**Windows Git+TortoiseGit简易使用教程**

时间：2014-11-13 18:24:34      阅读：9656      评论：0      收藏：0      [点我收藏+]

标签：[blog](http://www.mamicode.com/so/1/blog)   [http](http://www.mamicode.com/so/1/http)   [ar](http://www.mamicode.com/so/1/ar)   [os](http://www.mamicode.com/so/1/os)   [使用](http://www.mamicode.com/so/1/%e4%bd%bf%e7%94%a8)   [sp](http://www.mamicode.com/so/1/sp)   [for](http://www.mamicode.com/so/1/for)   [文件](http://www.mamicode.com/so/1/%e6%96%87%e4%bb%b6)   [on](http://www.mamicode.com/so/1/on)

转载自 [http://blog.csdn.net/jarelzhou/article/details/8256139](http://tortoisegit.org/docs/tortoisegit/)

官方教程：<http://tortoisegit.org/docs/tortoisegit/>（英文版）

**为什么选择Git**

**效率**

很多人有一种习惯吧，什么软件都要最新的，最好的。其实吧，软件就是工具，生产力工具，为的是提高我们的生产力。如果现有的工具已经可以满足生产力要求了，就没有必要换了。生产效率高低应当是选择工具的第一位。

**历史**

开源世界的版本控制系统，经历了这么几代：

第一代，CVS，开创性的产品，至今很多古老项目仍在使用，但存在不少设计上的缺陷。

第二代，SVN，在CVS基础上大幅度改进，很快就取代了CVS的地位，现在的项目数量仍然非常庞大，而且SVN的缺陷不算多，很多商业项目似乎没有迁移到下一代版本控制系统的打算。比如Chrome就是使用SVN的。

第三代，Hg与Git。说起这两个，有种既生瑜，何生亮的感脚。这两个项目几乎是同时启动的，目标都是为Linux kernel提供一个开源可靠的分布式版本控制系统，同样优秀，只不过Git是Linus亲自带队做的，于是Git就成了Linux的选择。除此之 外，Eclipse从原本的SVN转移到了Git，而Firefox与Python则使用Hg，国内的金山软件也使用Hg，而Cocos2d则是Git， 孰优孰劣，不好说。

总的来说吧，前两代由于集中式管理的缺陷，现在用的肯定会越来越少，不过老项目往往不会轻易换。Git胜在功能强大，分支完善，败在设计复杂，难以理解。Hg胜在清晰易懂，功能完善，败在不利于从SVN迁移，相应的社区活跃度不如Github。

我们暂时会使用git，也许以后也会尝试另一个。

**TortoiseXXX是什么东西**

也许你已经看到了TortoiseSVN、TortoiseGit与TortoiseHg这三个项目，你也许想知道这三个是什么东西。事实 上，SVN、Git与Hg都是命令行的程序，对于大多数人来说，尤其是Windows平台下，难以上手，门槛较大，而这三个项目正是解决这个问题。 TortoiseXXX会在Windows的右键菜单下添加绝大多数的版本控制功能，免去了命令行的麻烦。但是说实话，右键菜单的效率和命令行的效率哪一 个高，不好说，不过TortoiseXXX还是降低了门槛，对于初学者有很大的帮助，毕竟不是谁都能熟记那么多的命令的。

**准备工作**

**软件安装**

我指的是Windows下。由于Git从来没有官方的windows版（可以理解，linus做的嘛），windows下有两种非官方使用途径，一 个是Cygwin，另一个是msysGit。Cygwin比较凶残，直接把Gnu工具集转移到了windows上，不过不推荐，cygwin这个东西，如 果不做嵌入式，不需要交叉编译，用不着这么凶残，msysGit就足够了，也更简单。目前是1.8.0。

TortoiseGit同时支持两种使用Git的方式。它只是一个壳，实际调用的还是Git本身。这两个软件安装也没有什么好说的，除了安装路径之外，其他的全部默认就OK。

TortoiseGit安装后，请设置好你的姓名邮箱，这将会记录到版本信息中。开始菜单找到tortoisegit的settings就可以了，不用细说吧？

**密钥**

Git的身份验证不是密码，而是密钥。通常我们会有一个Git服务器，而我们的电脑就是工作机，我们需要一个RSA密钥来确认身份。密钥分为两个： 私有的和公开的，其实是这样，公开的存放在服务器上，而私有的放在你手里，核对的时候通过一系列复杂算法就可以匹配了。私有密钥绝对不要被其他人看到，这 是一个项目中，确认每个人身份的标识。

