git 服务器的建立——Git 折腾小记

这两天刚完成了一个小项目，整理资料发现现在写代码跟文档又没有版本控制的意识，刚好前两天电脑出问题，差点所有代码跟文档全丢掉，所以这两天又在自己的台式机上架起了 git，做所谓的代码服务器，之前折腾过一次，因为时间原因没有总结下来，结果这次弄的时候，又得满世界找资料，所以这次准备总结一下，也做了一个参考资料汇总，以备以后再折腾时用。

git 简介

git 是干嘛的呢？记得听哪位大牛说：没有版本控制意识的程序员，都不是好程序员！如果你认同这句话，那么 Git 就是用来帮助你成为好程序的。Git 应该是当下最受人推崇的版本管理系统了（仅仅是感觉，没有依据），之所以受推崇，除了因为它是 Linux 内核开发者 Linus Torvalds 最初开发的，其确实有很多其他版本控制系统所没有的特点，否则其也很难管理像 Linux 内核项目这样的超大项目。那么它有什么特点了？

首先，与以前广为流行的 SVN 不同，git 是分布式的，其没有服务器端与客户端之分（虽然在项目管理过程中，一般会人为地指定某一台非开发用的机器作为“服务器”，但就 git 自身功能来说，完全可以没有这个台“服务器”，至于实际中为什么会有，后面将会讲到）。

然后，git 的分支（branch）与合并（merge）功能非常强大与智能。据维基百科所说：git 最为出色的就是它的合并跟踪（merge tracing）能力。作为 SVN 与 git 的用户，亲身体验告诉我，git 提供的分支间代码合并功能的确非常强大，很少出现需要手动合并代码的情况，即使出现，其提供的冲突提示与解决方案也非常方便跟简单。

最后，它是开源的，它正在变得越来越强大跟方便，同时，好用配套工具也越来越多，使得 git 的使用与管理更简单方便。

当然，它一样是有缺点的，比如其学习曲线相对比较抖，但是只要你想学，网上系统的资料越来越多了。本人现在也基本算入门，本文也仅仅是一个备忘录，权当一个可能不大好的入门材料。本文的最后列出了一些可以更系统学习的资料。同时，git 自身没有权限控制，需要额外的权限控制工具，但是现在也有不错的工具可以弥补了，本文也有举例介绍我自己当前在用的。

好，闲话说得够多了，下面开始介绍 git 的安装，包括 git 本身的安装，以及“服务器”上一些配套的管理工具的安装，包括用于权限管理的 gitolite，用户 web 浏览文件的 gitweb。

git 安装

git 安装

git 自身的安装其实是很简单的，不论是 windows 下还是 Linux 下（以我自己用的 ubuntu

为例），尤其是如果你仅作为“客户端”（即别人不会直接从你的电脑上 clone 代码，不会向你的电脑提交或者获取代码），你只需要跟安装一个普通软件一样傻瓜式安装即可：

windows 下直接下载可执行安装程序 msysgit （选择 git for windows 或者 msysgit 都可以），然后双击运行，根据提示安装即可，非常简单！

Linux 的安装也很简单，尤其如果你用的 Ubuntu 或者 Debian 等有本地包管理系统的 linux 系统，一条命令即可解决问题（Ubuntu 为例）：

```
sudo apt-get install git-core
```

非常简单吧！（当然，除了跨平台可能是 git 的优势跟特点外，安装简便肯定不是它的特点，因为很多其他软件也一样……）

但是，如果你是在为你的小组或者自己配置专门的代码服务器，供所有成员备份代码、共享代码、交流代码、合作开发，你可能就需要一些额外的工作了。仍然以 Ubuntu 系统为例，参考这篇文章，记录一下整个配置的过程：

首先，git 的数据交换跟交互是基于 ssh 的，所以为了使所有成员都能从该机器上获取和提交代码，需要给系统配置 ssh 服务，当然，如果你已经配置了 ssh 服务，那么这一步就可以省了，所以你可以先通过下面的命令查看下自己是否已经配置了 ssh 服务：

```
ps -ef|grep "sshd"
```

如果你能看到一些 sshd 相关的进程信息，则说明你已经有这个服务了，否则（或者你想更新的话），使用下面的命令安装 openssh

```
sudo apt-get install openssh-server openssh-client
```

