# 注意事项

* 多人开发时，应该在commit前，先pull主分支，然后再commit和push

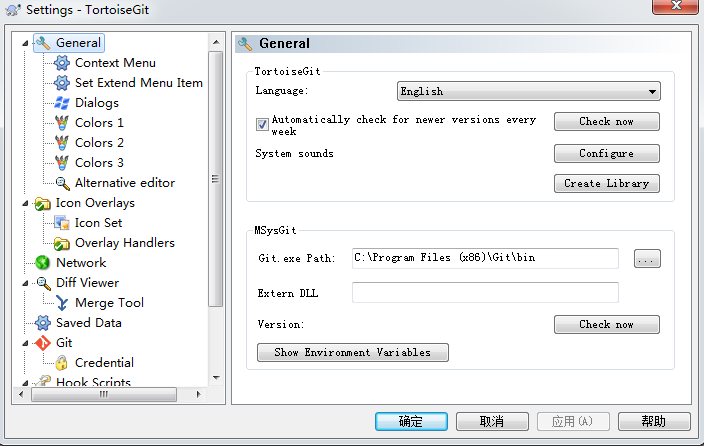
# 公司的gitlab服务器地址

<http://gitlab.alibaba-inc.com/>

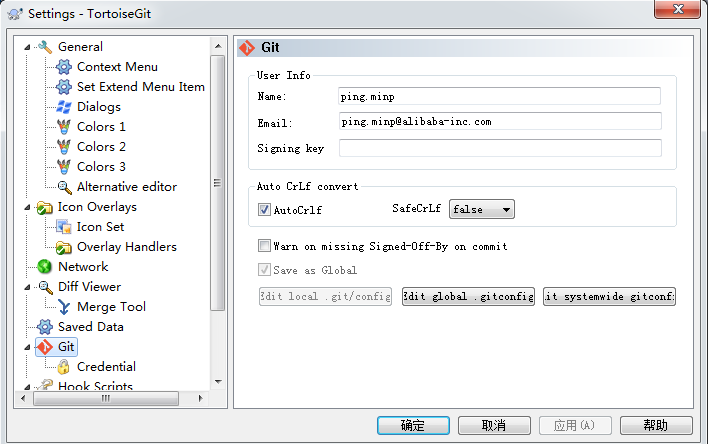
# win7下搭建gitlab客户端

Windows下使用TortoiseGit  
1. 需要先安装msysgit   
2. 安装Tortoisegit

3. 运行Tortoisegit setting 中关联msysgit。



设置邮箱和用户名



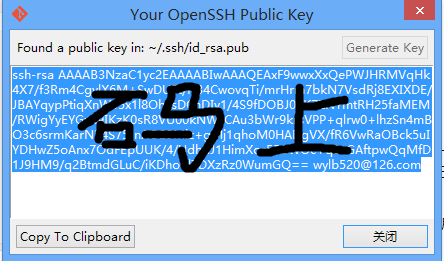
4、登录gitlab服务器（如http://gitlab.alibaba-inc.com/，并登陆我的账号），把鼠标移到页面右上角的”My profile“，该栏是用户的配置信息，包括账户信息、SSH Keys以及代码风格等  
5、首先需要把自己本机上的SSH Key复制到GitLab，是为了今后GitLab服务器能够无密码访问本机，生成ssh key的步骤是

a首先打开之前安装的”Git Bash“（Git安装后的一部分）打开msysGit->msys.bat命令行工具，输入”ssh-keygen -t rsa –C “邮箱地址”“，把里面的改成自己的注册邮箱地址（如ssh-keygen -t rsa –C “**yuanchao.yyc@alibaba-inc.com**”），然后一直敲回车就行，下图是我电脑生成SSH Key的情况：

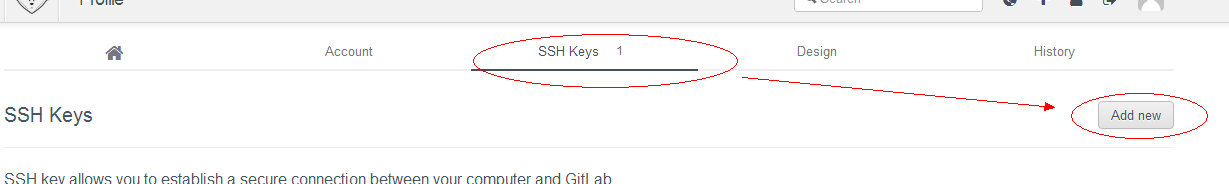


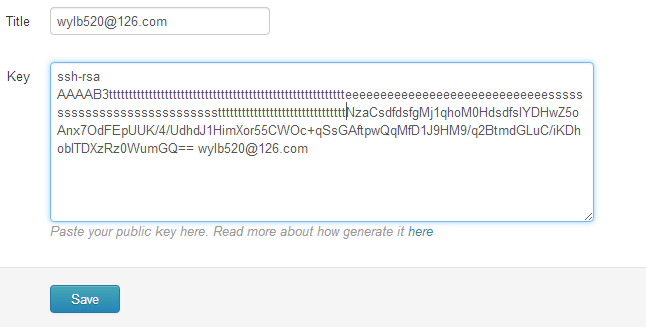
b 生成SSH Key之后，打开“Git GUI”（Git安装后的另一部分），点击菜单的help－ssh，然后copy当前的“key”

（或者也可以直接复制自己电脑下的C:\Users\<用户名>\.ssh文件夹下id\_rsa.pub文件中的内容）



c 然后回到之前的My profile，先点击页面“SSH Keys”，再点击”Add new“，出现的页面中”Title“为SSH key的名字，可不用理会，将复制到的ssh key贴到“Key”的文本框中，会自动生成key的名字，点击保存。