密钥可以用命令行生成一个。不过我们有TortoiseGit，看一下TortoiseGit的开始菜单，是不是有一个叫Puttygen的东西？ 密钥这个东西有很多工具都可以生成，Puttygen就是个图形化的工具。使用很简单，只要运行Puttygen，把右下角的数值改成4096（这个是最 长的）然后点击“Generate”就可以了。点击之后它会要求你随机移动鼠标，直到进度条满为止。

生成完之后呢？千万不要关闭这个窗口！要是手贱关了，那么……再来一遍吧。你要在passphrase和下面的确认框输入密码，这个密码你自己写咯，不过记着哦。写好之后，点“save private key”保存就行。

Public key是填在服务器上的，我们以github为例，如果是别的代码托管网站，或者你自己搭建，原理都一样。打开你的Github账户，Edit YourProfile，然后SSH keys，然后add，然后把putty生成的public key贴进去就行了。

**Github的那点事**

Github是开源社区目前最活跃的项目托管网站，类似的网站还有SourceForge，bitbucket，google code等，每个网站支持的版本控制系统不一样，主要就是SVN、Git与Hg，github仅支持Git，其他协议需要很麻烦的插件来扩展，一般很少会 用。

尽管Git本身是分布式的，但我们通常仍然需要一个中心的作为标准的服务器，方便协同开发。Github就是起到了这个作用。

通常我们这样进行协同开发：项目有几个成员，创建者在Github上创建项目，并把其他人添加到项目中。项目组成员可以直接向中心库提交代码，而其 他人如果想参与进来，就需要用“fork”创建一个项目的分支副本，然后向原始项目提交合并申请，非项目组成员的提交是需要批准才能合并的。这也是通常的 开源软件的开发模式：核心成员主导方向，所有开发人员都可以参与进来。

当然了，非开源项目就没有那么麻烦，只要项目成员向中心库提交代码就行了。

**Git工作流程**

不要着急，我们在正式使用之前，需要对Git的基本工作流程有一个清晰的理解。

**从本地库单一分支开始**

Git是分布式的，因此，服务器上的库和你本地的库其实是同等的，两边的库都是具有完整的历史版本信息的。事实上，如果没有服务器，那么成员之间也是可以直接进行代码同步的，只不过这样会很混乱。这一点和SVN与CVS这样的集中式系统是很不同的。

我们先研究本地的单一分支库。

**文件状态**

文件的两种基本状态：被追踪，和不被追踪。

原因很简单，项目中有我们需要进行控制的代码文件，也有不需要的临时文件。如果我们新添加了一个文件，那么它默认是不被追踪的，你需要主动去添加到追踪文件中（add操作）。

对于被追踪的文件，有三种状态，已修改（modified），已暂存（staged），已提交（commit）。

已修改：你把原有的库中的某个文件进行了修改，此时由于文件的校验值发生了变化，Git会检测到你做了修改。此时我们需要做暂存的操作。

已暂存：在修改之后，你可以把修改暂时写入缓冲区，在积累了几次修改足够构成一次提交之后，再一次性提交（commit）到本地库。其实上面讲的“add”操作，完整的应该说是什么？应该是“add files to stage”。

已提交：注意这个提交是提交到本地库，因为本地也有一套完整的代码库。

暂存的历史不会被记录，只有commit到库中的信息才会被记录到版本信息库中，以后可以随时回滚查看。

可能有的时候你会觉得，中间的那个暂存区要来干嘛？很多余嘛。小项目当然完全无所谓，但是如果项目大了，那么版本的历史信息会很复杂的。这个时候，不必要的版本过多，也不好。暂存区就是用来防止一些不必要的commit。

**分支的作用**

分支是git的最突出的一个地方。

其实概念很简单，看名字就知道了，不过为什么要分支？

比如一个项目，现在要同时开发A功能和B功能，怎么处理？

其实也很简单，比如有一个中心库，然后你克隆到本地，做了一些修改，commit到版本A，而我呢，commit到版本B，然后呢？合并吗？合并的 结果是什么，合并之后，如果代码一点问题都没有，如果A和B两个功能我们不需要再回溯，那么什么都不是问题。但是往往现实中，如果A突然发生了问题怎么 办。

分支就要用上了：我们有一个主干分支，然后分别开一个A分支和B分支，你做你的，我做我的，然后确认无误之后，合并到主干。那这样，如果发生问题，是不是就比原来要容易解决多了？否则你都不知道问题出在了哪里。