然后，安装 git“服务器”

```
sudo apt-get install git-core
```

再然后，为自己配置身份信息，这样多个人提交代码的时候，就可以方便的查看是谁提交的，该如何联系 ta 了(如果该机器只做服务器，不做开发，本步骤应该可以省略)

```
git config --global user.name "yourname"
git config --global user.email "your@email.com"
```

以上步骤是每个 git 用户都需要的，接下来，作为“服务器”，为了更好的管理，我们需要进行一些必要的配置！包括环境的配置、管理工具的安装与配置等。

首先，我们最好为其配置一个专门的 git 用户并设置密码，专门对代码进行管理

```
sudo useradd -m git
```

sudo passwd git(change to yours)

说明：上述命令生成一个用户名与密码均为 git 的账户，也可以自己创建别的用户名跟密码，只要进行相关操作（比如 clone）时指定用户名即可（本文章一律以 git 为例），-m 选项是让其在 home 目录下生成用户的主文件夹，我们的代码仓库会布置在这个主文件夹下。

然后，在新建的 git 用户主目录下创建一个文件夹作为 git 的仓库，并为这个仓库配备最基本的安防——权限控制

sudo mkdir /home/git/repositories # 最好使用 repositories 作为文件夹名称，这样可以简化后面的操作

sudo chown git:git /home/git/repositories

sudo chmod 755 /home/git/repositories

至此，一个简单（非常简单）的 git 服务器已经搭建好了，可以自己创建一个简单的工程测试一下。

为了方便后面的操作，我们先切换到 git 用户下

su git

在 respositories 下新建一个目录（仓库），并切换到这个目录

mkdir helloworld

cd helloworld

在此处初始化一个空的仓库（只能接受 push/pull 代码，不能本地 commit）

git --bare init

好了，一个空的仓库建立好了，用另外一台安装了 git 的机器（比如你的开发机）测试一下，此处假设你上面所用的服务器 IP 为 192.168.0.123

git clone git@192.168.0.123:/home/git/repositories/helloworld

然后在开发机上进行一些基本操作测试：为这个项目 add 一些文件，然后 commit，然后 push，如果除了要几次输入 git 用户的密码外，其他一切正常的话，那说明上面的安装与配置就已经成功了

取库文件

git clone xxxx:// address

Git 可以根据不同的协议取库. 比如 file , http , git 等只要提供支持的都可以. 例如使用用本地文件中取库.

git clone file:///home/git/repositories/helloworld

这里本应该是 file:// 但在 Unix, Linux 系统中根目录是 “/” 所以我这里是以绝对地址.来定位.(本没有什么,只是经常大意了搞错了.)

添加文件

1.在 clone 出来的目录中新建文件夹 test，并新建文件 test.txt

mkdir test

cd test

vi test.txt

可以随便输入一些信息并保存退出。

2.

git add * #这里的 “*” 只是个通配符,直接用文件名也可以.

git commit –m “这里填上提取的日志”

简单的两个步骤就把文件提取到本地库中了,和 SVN 差不多.

3.

git push #提交到远程库中.

但在第一次提交时,可能会碰到

[python] view plain copy

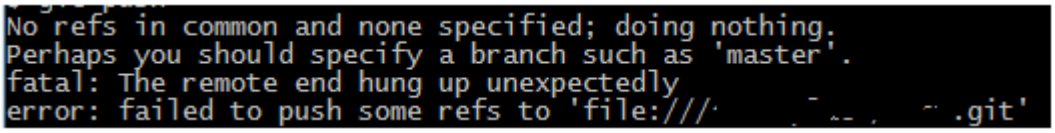
$ git push

No refs in common and none specified; doing nothing.

Perhaps you should specify a branch such as 'master'.

fatal: The remote end hung up unexpectedly

error: failed to push some refs to 'file:///xxxxxxx.git' 如下图:



哈哈,请不要惊慌.这是 Git 找不到你要提交的版本了.

