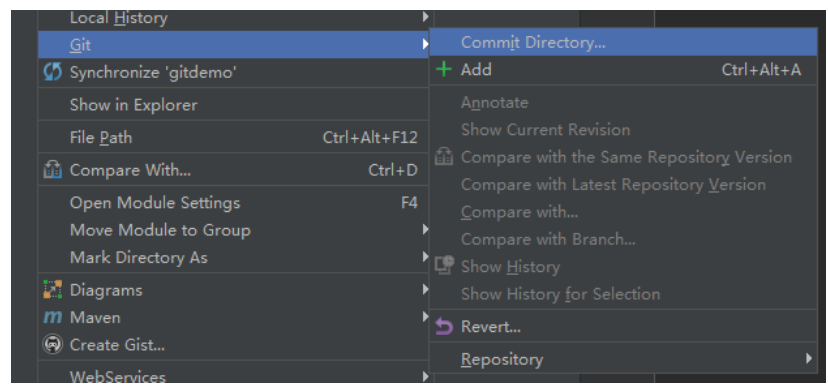
新建项目的提交

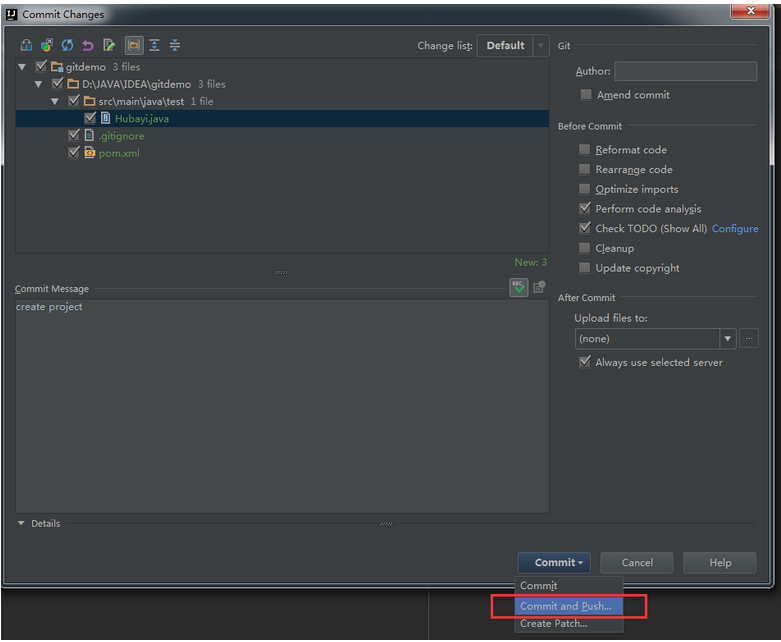
右击项目按如下操作执行

步骤一添加文件到暂存区



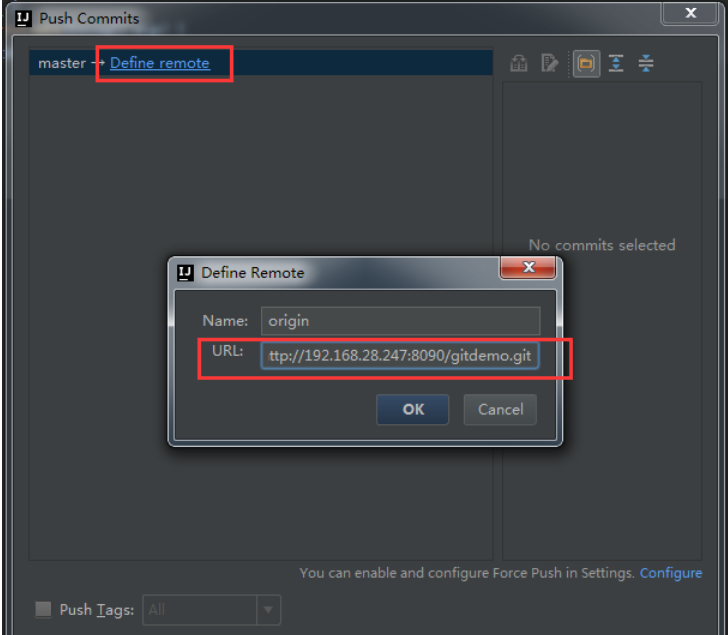
步骤二添加到本地仓库和远程仓库



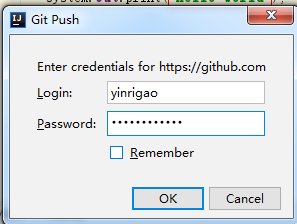


注意途中红色的框：表示将要执行的步骤为同时添加代码到本地和远程仓库并删除暂存区的代码

如果是第一次提交： Push前需要指定远程仓库的地址。如下图，点击Define remote后，在弹出的窗口中输入远程仓库地址

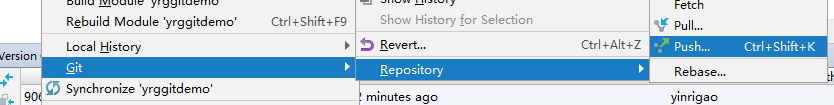


输入用户名密码完成提交



提交本地仓库到远程仓库

项目右击按下图操作



如果是第一次提交，会产生如下报错