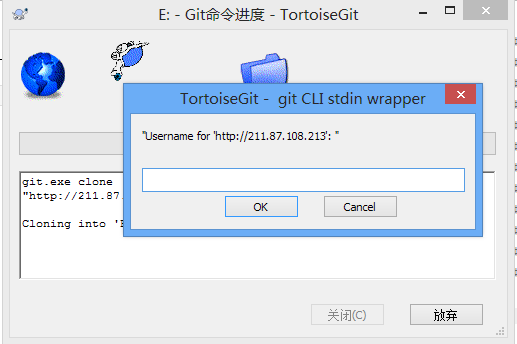
6、克隆版本库，这个很实用，当你想要跟踪大牛开发的最新动态，或者是参与到项目的开发中时，第一步就是要克隆版本库。

在任意位置右键，选择“在这里创建版本库”（我是汉化的版本，菜单不一致的请自行翻译），设置远程地址(注意下图远程地址有误，掉了.git)，和本地存储地址：

（注：我先使用的是公司项目的ssh地址，结果在克隆时一直喊我输入密码。然后我改为使用http地址，就能克隆成功了！）

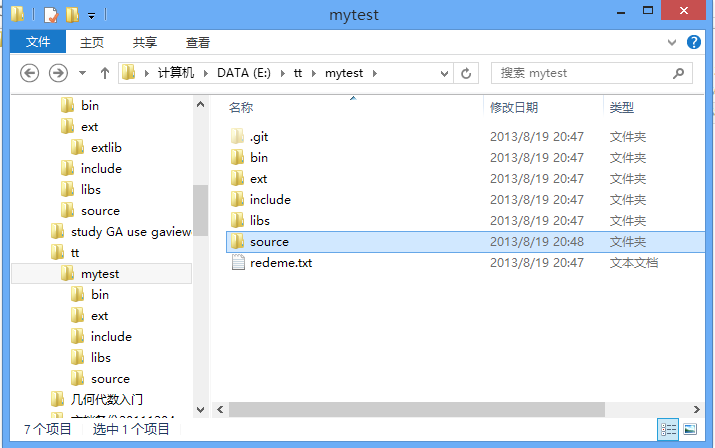


输入帐号密码开始克隆



克隆完成之后，这样我们已经得到一个受版本控制的整个项目文件了。





7、修改文件后，先commit，然后再push到服务器

每次修改代码时先执行git pull更新本地代码到最新版本。

# 使用ssh

## 配置ssh

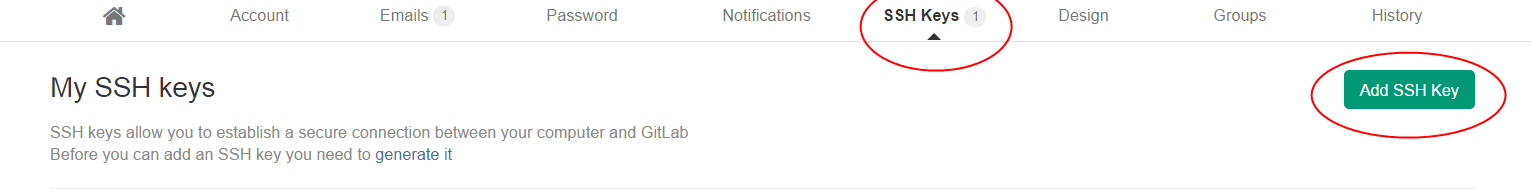
1、使用git bash打开命令框

2、输入：ssh-keygen -t rsa -C <email地址>

Email地址如：**yuanchao.yyc@alibaba-inc.com**

3、然后三次回车，私有密钥保存在id\_rsa中，公有密钥保存在id\_rsa.pub中

4、将id\_rsa.pub的所有内容复制到gitlab中（title随便输入）：



现在就配置好了，每次使用ssh连接gitlab时，服务器会自动匹配私有和公钥，不用自己输入密码了！

## 使用ssh clone git项目

1、可直接在git bash中使用git clone命令

如：git clone ssh://git@gitlab.alibaba-inc.com:dingding/web.git

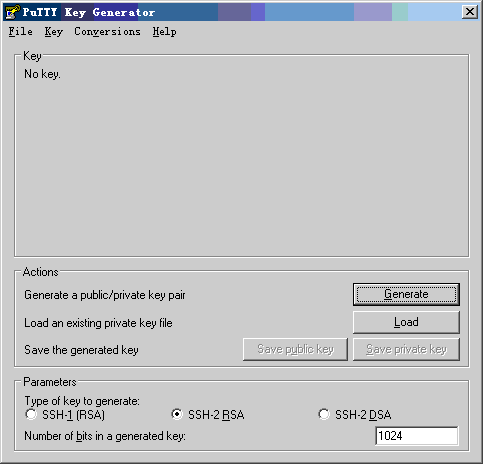
如果不行，就试下

git clone ssh://git@gitlab.alibaba-inc.com/dingding/web.git

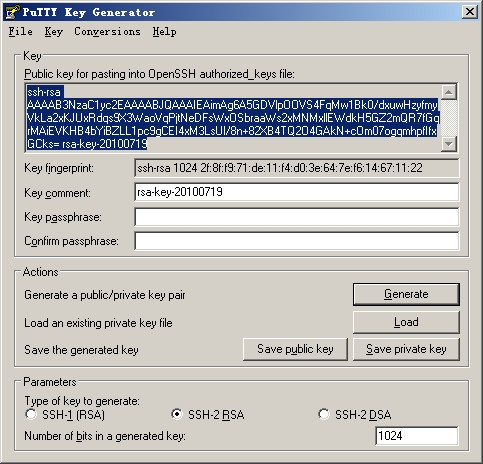
2、或者使用tortoisegit，但是需要配置（不能直接clone，直接clone使用的是http形式）

[TortoiseGit](http://code.google.com/p/tortoisegit/) 使用扩展名为ppk的密钥，而不是ssh-keygen生成的rsa密钥。也就是说使用ssh-keygen -C "username@email.com" -t rsa产生的密钥在TortoiseGit中不能用。而基于[github](https://github.com/)的开发必须要用到rsa密钥，因此需要用到TortoiseGit的putty key generator工具来生成既适用于github的rsa密钥也适用于TortoiseGit的ppk密钥，配置步骤如下：