那请试试

git push origin master

另外有时 push 之前需要 pull，而我在同一个虚拟机下操作的，操作时的用户为 djf，而 git 的用户为 git，所以 push 需要 sudo 才能成功。

删除文件

一般情况下，你通常直接在文件管理器中把没用的文件删了，或者用 `rm` 命令删了：

```
$ rm test -r
```

这个时候，Git 知道你删除了文件，因此，工作区和版本库就不一致了，`git status` 命令会立刻告诉你哪些文件被删除了:

```
# On branch master
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   deleted:    test/test.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

现在你有两个选择，一是确实要从版本库中删除该文件，那就用命令 `git rm` 删掉，并且 `git commit`：

```
$ git rm test -r

rm 'test/test.txt'

$ git commit -m "remove test.txt"

[master d17efd8] remove test.txt

 1 file changed, 1 deletion(-)

 delete mode 100644 test/test.txt
```

现在，文件就从本地版本库中被删除了。

接下来使用 git push 即可将服务器端更新，即删除此文件。

另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

```
$ git checkout -- test.txt
```

`git checkout`其实是用本地版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

如果是已经 push 了，则需要回退之前的版本，使用 git reset 命令。如下描述

reset 命令有 3 种方式：
1：git reset –mixed：此为默认方式，不带任何参数的 git reset，即时这种方式，它回退到某个版本，只保留源码，回退 commit 和 index 信息
2：git reset –soft：回退到某个版本，只回退了 commit 的信息，不会恢复到 index file 一级。如果还要提交，直接 commit 即可
3：git reset –hard：彻底回退到某个版本，本地的源码也会变为上一个版本的内容

git 版本

Git 可以为 SHA-1 值生成出简短且唯一的缩写。 如果你在 `git log` 后加上 `--abbrev-commit` 参数，输出结果里就会显示简短且唯一的值；默认使用七个字符，不过有时为了避免 SHA-1 的歧义，会增加字符数：

```
$ git log --abbrev-commit --pretty=oneline
```

```
ca82a6d changed the version number
085bb3b removed unnecessary test code
a11bef0 first commit
```

通常 8 到 10 个字符就已经足够在一个项目中避免 SHA-1 的歧义。

git 获取历史版本的几种方式

我们简单的描述一个例子：

a)初始化操作

有两个文件 file1.txt 和 file2.txt

1， 初始化的时候就有这两个文件

操作：

git init

git status

git add .

git commit -m “init version”

2， 在 master 分支上修改了 file1.txt，并提交

操作：

(修改 file1.txt)

git add file1.txt

git commit -m “change file1”

3， 然后新建分支 banana，并切换到 banana 分支上

操作：

git branch banana

git checkout banana

4， 修改 file1.txt 和 file2.txt，并提交。

操作：

(修改 file1.txt 和 file2.txt)

git add file1.txt file2.txt

git commit -m “change by banana”

这个时候，我们可以输入 gitk，查看一下当前的版本情况。如下图：git-001

5， 然后，切换到 master 分支上，修改 file2.txt，并提交。

操作：

git checkout master

(修改 file2.txt)

git add file2.txt

git commit -m “change by master”

b)发现问题需要查看历史版本

我们现在发现当前的版本有点问题，还不能提交到版本库。

1，我们需要从 git commit 中返回.

则输入：

git reset --soft HEAD^

解释一下，HEAD 是当前分支的最新版本。^表示父节点。当前节点的父节点，

就是上一次提交的版本。也就是标记为“change file1”的版本。

问为什么不是”change by banana”这个版本呢？不同的分支哦。”change by banana”是 banana 分支的最新代码，和 master 分支不同的。

这个时候输入

git status

看看，是不是显示 file2.txt 修改了没有提交呢。

2，我们需要从 git add 中返回

再仔细查看之后，我们发现 file2.txt 真的写错了，需要返回到 git add 之前的状态。

输入：

git reset -q file2.txt

这个时候，file2.txt 就回到了解放前了。用 git status 查看一下，file2.txt 是”change not staged for commit”状态。

3，回到没有做过的情况

我们最终确定，最后一次修改的 file2.txt 是无用的代码，我们需要废弃掉。

注意，这个操作不能恢复的哦。

git reset --hard

这个命令，不能指定具体的文件。是把当前的修改全部清除，恢复到最后一次提交的版本。

这个时候，用 gitk 查看一下：git-003

已经彻底回复到了“change file 1”的版本了。

4，直接回复到某个版本

我们现在切换到 banana 分支。

git checkout banana

然后用 gitk 看一下。可以看出，我们之前的操作，对 banana 分支一点影响也没有。现在我们需要把 banana 分支回复到初始状态，但是当前的改动的代码还是需要留着。我们可以看，init 版本是当前版本的父节点的父节点。我们可以这么操作：

git reset --soft HEAD^^

然后用 gitk 看一下：git-004

最近的版本已经变成了 init version 了。所有的改动都是 add 未提交状态

5，得到当前最新代码

最后。我们把 file1.txt 和 file2.txt 都删掉。我们需要从版本库中取得当前最新的代码。

很简单：

git checkout master

如果是要 banana 分支的最新代码，则：

git checkout banana

以上的操作，我们知道了如何查看版本分支，和如何回复到以前的版本。

git 工具与配置

在上面的测试过程中，是不是发现那个 clone 的路径好长，很容易错？几个步骤都需要输入 git 用户的密码（clone、push），是不是很烦，而且照这个节奏，每增加一个开发成员，就得告诉他 git 用户的密码，是不是既繁琐又不安全？所以，如果能 clone 的时候只要指定项目名称，clone/push/pull 自动完成身份认证，并且最好能对不同的仓库给予不同用户不同的权限，那就好了！接下来介绍的工具与配置就可以搞定这一切，让 git 仓库的管理变得更简

单，操作更方便！

首先，针对身份认证的问题，我们前面已经提到 git 的数据交换与操作都是基于 ssh 的，所以，我们的身份认证自然可以通过配置 ssh 来解决。ssh 是通过密钥进行认证管理的，密钥包括一个公钥（交给服务器）和一个私钥（自己保留），每个公钥对应一个私钥，每个私钥也只对应一个公钥。。。。此处省略若干字。。。。，所以一个简单的解决自动身份认证的方式就是：每个需要访问代码仓库的人员，在自己的机器上通过 ssh-keygen 生成自己的公钥与私钥，将公钥提交给服务器，服务器管理员将改用户的公钥添加到服务器 git 用户的.ssh/authorized_keys 文件中，可能用到的命令如下（Ubuntu 终端/Windows 下使用 Git Bash）：

[plain] view plaincopy