Push to origin/master was rejected



原因GIT远程仓库的文件本地仓库没有需要先关联到远程仓库

在本地仓库命令行执行如下代码

git remote add origin + 远程GIT的url

git push -u origin master

GIT远程仓库的地址的操作

git remote：查看

git remote rm origin：删除

git remote add origin yourURL：关联

本地仓库提交到github远程仓库有两种方式

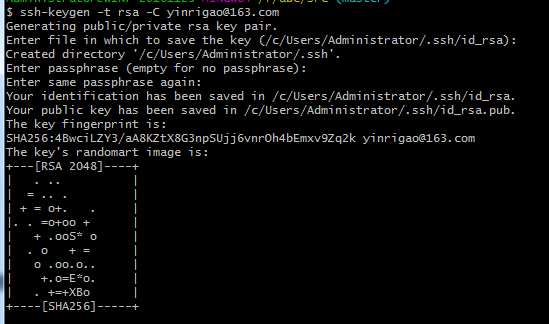
一种是通过ssh（每次提交的时候不需要输入用户名密码），另一种是https

ssh方法

a．在本地仓库打开bash生成ssh密匙

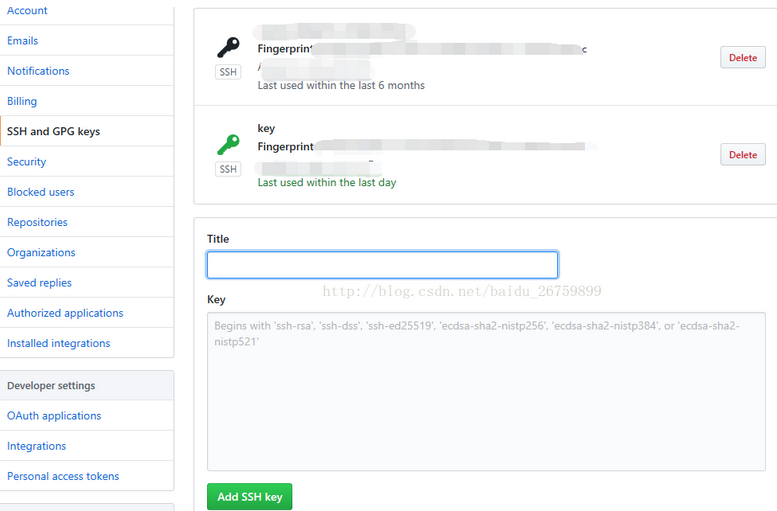
ssh-keygen -t rsa -C "your\_email@youremail.com"