**流程**

真正开发的时候，往往至少会有两个长期的分支（develop，master），最新的进度在develop中，而master则是稳定可靠的随时 可以发布的版本。当然，develop中的代码也是可运行的完整代码，不能有残缺的代码。开发中会有很多临时的分支，用来开发一些新功能，在确认无问题之 后，合并到develop中，然后删除临时分支。Develop中的代码不一定是合适发布的，可能是实验性的，在develop中的某个版本到了比较成熟 的时候，去和master合并。

当master中有bug了怎么办？那就新开一个bugfix分支，解决掉master的bug，同时再和master与develop合并（develop一般也会存在同样的bug嘛），然后删除临时分支。

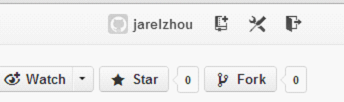
我们在开始工作前，先从服务器上拉去最新的版本，然后开发，然后把自己的版本合并到服务器上去。

**正式开始**

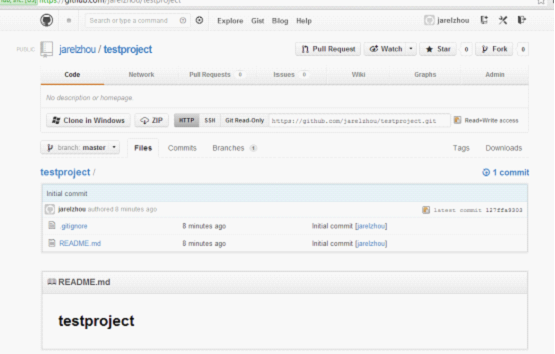
**建立github仓库**

建立仓库的操作需要在网站上进行。我们通常是在服务器上建立仓库（repository），然后克隆（clone）到本地。如果你想在本地直接建立仓库（init），那么通常是本地的单人代码库，或者是局域网内部共享的库，或者你就是代码服务器的管理员。

Github上建立仓库的操作很简单，打开网站，右上角右数第三个按钮就是了（看图）。你只需要输入项目名字，以及建立“.gitignore”文 件就行。这个文件其实就是我们明确指明不需要版本跟踪的文件，上文已经提到过了。通常这个文件是我们自己写的，但是github已经为我们的各种常见项目 提供了现成的样板，更加方便。



一般不会有什么问题，你填写完之后“create repository”就行了，我们会看到这样的页面：



这样就创建成功了。可以看到，现在初始的库中，有一个master分支，并且有了一次commit，作为初始化的操作。你也可以在本地建立仓库查 看，其实是一样的。不过我们通常不在本地建立仓库，如果你需要协同开发的话，因为你不能让服务器去克隆你电脑上的仓库，只能是你去克隆服务器的仓库。当然 了，如果你不打算用服务器的话，那是另外一回事了。

可以看到上面有项目的三种协议的地址：http，ssh，以及git协议的。这三种协议各有优缺点，都是指向同一个项目的。你还可以下载zip压缩 包，或者用github的windows客户端进行方便的克隆。这个客户端做的不错，不过它只能用在github上面，其他的代码托管平台不能用，这个就 不好了。我们还是用msysgit+tortoisegit的方式，对命令行熟悉的童鞋可以考虑直接上命令行啦。又扯开去了，不说了。

**克隆仓库到本地**

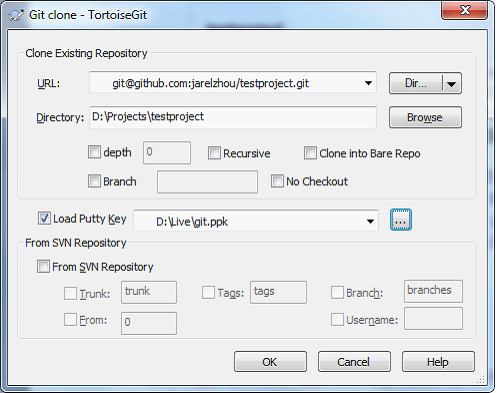
我们用ssh的协议为例。ssh的仓库地址大概是长这个样子：

[git@github.com:jarelzhou/testproject.git](mailto:git@github.com:jarelzhou/testproject.git)

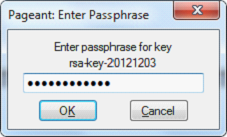
在你想要放本地仓库的地方，右键，git clone…

据说本地仓库不建议放在eclipse的workspace中，不知为何。题外话了，这样可能会造成项目间的混乱。

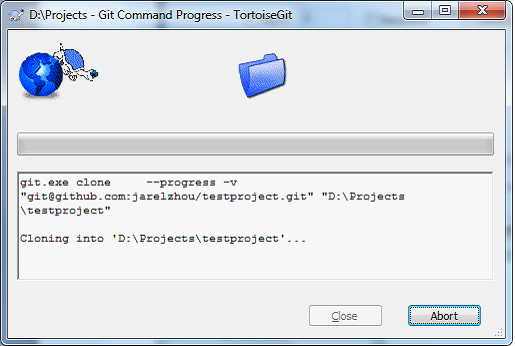
右键之后，看到这样的界面，url中填写之前github提供的ssh地址，putty key记得选择之前我们创建的private key。