```
# 某开发机上
ssh-keygen #接下来一路回车就好了，在默认目录下生成默认密钥文件
cp ~/.ssh/id_rsa.pub /path/to/one/visiabl/fold/ #将隐藏文件夹下的公钥文件拷贝到一个可以文件夹下（如果接下来用 scp 提交，此步骤可省略）
# 将上面的公钥文件以某种方式提交给服务器

# 管理员在服务器下
# 将开发机提交上来的公钥文件，添加到/home/git/.ssh/authorized_keys 文件中（每行一个）
```

这种方式可以在开发人员较少，管理的仓库较少的时候使用，因为简单，但是其权限控制比较单一（均可以 read/update 代码），而且如果人一多，要管理 authorized_keys 文件也是个头疼的事情，所以不是很推荐这种方式，建议采用后面介绍的工具。
然后，对于默认路径的问题，这个我真的只知道通过下面的管理工具来配置了。所以，接下来介绍一个 git 仓库的开源管理工具 gitolite

gitolite 安装

gitolite 的官方介绍是这样的：Hosting git repositories -- Gitolite allows you to setup git hosting on a central server, with very fine-grained access control and many (many!) more powerful features.如其所述，它的主要功能就是对 git 仓库进行权限控制，并提供其他很多给力的方便管理的特性。其实我本来是要介绍 gitosis 的，但是自己在弄的时候，gitosis 的配置出了一个非常诡异的问题，不管我怎么配置，甚至重新安装从零开始配置，总是再我更新它的配置文件后，之前的配置就不起作用了，折腾了我一天，最终发现了它的升级版——gitolite，所以决定采用它了。它们的本质都是上面介绍的 ssh 验证，只是它们提供更方便的管理方式，然后自动生成 authorized_keys 文件。这两个工具最有意思的一点就是，它们自身就是一个特殊的 git 版本库（gitolite-admin），他们的管理与配置都可以通过 git 的方式，分布式

的进行修改，然后通过 push 的方式提交到服务器，服务器会通过所谓的钩子脚本自动更新权限控制文件。
下面介绍它的安装，它的官方主页上有详细的安装介绍，我这权当翻译跟经验分享。
首先，因为我的 git 仓库是在 git 用户下，所以，切换或者登陆到 git 用户下

[plain] view plaincopy