1）运行开始菜单中的TortoiseGit puttygen程序，如下图示



2）点击“Generate”按钮，鼠标在上图的空白地方来回移动直到进度条完毕，就会自动生一个随机的key，如下图示

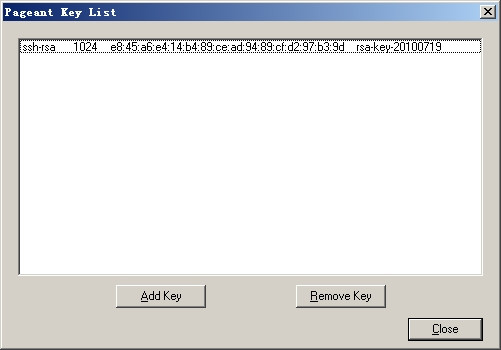


 如有需要，可以为密钥设置对应的访问密码，就是修改上图中“Key passphrase”和“Confirm passphrase”的值。

3）将上图中多行文本框的内容全选、复制，并粘贴到github账户的 SSH public key中，这就是适用于github的公钥。

4）点击上图中的“Save private key”按钮,将生成的key保存为适用于TortoiseGit的私钥（扩展名为.ppk）。

5）运行TortoiseGit开始菜单中的Pageant程序，程序启动后将自动停靠在任务栏中，图标显示为TortoiseGit密钥的配置 - rongjih - dragon的博客，双击该图标，弹出key管理列表，如下图示



6）点击上图中的“Add Key”按钮，将第4步保存的ppk私钥添加进来，关闭对话框即可

7）经上述配置后，你就可以使用TortoiseGit进行push、pull操作了。

# http clone

## 一步设置让TortoiseGit+msysgit记住帐号密码

使用github的公共仓库，但每次提交都要求填写帐号密码，有点麻烦，下面两种方法可以让git“记住密码”

==================================================================

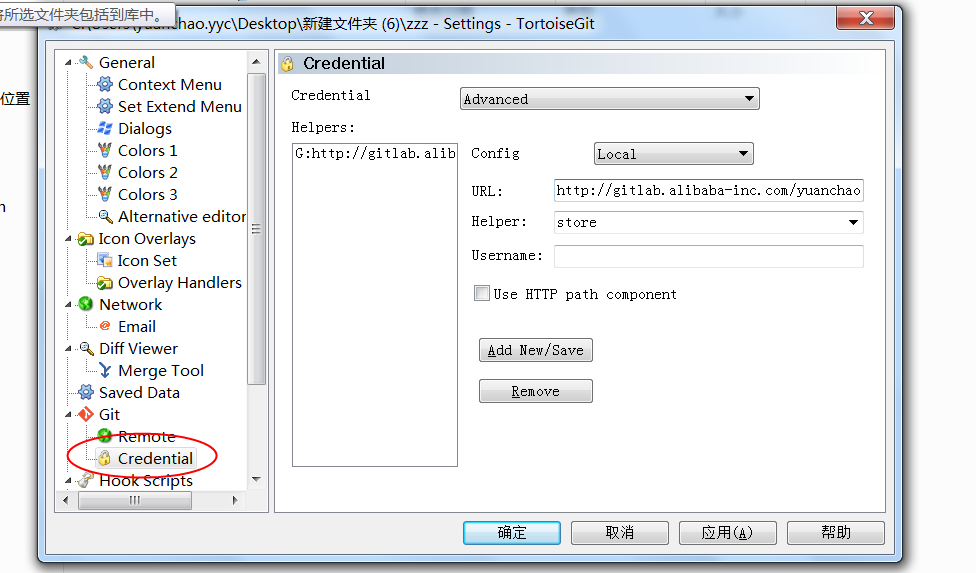
1. 在"C:\Documents and Settings\Administrator\.gitconfig" 文件 或 "项目/.git/config"文件 里增加两行：

**[plain]** [view plaincopy](http://blog.csdn.net/xiangzi1113/article/details/9213667)

1. [credential]
2. helper = store

设置完毕！

1. 或者设置TortoiseGit->settings->Git->Credential，设置url和helper即可：



下次git操作提示输入帐号密码后就会自动往了，帐号密码信息会保存在 C:\Documents and Settings\Administrator\.git-credentials里，如下效果：

**[plain]** [view plaincopy](http://blog.csdn.net/xiangzi1113/article/details/9213667)

1. https://username:987654321@github.com
2. https://myusername:123456789@other.com

安全性麻，自己看着办。

# 操作

## 撤消操作

如果你现在的工作目录(work tree)里搞的一团乱麻, 但是你现在还没有把它们提交; 你可以通过下面的命令, 让工作目录回到上次提交时的状态(last committed state):

$ git reset --hard HEAD

这条件命令会把你工作目录中所有未提交的内容清空(当然这不包括未置于版控制下的文件 untracked files). 从另一种角度来说, 这会让"git diff" 和"git diff --cached"命令的显示法都变为空.

如果你只是要恢复一个文件,如"hello.rb", 你就要使用 [git checkout](http://www.kernel.org/pub/software/scm/git/docs/git-checkout.html)

$ git checkout -- hello.rb

这条命令把hello.rb从HEAD中签出并且把它恢复成未修改时的样子.

译者:上面二行和原文有出入，经验证是原文有误,所以我据正确的重写了.

### 修复已提交文件中的错误

如果你已经做了一个提交(commit),但是你马上后悔了, 这里有两种截然不同的方法去处理这个问题:

1. 创建一个新的提交(commit), 在新的提交里撤消老的提交所作的修改. 这种作法在你已经把代码发布的情况下十分正确.

2　你也可以去修改你的老提交(old commit). 但是如果你已经把代码发布了,那么千万别这么做; git不会处理项目的历史会改变的情况,如果一个分支的历史被改变了那以后就不能正常的合并.

#### 创建新提交来修复错误

创建一个新的，撤消(revert)了前期修改的提交(commit)是很容易的; 只要把出错的提交(commit)的名字(reference)做为参数传给命令: [git revert](http://www.kernel.org/pub/software/scm/git/docs/git-revert.html)就可以了; 下面这条命令就演示了如何撤消最近的一个提交:

$ git revert HEAD

这样就创建了一个撤消了上次提交(HEAD)的新提交, 你就有机会来修改新提交(new commit)里的提交注释信息.