之后会让你确认一下秘钥的路径和密码，**直接三次回车就好**。然后在保存秘钥的路径下找到生成的.ssh文件夹，进去，打开id\_rsa.pub，复制里面的key

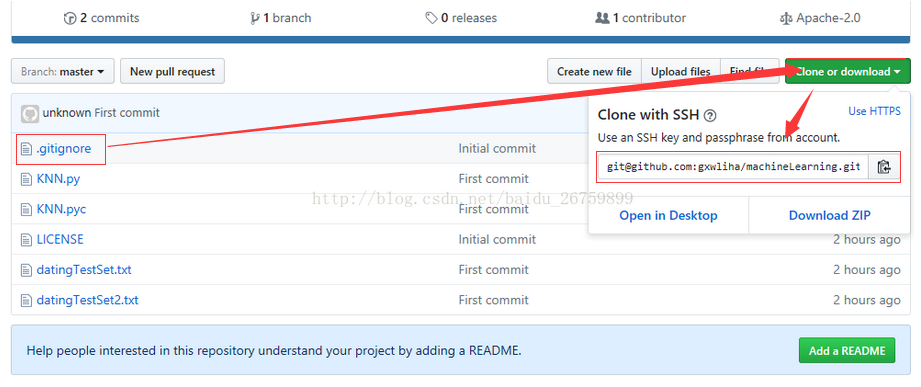


b.在github上配置ssh

回到github上，进入 Account Settings（账户配置），左边选择SSH Keys，Add SSH Key,title随便填，粘贴在你电脑上生成的key

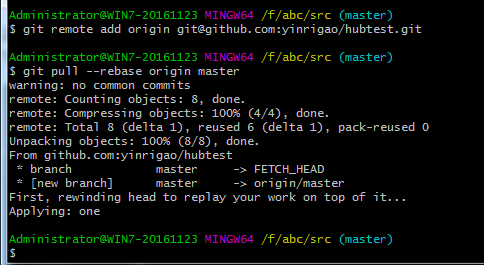


c．关联本地仓库（注意使用sshURL），并同步远程仓库文件

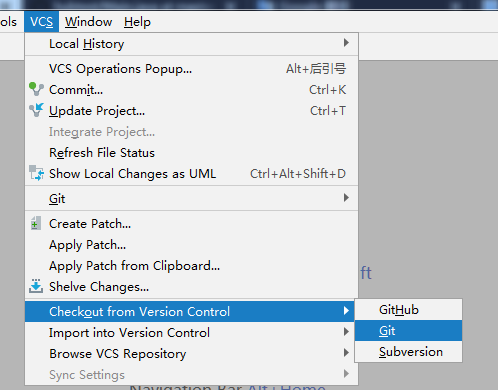


git remote add origin [git@github.com:yourName/yourRepo.git](mailto:git@github.com:yourName/yourRepo.git)

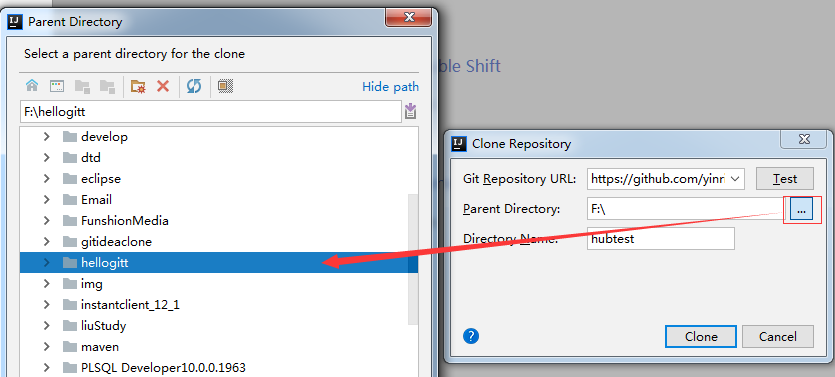
git pull --rebase origin master（同步远程仓库文件ssh）