```
1. su git
```

上面说过，gitolite 本质就是根据你的配置，自动生成 authorized_keys 文件，所以它要求你的 authorized_keys 文件必须是空的，或者不存在，所以我们干脆删了它(请注意一定要切换到 git 用户，否则，误删除了其他用户下的 authorized_keys 文件导致服务器的其他功能受影响，那就悲剧了，这应该也算是为什么要专门弄一个 git 用户来管理的原因吧)

[plain] view plaincopy

```
1. rm ~/.ssh/authorized_keys
```

然后，gitolite 在初始化时需要通过某一用户的公钥文件指定一个超级管理员，gitolite 安装成功后，只有这个超级管理员可以更新 gitolite 以更新各种权限控制（包括对其自身的更新权限控制），所以在初始化时需要指定该超级管理员账户的公钥文件（最好直接将其拷贝到 git 用户的主文件夹下）（下面的示例程序使用同一服务器上的另一常用管理员用户 admin）

[plain] view plaincopy

```
1. su admin
2. ssh-keygen
3. sudo cp ~/.ssh/id_rsa.pub /home/git/id_rsa.pub
4. su git
```

好~准备工作已经完成了，开始安装 gitolite

[plain] view plaincopy

```
1. cd ~ # 回到 git 主文件夹下
2. git clone git://github.com/sitaramc/gitolite # 获取 gitolite 的源码
3. mkdir -p $HOME/bin  #为 gitolite 的二进制文件生成创建目录
4. gitolite/install -to $HOME/bin  # 编译生成安装文件
5. $HOME/bin/gitolite setup -pk id_rsa.pub # 安装并初始化，指定 id_rsa.pub 公钥文件对应的用户为超级管理员
```

Bingo! gitolite 安装完成！不过，你要用它来进行管理，那还需要一定的操作。上面提到，gitolite 安装后本身是一个特殊的 git 版本库——gitolite-admin，分布式的进行修改，然后通过 push 的方式提交，其会通过钩子脚本执行权限更新。看一下上述步骤的最后一步你会发现，gitolite 自动生成了两个版本库：gitolite-admin.git 和 testing.git，其中的 gitol-admin.git

就是那个特殊的神奇版本库。所以，接下来我们要做的，就是回到你刚刚指定的超级管理员账户的电脑跟账户下，clone 出 gitolite-admin 这个特殊的 git 版本库（当前情况下，只有该超级管理员账户可以 clone 并更新 gitolite-admin 这个版本库），然后根据自己的需要对其进行配置（如添加更多的管理员账户、添加新的版本库并为不同的用户指定权限）

[plain] view plaincopy

```
su admin # 回到指定的超级管理员账户
git clone git@192.168.0.123:gitolite-admin.git # clonegitolite-admin 这个特殊的版本库
```

如果上面的步骤都成功了的话，应该可以查看到有一个 gitolite-admin 的文件夹，文件夹下有两个目录 conf、keydir

如果你回到 git 用户，查看 repositories 目录（如果之前创建仓库时，创建的文件夹不是 repositories，gitolite 会自动创建这个文件夹，并将该文件夹作为默认访问时的默认路径），目录下就会多了 gitolite-admin.git 与 testing.git 两个版本库。

有没有发现，这次 clone 的时候，后面的路径变短了？密码也不用输了？腰不酸腿不痛了？对，你没猜错，这一切 gitolite 已经自动帮你搞定了：默认路径是/home/git/repositories，权限控制是只有当前的超级管理员用户可以访问 gitolite-admin 和 testing 两个版本库，你之前测试创建的版本库也已经无法访问，如果你尝试再次 clone 之前创建的测试版本库，应该就会提示如下错误信息：

[plain] view plaincopy

```
# 假设你之前创建了 helloworld 版本库，现在使用
git clone git@192.168.0.123:helloworld 试图 clone
FATAL: R any helloworld id_rsa DENIED by fallthru
(or you mis-spelled the reponame)
fatal: The remote end hung up unexppectedly
```

要继续访问之前创建的项目，需要将这个项目添加到 gitolite 的权限控制内，下面演示一下为当前的超级管理员用户指定之前创建的 helloworld 测试版本库的读写权限（可读可写），以此演示 gitolite 指定权限的一般流程：

1、将需要指定权限的用户的 ssh 公钥文件，存放在 gitolite-admin 版本库的 keydir 目录下（如果提交的都是 id_rsa.pub，可以将其重命名为该用户的 id 或者名称，同时也推荐这样重命名，以明示哪个公钥文件是哪个用户的），因为我们初始化时，gitolite 已经将该超级管理员的公钥文件自动拷进去了，所以省略此步骤

2、编辑 conf 目录下的 gitolite.conf 文件，添加 helloworld 版本库管理组，为超级管理员指定读写权限（RW+，具体的权限定义，参考 gitolite 官方文档）

[plain] view plaincopy