你也可撤消更早期的修改, 下面这条命令就是撤消“上上次”(next-to-last)的提交:

$ git revert HEAD^

在这种情况下,git尝试去撤消老的提交,然后留下完整的老提交前的版本.　如果你最近的修改和要撤消的修改有重叠(overlap),那么就会被要求手工解决冲突(conflicts),　就像解决合并(merge)时出现的冲突一样.

译者注: git revert 其实不会直接创建一个提交(commit), 把撤消后的文件内容放到索引(index)里,你需要再执行git commit命令，它们才会成为真正的提交(commit).

#### 修改提交来修复错误

如果你刚刚做了某个提交(commit), 但是你又想马上修改这个提交; [git commit](http://www.kernel.org/pub/software/scm/git/docs/git-commit.html) 现在支持一个叫**--amend**的参数，它能让你修改刚才的这个提交(HEAD commit). 这项机制能让你在代码发布前,添加一些新的文件或是修改你的提交注释(commit message).

如果你在老提交(older commit)里发现一个错误, 但是现在还没有发布到代码服务器上. 你可以使用 [git rebase](http://www.kernel.org/pub/software/scm/git/docs/git-rebase.html)命令的交互模式, "git rebase -i"会提示你在编辑中做相关的修改. 这样其实就是让你在rebase的过程来修改提交.

## 回滚

git代码库回滚: 指的是将代码库某分支退回到以前的某个commit id

【本地代码库回滚】：

git reset --hard commit-id :回滚到commit-id，讲commit-id之后提交的commit都去除

git reset --hard HEAD~3：将最近3次的提交回滚

git reset --hard：将最近一次提交回滚

【远程代码库回滚】：

这个是重点要说的内容，过程比本地回滚要复杂

应用场景：自动部署系统发布后发现问题，需要回滚到某一个commit，再重新发布

原理：先将本地分支退回到某个commit，删除远程分支，再重新push本地分支

操作步骤：

1、git checkout the\_branch

2、git pull

3、git branch the\_branch\_backup //备份一下这个分支当前的情况

4、git reset --hard the\_commit\_id //把the\_branch本地回滚到the\_commit\_id

5、git push origin :the\_branch //删除远程 the\_branch

6、git push origin the\_branch //用回滚后的本地分支重新建立远程分支

7、git push origin :the\_branch\_backup //如果前面都成功了，删除这个备份分支,(远程无此分支，只需删除本地分支)

如果使用了gerrit做远程代码中心库和code review平台，需要确保操作git的用户具备分支的push权限，并且选择了 Force Push选项（在push权限设置里有这个选项）

另外，gerrit中心库是个bare库，将HEAD默认指向了master，因此master分支是不能进行删除操作的，最好不要选择删除master分支的策略，换用其他分支。如果一定要这样做，可以考虑到gerrit服务器上修改HEAD指针。。。不建议这样搞

有时候使用Git工作得小心翼翼，特别是涉及到一些高级操作，例如 reset, rebase 和 merge。甚至一些很小的操作，例如删除一个分支，我都担心数据丢失。  
  
不久之前，我在做一些大动作（rebasing）之前，我总是备份整个版本库，以防万一。直到最近我才发现git的历史记录是不可修改的，也就是说你不能更改任何已经发生的事情。你做的任何操作都只是在原来的操作上修改。也就是说，即使你删除了一个分支，修改了一个提交，或者强制重置，你仍然可以回滚这些操作。  
  
让我们来看一些例子：  
  
$ git init  
$ touch foo.txt  
$ git add foo.txt  
$ git commit -m "initial commit"  
  
$ echo 'new data' >> foo.txt  
$ git commit -a -m "more stuff added to foo"  
  
你现在看git的历史记录，你可以看到两次提交：  
$ git log  
\* 98abc5a (HEAD, master) more stuff added to foo  
\* b7057a9 initial commit  
  
现在让我们来重置回第一次提交的状态：  
$ git reset --hard b7057a9  
$ git log  
\* b7057a9 (HEAD, master) initial commit  
  
这看起来我们是丢掉了我们第二次的提交，没有办法找回来了。但是 reflog 就是用来解决这个问题的。简单的说，它会记录所有HEAD的历史，也就是说当你做 reset，checkout等操作的时候，这些操作会被记录在reflog中。  
  
$ git reflog  
b7057a9 HEAD@{0}: reset: moving to b7057a9  
98abc5a HEAD@{1}: commit: more stuff added to foo  
b7057a9 HEAD@{2}: commit (initial): initial commit  
  
所以，我们要找回我们第二commit，只需要做如下操作：  
$ git reset --hard 98abc5a  
  
再来看一下 git 记录：  
$ git log  
\* 98abc5a (HEAD, master) more stuff added to foo  
\* b7057a9 initial commit  
  
所以，如果你因为reset等操作丢失一个提交的时候，你总是可以把它找回来。除非你的操作已经被git当做垃圾处理掉了，一般是30天以后

### 已解决的问题

1、报错：branch is behind remote branch

|  |
| --- |
| You probably did some history rewriting? Your local branch diverged from the one on the server. Run this command to get a better understanding of what happened:  gitk HEAD @{u}  I would strongly recommend you try to understand where this error is coming from. To fix it, simply run:  git push -f  **The -f makes this a “forced push” and *overwrites* the branch on the server. That is very dangerous when you are working in team.** But since you are on your own and sure that your local state is correct this should be fine. You risk loosing commit history if that is not the case. |

## 加入tag

If you want to add tag with Tortoise GIT:

1. *show log*
2. right click on commit and *create tag at this version*
3. push with checked *include tags*