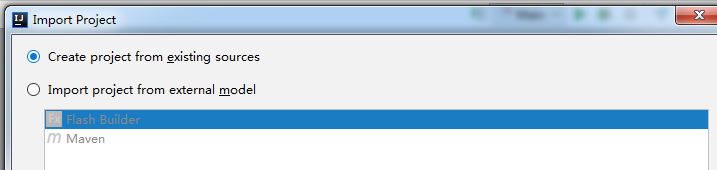


idea从远程仓库检出项目



选择项目检出地址





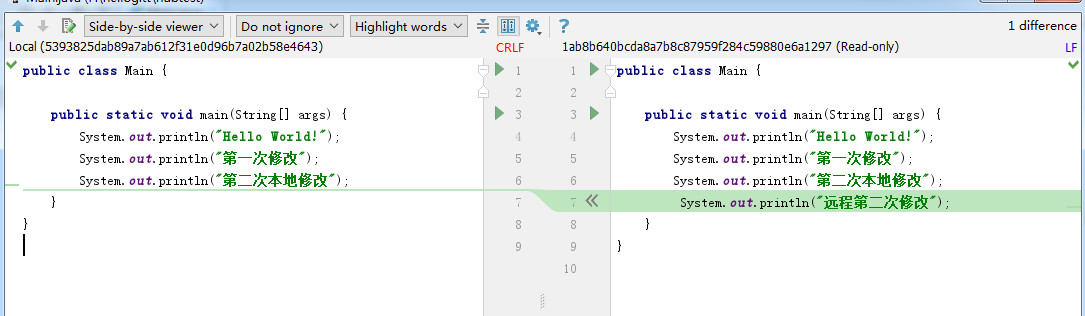
下一步。。。结束

本地获取当前项目其他人员已经同步到远程仓库提交

获取更新有两个命令：Fetch和Pull，Fetch是从远程仓库下载文件到本地的origin/master，然后可以手动对比修改决定是否合并到本地的master库。Push则是直接下载并合并。如果各成员在工作中都执行修改前先更新的规范，则可以直接使用Pull方式以简化操作

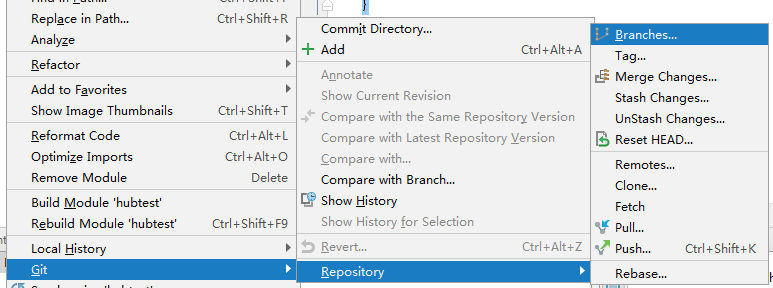
使用Fetch后本地对比操作

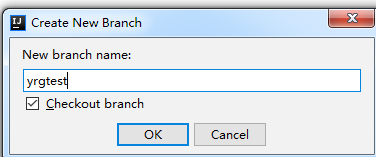
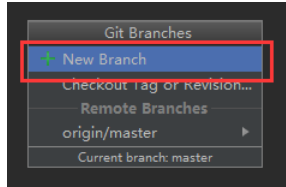