```
repo helloworld
    RW+  =  id_rsa
```

3、commit 到本地

[plain] view plaincopy

```
git commit -am 'add the helloworld repo and add RW+ to id_rsa'
```

4、push 到 git 仓库

[plain] view plaincopy

```
git push
```

如果 push 成功，当前超级管理员用户应该就可以成功 clone helloworld 版本库，并进行添加、删除、修改与 push 等操作了。
以上便是一个经典的管理过程，至于具体的权限控制、配置文件的格式、更多高级功能等，如果都介绍的话，这文章也太长了，所以读者还是参考官方文档（简单介绍，详细版本）吧，顺便学学英语。

一个基本的 git 服务器算基本完成了，当然如果要管理好，里面涉及到管理细节（不论是技术细节，还是管理策略）还有很多很多，我自己现在也只知道些皮毛，同时因为没有大项目的代码管理经验，都是我自己的小打小闹，所以就完全不敢出来献丑了，更多技术细节，大家可以查看官方文档，有机会我也许会再做点笔记分享，至于管理策略，推荐这个:A successful Git branching model ，这个也有”中文版“

git 基本操作

其实这个已经有很多不错的资料了，所以也就懒得自己写了，罗列一点自己看过的资料好了：
我第一次接触 git 时，大哥推荐给我的资料：Git 魔法，有 pdf 版本的，读者自己找找吧，如果没找到，也可以找我要，仅限学习~
git 官网貌似一直就没上去过，看看这个中文翻译版本吧
一个不错的 git 简易指南

1.在服务器端创建专用帐号，所有用户通过此帐号访问 git 库，一般方便易记，选择 git 作为专用帐号名称。

$sudo adduser --system --shell /bin/bash --group git

添加 git 用户到 ssh 用户组中

$sudo adduser git ssh

为 git 用户设置口令，当整个 git 服务配置完成，最好取消 git 口令，只允许公匙认证。

$sudo passwd git

2.到管理员主机将管理员公匙添加到服务器主机的.ssh/authorized_keys 文件中，建立新的公匙认证，如：

$ssh-copy-id -i .ssh/<filename>.pub git@server

3.服务器切换到 git 用户

$su git

安装 gitolite

$sudo apt-get install gitolite

执行 gitolite 安装

$gl-setup /<filename>.pub 以管理员公匙安装 gitolite

安装过程会询问是否修改配置文件，一般会自动打开 vi 编辑.gitolite.rc 文件。有些配置需要修改

$REPO_BASE="repositories"

用于设置 Git 服务器的根目录，缺省是 git 用户主目录下的 repositories 目录，可以使用绝对路径。所有 git 库都

部署在该路径下。

$REPO_UMASK=0007；#gets you 'rwxrwx--'

版本库创建使用的掩码。即新建立的版本库权限为‘rwxrwx’

$GL_BIG_CONFIG=0

如果授权文件非常复杂，更改此项配置为 1，以免产生庞大的授权编译文件。

$GL_DILDREPOS=1

支持通配符版本库授权。

退出保存。

如果安装时没有配置，后续可以打开 git 用户跟目录下.gitolate.rc 文件配置。

4.管理 gitolite

当 gitolite 安装完成后，在服务器端创建了一个用于管理 gitolite 的库 gitolite-admin.git

切换到管理员主机 $git clone git@server:gitolite-admin.git

$cd gitolite-admin

$ls -F

conf/ keydir/

$ls /conf

gitolite.conf

$ls keydir/

<filename>.pub

我们可以看出 gitolite-admin 目录下有两个目录 conf/和 keydir/.

keydir/<filename>.pub 文件

目录 keydir 下初始时只有一个用户公匙，及管理员用户的公匙。

conf/gitolite.conf 文件

该文件为授权文件，初始内容为

#gitolite conf

please see conf/example.conf for details on syntax and features

repo gitolite-admin

    RW+   = admin

repo testing

    RW+   = @all

缺省授权文件中只设置了两个版本库的授权：

    gitolite-admin

    即本版本库（gitolite 管理版本库）只有 admin 用户有读写和强制更新的权限。

    testing

    缺省设置的测试版本库，设置为任何人都可以读写以及强制更新。

5.增加新用户

只用将信用户的公匙添加到 gitolite-admin 版本库的 Keydir 目录下，即完成新用户的添加。

如：

$ cp /path/to/dev1.pub keydir/

$ cp /path/to/dev2.pub keydir/

$ cp /path/to/jiangxin.pub keydir/

执行 git add 命令，将公钥添加入版本库。

$ git add keydir

$ git status

On branch master

Changes to be committed:

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

#