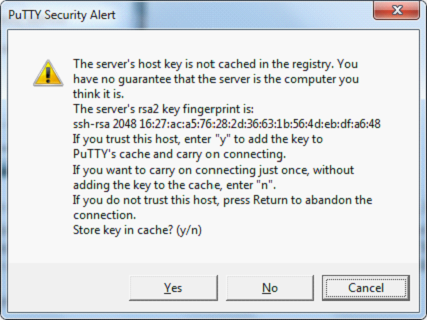
然后会要你输入passphrase，这也是之前设置好的：



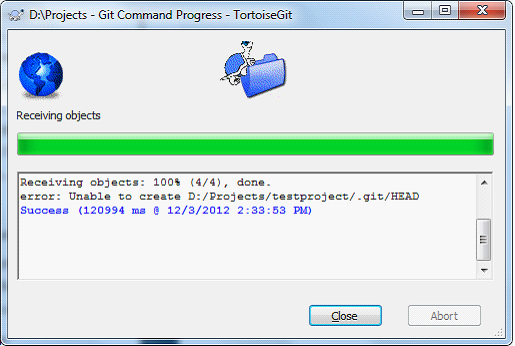
创建中：



然后是这个警告窗口，这个窗口只会出现一次，以后就不会有了，这是要你核对sshkey的，你可以核对一下：

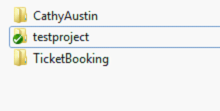


如果没错的话，点“Yes”就行了。过了一小会儿，会出来提示成功：



这个关掉就可以了。

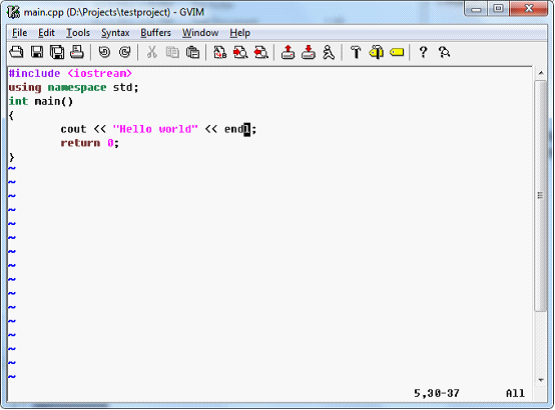
此时仓库信息中已经有了原本仓库的位置了，不再需要额外的设置。



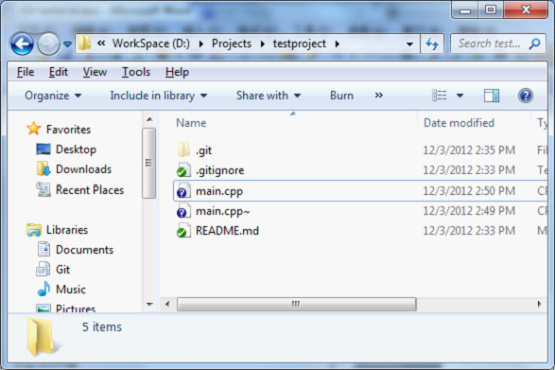
有没有看到，仓库已经克隆下来啦？被版本控制的目录会打上绿勾，如果没有的话你可以刷新一下看看，windows的图标缓存有bug，而且至今没有解决，刷新如果还没有的话可能得注销一下了。