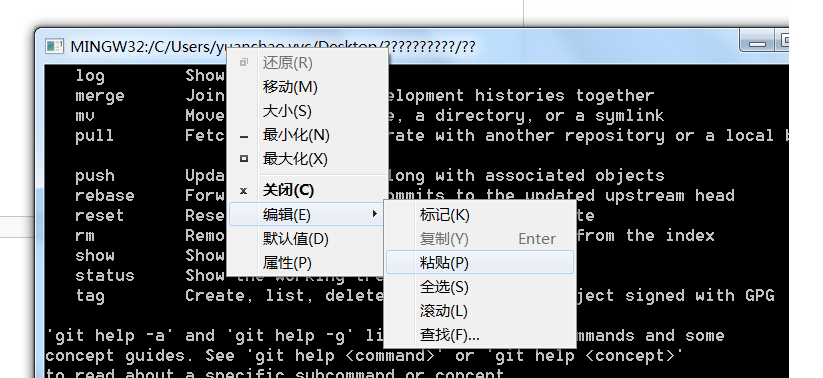
如果已有该tag，则在create tag和push时都要选中force

## 命令行操作

使用git bash命令行工具

### 粘贴与复制

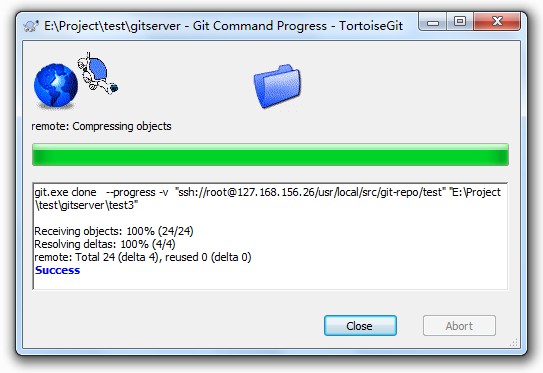
鼠标右击命令行tile：



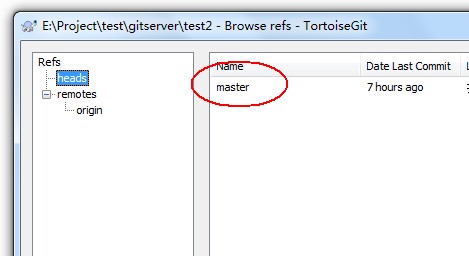
## 分支操作

**版本克隆分支问题**

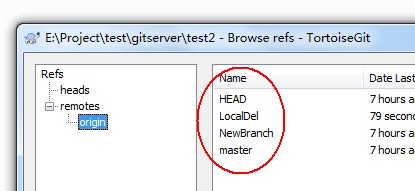
TortoiseGit在克隆分支的时候，默认克隆master分支，克隆后本地工作目录为中心器的Master分支.

[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/1-4)

克隆后本地分支

[[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/2-2)](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/2-2)

中心库分支

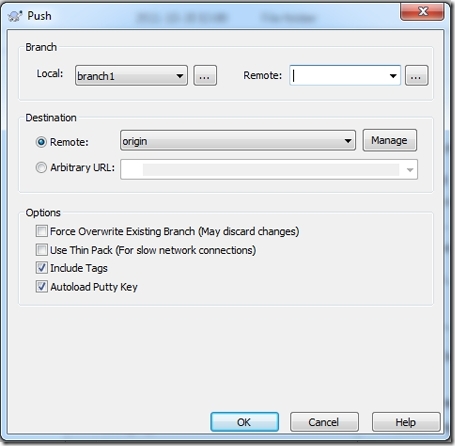
[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/3-2)

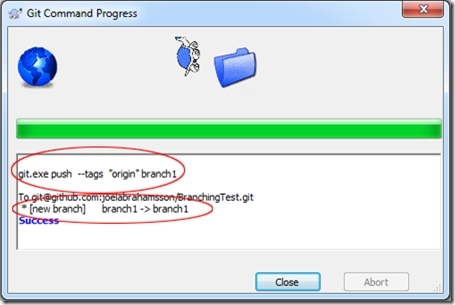
### Push分支到中心服务器（Pushing the local branch to a new remote branch）

Git在本地创建分支后，如果我们需要和团队其他成员做代码交流时就需要将本地分支上传到中心服务器。

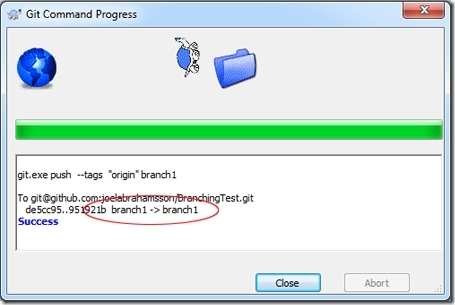
TortoiseGit在将一个新分支上传到中心服务器时，首先在中心服务器创建一个新的分支，然后再将本地分支上面的修改push到远程新创建的分支上面.

在Push新分支到中心服务器上时，通过在本地服务器执行PUSH操作，在PUSH对话框里保持远程分支为空白，则在PUSH的时候远程服务器发现远程没有该分支，此时会自动创建一个和本地分支名称一样的分支，并将本地分支的内容上传到该分支.

[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/1-3)

[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/2)

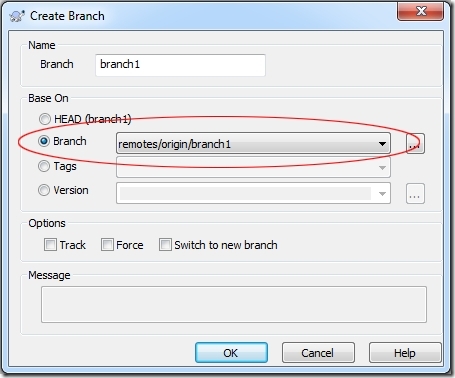
如果再次PUSH本地分支内容到中心服务器的时候，此时在TortoiseGit的PUSH对话框保持远程分支为空白的时候，中心服务器会首先查看远程是包含与本地分支同名分支，有则直接将本地版本上传到该分支.

[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/3)

### 获取远程分支（Fetching the remote branch）

有时候我们想要从远程服务器Fetch获取最新上传的新分支，有人可能会想可通过先在本地生成一个新分支，然后再pull远程我们想要的分支到我们在本地刚刚新建的分支上。通过pull方式实际上是，先在本地生成分支再通过pull方式将远程分支合并到本地生成的分支上.