```
#       new file:   keydir/dev1.pub
#       new file:   keydir/dev2.pub
#       new file:   keydir/jiangxin.pub
#
$ git commit -m "add user: jiangxin, dev1, dev2"
[master bd81884] add user: jiangxin, dev1, dev2
 3 files changed, 3 insertions(+), 0 deletions(-)
 create mode 100644 keydir/dev1.pub
 create mode 100644 keydir/dev2.pub
 create mode 100644 keydir/jiangxin.pub
```

执行 git push，同步到服务器，才真正完成新用户的添加。

```
$ git push
Counting objects: 8, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.38 KiB, done.
Total 6 (delta 0), reused 0 (delta 0)
remote: Already on 'master'
remote:
remote:                 ***** WARNING *****
remote:         the following users (pubkey files in parens) do not appear in the config file:
remote: dev1(dev1.pub),dev2(dev2.pub),jiangxin(jiangxin.pub)
```

6.更改授权

新用户添加完毕，可能需要重新进行授权。更改授权的方法也非常简单，即修改 conf/gitolite.conf 配置文件，提交并 push。

管理员进入 gitolite-admin 本地克隆版本库中，编辑 conf/gitolite.conf 。

```
$ vi conf/gitolite.conf
```

授权指令比较复杂，我们先通过建立新用户组尝试一下更改授权文件。

考虑到之前我们增加了三个用户公钥之后，服务器端发出了用户尚未在授权文件中出现的警告。我们就在这个示例中解决这个问题。

例如我们在其中加入用户组 @team1，将新添加的用户 jiangxin, dev1, dev2 都归属到这个组中。

我们只需要在 conf/gitolite.conf 文件的文件头加入如下指令。用户之间用空格分隔。

```
@team1 = dev1 dev2 jiangxin
```

编辑结束，提交改动。

$ git add conf/gitolite.conf

$ git commit -q -m "new team @team1 auth for repo testing."

执行 git push ，同步到服务器，才真正完成授权文件的编辑

7.创建仓库示例

在 conf/gitolite.conf 中添加类似下面的内容进去

repo notes

  PW = <filename>

保存，提交，并推送到服务器

git add -u

git commit -m 'add new repo notes '

git push

推送的时候应该看到类似这样的信息

Counting objects: 7, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (4/4), 395 bytes, done.

Total 4 (delta 1), reused 0 (delta 0)

remote: Initialized empty Git repository in /home/git/repositories/notes.git/

To git@desktop:gitolite-admin

   6de90b8..52737aa master -> master

注意 remote 开头的一行,它已经帮你创建了这个仓库

通配符创建仓库示例

通配符仓库事先不能确定名字,所以不会帮你创建,在你 clone 的时候才会创建

编辑 conf/gitolite.conf 文件在里面加入类似下面的内容

repo e2source/.+$

  C   =   <filename>

  RW+C   =   <filename>

注意 C = username 的一行必不可少,这里的 C 是指创建仓库的意思,下一行的 RW+C 中的 C 是指创建引用(branch,tag)的意思

保存后提交并推送到服务上去

git add -u

git commit -m 'add wildcard repo'

git push

注意看 push 时输出的信息,应该没有创建仓库的信息

这时 filename 克隆仓库的时候会自动创建

as filename user

git clone git@server:e2source/enigam2-plugins.git

输出应该类似这样

Cloning into 'enigam2-plugins'...

Initialized empty Git repository in /home/git/repositories/e2source/enigam2-plugins.git/

warning: You appear to have cloned an empty repository.

如果你的输出报这样的错

FATAL: R any e2source/enigam2-plugins jenny DENIED by fallthru

(or you mis-spelled the reponame)

fatal: The remote end hung up unexpectedly

一般是没有 C = username 这一行,注意是只有 C 的一行