如果确定更行然执行Pull

本地创建一个新分支执行新功能开发，开发完成后合并到远程分支

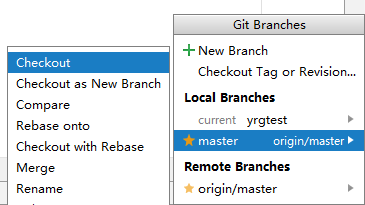




Idea右下角：自动切换到新建的分支

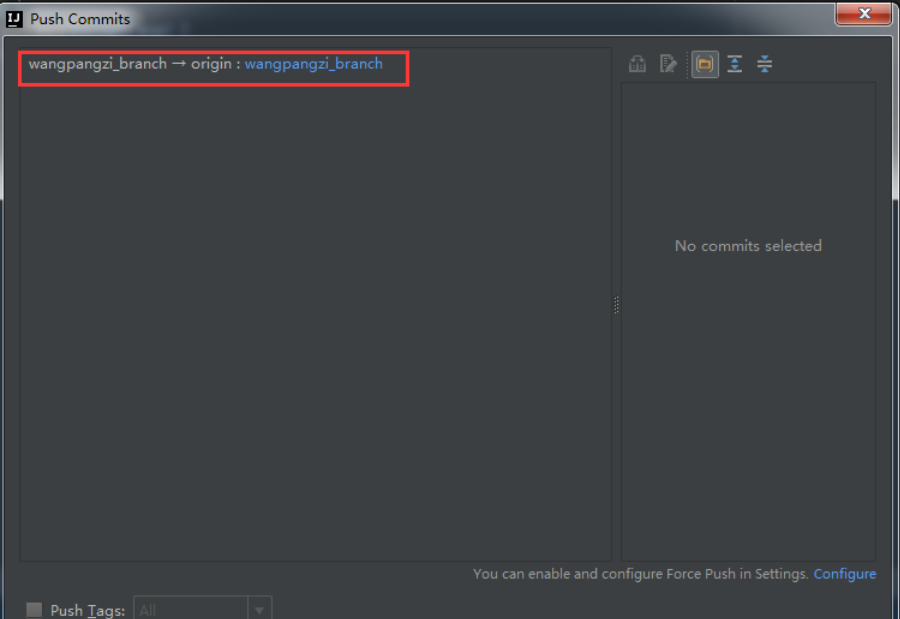
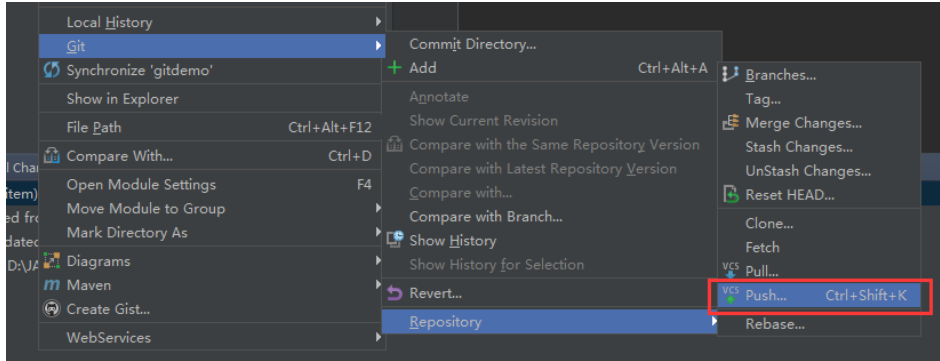


点击后弹出一个小窗口，在Local Branches中有其他可用的本地分支选项，点击后选择Checkout即可切换当前工作的分支

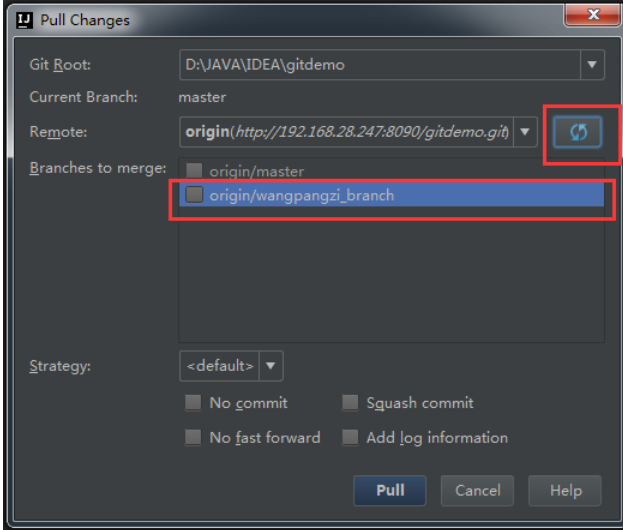


提交本地分支到远程

切换到新建分支使用push



小组其他成员获取提交到远程仓库的分支

使用Pull功能打开更新窗口，点击Remote栏后面的刷新按钮，会在Branches to merge栏中刷新出新的分支。这里并不想做合并，所以不要选中任何分支，直接点击Pull按钮完成操作。

更新后，再点击右下角，可以看到在Remote Branches区已经有了新的分支，点击后在弹出的子菜单中选择Checkout as new local branch，在本地仓库中创建该分支。完成后在Local Branches区也会出现该分支的选项，可以按上面的方法，点击后选择Checkout切换