我们或许还可以有另外一种方式：本地生成一个基于远程分支的分支，也可以达到同样效果，而且更方便。

[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/1658_1992)

注意，勾选“Base on“时，新建分支默认与远程分支同名，所以为了避免混淆，尽量保持本地分支与远程分支名称一致，即可

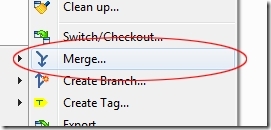
在生成新分支时勾选”Switch to new branch“，将在生成新分支之后直接将当前工作目录转换到新建分支上

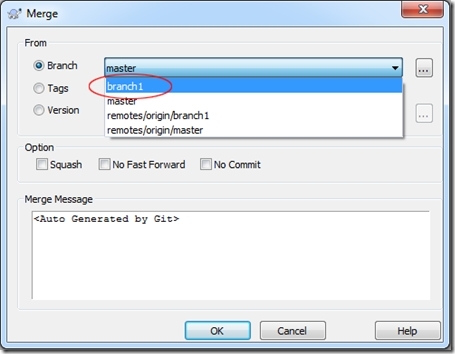
### 分区合并（Merging Branch）

完成某分支工作后，将会考虑将该分支上的工作合并到主分支上面，这时我们将进行分支合并操作.

进行分支合并之前我们需要明确哪个分支将要合并到哪个分支，首先要“CheckOut/Switch”到将要合并到的分支（如master分支）,然后通过“Merge”菜单打开分支合并对话框，选择需要合并的分支。

分支合并成功后，我们即可以通过PUSH操作将合并上传到中心服务器。

[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/1-6)

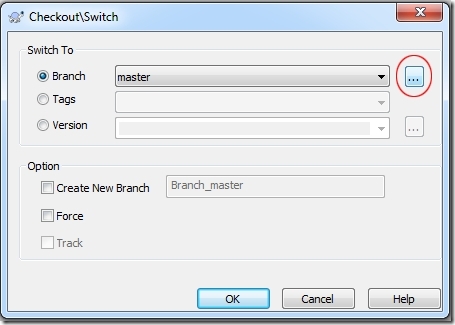
[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/2-4)

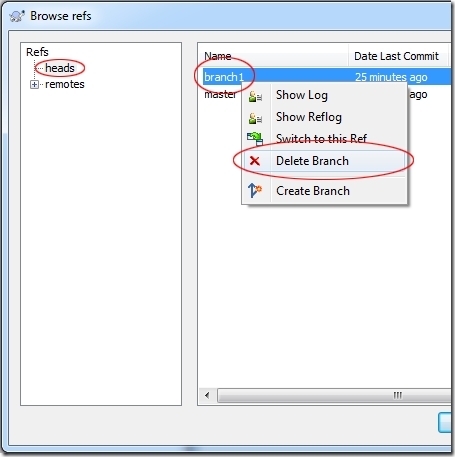
### 删除本地/远程分支(Deleting the local/remote branch)

当我们将分支合并到主分支，或者放弃该分支的时候，可以对分支进行删除操作.

在TortoiseGit上进行分支删除操作非常简单，首先打开“CheckOut/Switch”对话框，通过“Switch to 》Branch”更多按钮打开分支列表,或者通过菜单“Browser References”打开分支列表（默认该菜单是隐藏的）.选择相应的分支，单击右键，选择删除.

*注意，在删除远程分支的时候，本地分支并不会删除，这也说明了本地分支与远程分支并无从属关系。*

[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/1-5)

**[](http://doophp.sinaapp.com/archives/other/branches-options-with-tortoisegit.html/attachment/2-3)**

## Ignore、exclude、Assume Unchanged

Ignore：不加入ignore的文件。但是pull时会拉取ignore的文件到本地（如ignore了本地的a.txt文件，则commit和push时不会加入a.txt文件到服务器中，但是pull时会将服务器的a.txt的修改拉倒本地！）

exclude：功能与ignore一样，设置的是本地需要排除的文件。 不会影响到其他人。

不同点就是不会修改. gitignore文件。

Assume Unchanged：忽略已追踪文件的改动。

例如，工程里已经有了pom.xml，并且已经放到版本库，这时候我想修改下pom.xml让其适配自己的环境（比如设置连自己的测试数据库），但是又不想不小心push到远程仓库。由于已经存在于版本库，

因此上面提到的两种方式都已经失效了。查阅文档，发现有办法解决，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 本地修改不提交到远程仓库<br><br>git update-index --assume-unchanged index.jsp    取消本地忽略    git update-index --no-assume-unchanged index.jsp    查看本地仓库哪些文件被加入忽略列表    git  ls-files -v |