**做一些改动**

我们现在来尝试给这个测试的项目添一点东西。比如添加一个main.cpp：



保存之后看到目录变成了这个样子：

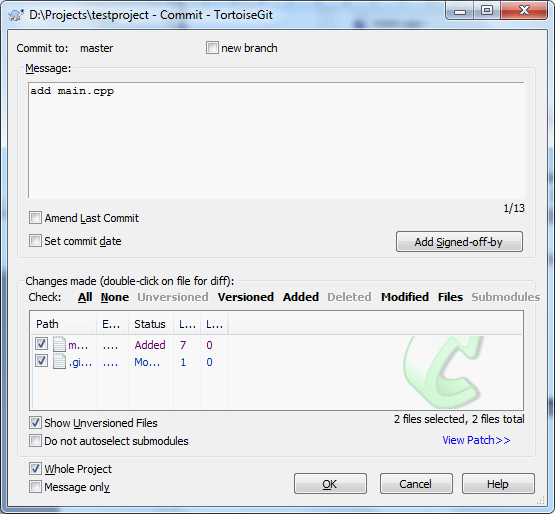


发现有一个~结尾的存档文件，不应该被跟踪的。右键文件，选择ignore即可。添加到忽略列表之后，你再刷新，就可以看到，main.cpp~这个文件上的图标消失了。

而main.cpp这个文件是我们要添加到控制中的。右键，Tortoisegit-》add，它就从未跟踪状态到被跟踪状态，而且是被添加到了stage中。

有一种说法，.gitignore本身也应该被忽略，不过这样的话可能大家添加的文件不一致，所以我们还是不忽略了。

这个时候，我们来commit一下，记录到本地版本库中。右键，commit即可。注意，git要求每次commit必须输入描述信息。

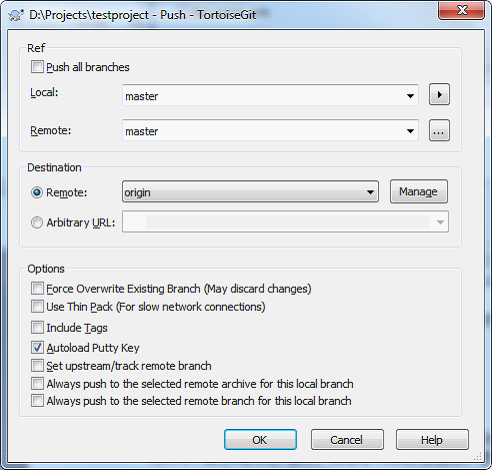


点OK就可以了。此时目录中所有文件又变成了绿勾的状态。

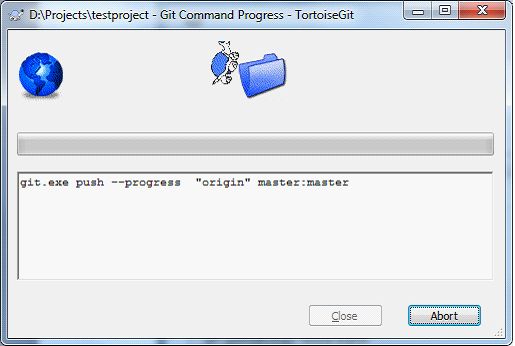
**推送到服务器端**

我们已经在本地做了修改，并commit到了本地的库。Git不允许直接commit到远程库，只允许两个库之间的同步操作，而没有commit的修改，是没有记录到仓库中的。现在，我们来推送（push）到服务器端。

仍旧是强大的右键……tortoisegit把几乎全部的操作放到了右键菜单中去。选择push：

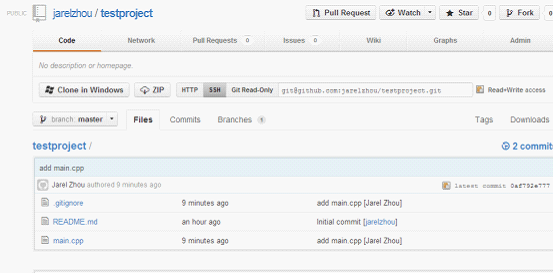


点OK就行了。然后又是一个乌龟翻跟斗的动画：



推送完了之后，关掉这个窗口就行了。

现在我们回到github的网站，刷新一下，看看，是不是文件被推送上去啦？



除此之外，还有pull操作，是从服务器端拉取最新的版本库，因为可能项目组的其他成员已经做了修改，你需要把其他人的改动同步下来。

**分支操作**

我们现在只有一个分支，更合理的设置是两个常设分支，然后按需出现的临时分支。建立分支的方法：右键，create branch，设置好这个分支是从哪个状态分叉出来的，默认是当前的工作状态，你也可以设成其他的。切换工作分支是checkout/switch。

在同步的时候，我们通常是每个分支单独同步的，你需要推送什么分支，就选择哪个分支，不要推送错了。Pull的时候，也是这样。Tortoisegit提供了拉取/推送全部分支的选项，不过你在做这个操作之前，必须要清楚你在做什么。

**最后**

写到这里，windows下tortoisegit的大概操作就说完了。你可以发现，我其实并没有过多地去描述tortoisegit怎么操作，我 更多的是在讲，git的设计，以及一种比较清晰的工作流程。这种工作流程不仅仅是tortoisegit，这在你用命令行的git，甚至是其他的版本控制 系统，都是差不多的。工具不重要，重要的是如何去用工具提高我们的效率。

写这篇文章到最后，我也把整个的过程理了一遍，对git也有了更深的理解。还是写下来比较清楚。