优点：本地命令，设置不会提交到remote，只需设置一次即可。  
缺点：不支持正则表达式，也不支持文件夹，只支持单个文件的添加。

### 具体做法

Ignore做法：

可以设置项目中的.gitignore文件（没有则增加该文件），设置ignore的文件

规则：

# 此为注释 – 将被 Git 忽略

\*.a # 忽略所有 .a 结尾的文件

!lib.a # 但 lib.a 除外

/TODO # 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO

build/ # 忽略 build/ 目录下的所有文件

doc/\*.txt # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt

这样设置了以后 所有的 .pyc 文件都不会添加到版本库中去。

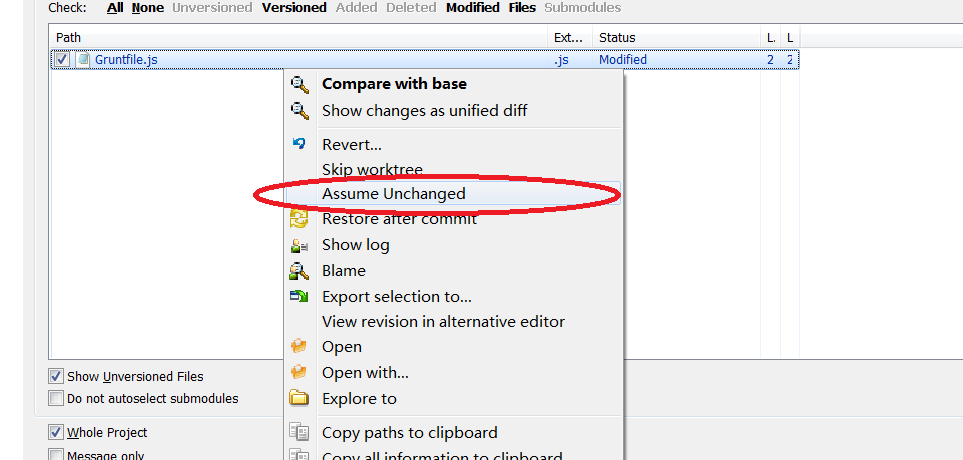
exclude做法：

.git->info->exclude 设置本地需要排除的文件

规则同ignore

Assume Unchanged做法：

可以使用TortoiseGit->commit，然后右击该文件，选择Assume Unchanged，这样就不会将修改后的该文件提交上去：



取消Assume Unchanged的话要使用命令：

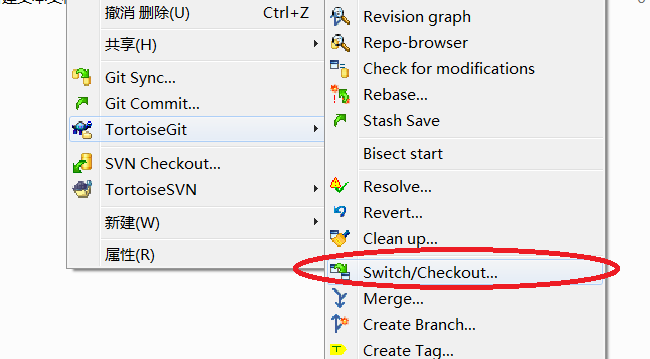
如git update-index --no-assume-unchanged index.jsp

# 专题研究

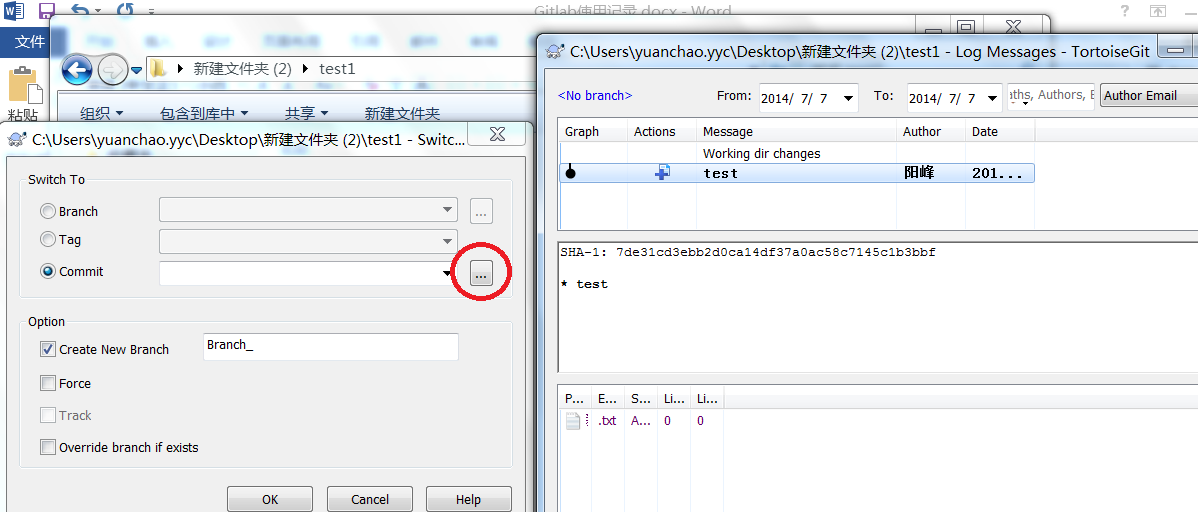
## 从服务器pull拉取指定版本的项目

### 使用TortoiseGit

使用Swtich/Checkout选项：



可pull指定branch或者tag或者commit版本的项目：



### 使用git bash

git clone下载下来的就是完整的版本库, 默认checkout出HEAD分支, 如果要checkout出之前的版本, 可以通过git log获取之前的提交信息, 然后找到想要checkout的部分"git checkout <SHA1 ID的前8位(如76bd774c)>"就可以吧之前时间提交的版本checkout出来, 如果要checkout到仓库其他的分支, 先用git branch -a查看分支, 再用命令: "git checkout -b <new\_branch\_name> <remote\_branch\_name>" checkout出remote\_branch\_name这个分支出来.

## 初次运行 Git 前的配置（git -config）

一般在新的系统上，我们都需要先配置下自己的 Git 工作环境。配置工作只需一次，以后升级时还会沿用现在的配置。当然，如果需要，你随时可以用相同的命令修改已有的配置。

Git 提供了一个叫做 git config 的工具（译注：实际是 git-config 命令，只不过可以通过 git 加一个名字来呼叫此命令。），专门用来配置或读取相应的工作环境变量。而正是由这些环境变量，决定了 Git 在各个环节的具体工作方式和行为。这些变量可以存放在以下三个不同的地方：

* /etc/gitconfig 文件：系统中对所有用户都普遍适用的配置。若使用 git config 时用 --system 选项，读写的就是这个文件。
* ~/.gitconfig 文件：用户目录下的配置文件只适用于该用户。若使用 git config 时用 --global 选项，读写的就是这个文件。
* 当前项目的 Git 目录中的配置文件（也就是工作目录中的 .git/config 文件）：这里的配置仅仅针对当前项目有效。每一个级别的配置都会覆盖上层的相同配置，所以 .git/config 里的配置会覆盖/etc/gitconfig 中的同名变量。

在 Windows 系统上，Git 会找寻用户主目录下的 .gitconfig 文件。主目录即 $HOME 变量指定的目录，一般都是 C:\Documents and Settings\$USER。此外，Git 还会尝试找寻 /etc/gitconfig 文件，只不过看当初 Git 装在什么目录，就以此作为根目录来定位。

### [用户信息](http://git-scm.com/book/zh/%E8%B5%B7%E6%AD%A5-%E5%88%9D%E6%AC%A1%E8%BF%90%E8%A1%8C-Git-%E5%89%8D%E7%9A%84%E9%85%8D%E7%BD%AE#用户信息)

第一个要配置的是你个人的用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，所以会随更新内容一起被永久纳入历史记录：

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

如果用了 --global 选项，那么更改的配置文件就是位于你用户主目录下的那个，以后你所有的项目都会默认使用这里配置的用户信息。如果要在某个特定的项目中使用其他名字或者电邮，只要去掉 --global选项重新配置即可，新的设定保存在当前项目的 .git/config 文件里。

### [文本编辑器](http://git-scm.com/book/zh/%E8%B5%B7%E6%AD%A5-%E5%88%9D%E6%AC%A1%E8%BF%90%E8%A1%8C-Git-%E5%89%8D%E7%9A%84%E9%85%8D%E7%BD%AE#文本编辑器)

接下来要设置的是默认使用的文本编辑器。Git 需要你输入一些额外消息的时候，会自动调用一个外部文本编辑器给你用。默认会使用操作系统指定的默认编辑器，一般可能会是 Vi 或者 Vim。如果你有其他偏好，比如 Emacs 的话，可以重新设置：

$ git config --global core.editor emacs

### [差异分析工具](http://git-scm.com/book/zh/%E8%B5%B7%E6%AD%A5-%E5%88%9D%E6%AC%A1%E8%BF%90%E8%A1%8C-Git-%E5%89%8D%E7%9A%84%E9%85%8D%E7%BD%AE#差异分析工具)

还有一个比较常用的是，在解决合并冲突时使用哪种差异分析工具。比如要改用 vimdiff 的话：

$ git config --global merge.tool vimdiff

Git 可以理解 kdiff3，tkdiff，meld，xxdiff，emerge，vimdiff，gvimdiff，ecmerge，和 opendiff 等合并工具的输出信息。当然，你也可以指定使用自己开发的工具，具体怎么做可以参阅第七章。

### [查看配置信息](http://git-scm.com/book/zh/%E8%B5%B7%E6%AD%A5-%E5%88%9D%E6%AC%A1%E8%BF%90%E8%A1%8C-Git-%E5%89%8D%E7%9A%84%E9%85%8D%E7%BD%AE#查看配置信息)

要检查已有的配置信息，可以使用 git config --list 命令：

$ git config --list

user.name=Scott Chacon

user.email=schacon@gmail.com

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

...

有时候会看到重复的变量名，那就说明它们来自不同的配置文件（比如 /etc/gitconfig 和~/.gitconfig），不过最终 Git 实际采用的是最后一个。

也可以直接查阅某个环境变量的设定，只要把特定的名字跟在后面即可，像这样：

$ git config user.name

Scott Chacon

## [git修改文件大小写](http://www.4wei.cn/archives/1002314)

两步走，使用git mv -f来移动文件，-f是强制覆盖已经存在的文件，然后修改git忽略大小写的设置:ignorecase = false

You can try:  
1. "git mv -f foo.txt Foo.txt"  
2. set `ignorecase' to false in the config file

[帮助](http://www.4wei.cn/archives/1002314)

|  |  |
| --- | --- |
| 1  2  3  4 | vim .git/config  ignorecase = false  git mv -f Foo foo  git commit -am "recase foo" |

当然，也有其它同学的办法，但这样会产生两次commit

[帮助](http://www.4wei.cn/archives/1002314)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | mv foo foo2  git add -A  git commit -m "renaming"  mv foo2 FOO  git add -A  git commit --amend -m "renamed foo to FOO" |

# 基础知识

## Fetch和pull区别

1. git fetch：相当于是从远程获取最新版本到本地，不会自动merge

git fetch origin master  
git log -p master..origin/master  
git merge origin/master

    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  
   上述过程其实可以用以下更清晰的方式来进行：

git fetch origin master:tmp

git diff tmp

git merge tmp

    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  
2. git pull：相当于是从远程获取最新版本并merge到本地

git pull origin master

上述命令其实相当于git fetch 和 git merge  
在实际使用中，git fetch更安全一些  
因为在merge前，我们可以查看更新情况，然后再决定是否合并

结束

## 标签Tag和分支Branch的区别

项目默认有个master分支，该分支为主分支，用户可以创建自己的私有分支，从而能够独立私有地在自己分支中修改

Tag是一个符号，映射具体的某次commit。

一般来说，可以在分支中自己修改，然后修改完成后merge到master分支中，然后提交该次merge，并打上tag后push到服务器中。这样该tag就映射本次merge提交了。

# gitlab服务端说明

项目的issues中可发布项目信息，如开发时遇到的坑等。

# 已解决的问题

## 1、公司的gitlab的push时，用户名和邮箱不是公司中我使用的用户名和邮箱，导致push失败

**问题描述**

我commit时，config是我的用户名和邮箱

Push时，config仍然是我的用户名和邮箱，结果导致push失败（失败信息提示修改用户名和邮箱）

**原因分析**

公司的git服务端对author信息合法性进行了强制性校验。对非法的私人邮箱将拒绝push。

**解决方案**

失败信息提示我修改config的用户名和邮箱为公司中我使用的用户名和邮箱后，使用git-m工具（用法参考http://baike.corp.taobao.com/index.php/Git-m）

windows用户如果安装了命令行git(msysgit)[下载](http://tools.taobao.net/site/store/product_detail.htm?product_id=283)，可以运行(更新于2013-1-4)：

curl http://openbase.oss-cn-hangzhou.aliyuncs.com/git-m -o git-m

我在项目根目录运行git-m工具后，仍然push失败！

**最后我找到的办法就是reset到前一次push，这样就可以放弃本次commit**：

