

## git简介

**笔记本:** 廖雪峰 - git教程

**创建时间:** 2019/3/3 15:17

**更新时间:** 2019/3/3 16:04

**作者:** 刘坤鑫

**URL:** <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

---

Git是目前世界上最先进的分布式版本控制系统

神犇 Linux 开发

集中式vs分布式：简单来说，分布式就是，在每个人的电脑都是一个完整的版本库，无需联网从中央服务器下载最新版本即可工作，当然也有像集中式那样有一个中央服务器负责收集各方的修改内容和维护版本号。

## 安装git

笔记本: 廖雪峰 - git教程

创建时间: 2019/3/3 16:04

更新时间: 2019/3/3 16:04

作者: 刘坤鑫

---

安装后在git bash中输入以自报家门

```
$ git config --global user.name "Your Name"  
$ git config --global user.email "email@example.com"
```

## 基本操作

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/3 16:05

更新时间： 2019/3/3 16:46

作者： 刘坤鑫

URL: <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

版本库又名仓库，英文名**repository**

创建一个目录以作为仓库存储地方，git bash进入之，用命令初始化

```
$ git init
```

添加一个**文本文件 (UTF-8)**：

1.

```
$ git add readme.txt
```

2.

```
$ git commit -m "写了一个readme.txt文件:)"
```

修改readme.txt后，使用命令查看仓库变化，显示readme.txt被修改了，但还没有提交修改

```
$ git status
```

查看文件被修改的具体内容

```
$ git diff readme.txt
```

然后提交同样有两个——add和commit

查看历史纪录，可选参数使之变得紧凑

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

版本回退：

在Git中，用HEAD表示当前版本，上一个版本就是HEAD<sup>^</sup>，上上一个版本就是HEAD<sup>^^</sup>，当然往上100个版本写100个<sup>^</sup>比较容易数不过来，所以写成HEAD~100。

回退：

```
$ git reset --hard HEAD^
```

回到未来：

必须找到未来那个版本的版本号（commit id），只需要前几位即可，git会自己找，如果找不到，用命令

```
$ git reflog
```

找到版本号后，用命令回到未来：

```
$ git reset --hard 90422
```

## 工作区和暂存区

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/3 16:54

更新时间： 2019/3/3 16:57

作者： 刘坤鑫

URL: <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

工作区 (Working Directory)：即目录里的东西

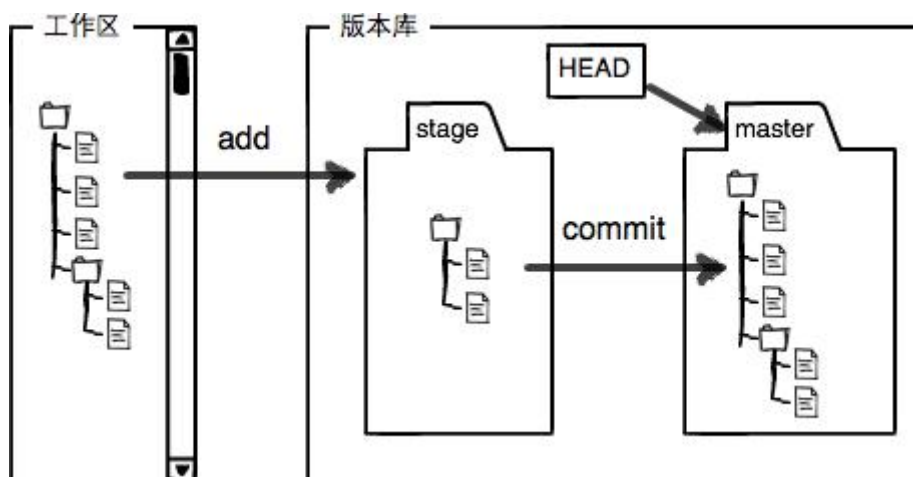
stage (或者叫index) 的暂存区

还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。

首先我们现在Working Directory写好东西

add操作将Working Directory的修改写到stage里

commit操作将stage里所有东西写道master里



## 更多操作

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/3 17:03

更新时间： 2019/3/3 17:12

作者： 刘坤鑫

URL: <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

---

撤销修改：

让这个文件回到最近一次 `git commit` 或 `git add` 时的状态。

```
$ git checkout -- readme.txt
```

用命令 `git reset HEAD <file>` 可以把暂存区的修改撤销掉（unstage），重新放回工作区：

```
$ git reset HEAD readme.txt
```

已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](#)一节，不过前提是没有推送到远程库。

删除文件：如果删除了，可以撤销修改

```
$ git rm test.txt
```

## 远程仓库

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/3 17:13

更新时间： 2019/3/3 17:30

作者： 刘坤鑫

URL: <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

---

第1步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

```
$ ssh-keygen -t rsa -C "youremail@example.com"
```

如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的密钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

第2步：登陆GitHub，打开“Account settings”，“SSH Keys”页面：

然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容：

GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub

如果你不想让别人看到Git库，有两个办法，一个是交点保护费，让GitHub把公开的仓库变成私有的，这样别人就看不见了（不可读更不可写）。另一个办法是自己动手，搭一个Git服务器，因为是你自己的Git服务器，所以别人也是看不见的。这个方法我们后面会讲到的，相当简单，公司内部开发必备。

## 远程库操作

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/3 17:31

更新时间： 2019/3/4 10:36

作者： 刘坤鑫

URL： <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

首先，登陆GitHub，然后，在右上角找到“Create a new repo”按钮，创建一个新的仓库：

在Repository name填入 `learn git`，其他保持默认设置，点击“Create repository”按钮，就成功地创建了一个新的Git仓库：

根据GitHub的提示，在本地的 `learn git` 仓库下运行命令：

```
$ git remote add origin https://github.com/qq734628996/git-learn.git
```

后面的地址还可以用：

```
git@github.com:user-name/repo-name.git
```

使用 `https` 除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令，但是在某些只开放 `http` 端口的公司内部就无法使用 `ssh` 协议而只能用 `https`。

```
$ git push -u origin master
```

加上了 `-u` 参数，Git不但会把本地的 `master` 分支内容推送的远程新的 `master` 分支，还会把本地的 `master` 分支和远程的 `master` 分支关联起来，在以后的推送或者拉取时就可以简化命令。

从现在起，只要本地作了提交，就可以通过命令把本地 `master` 分支的最新修改推送至GitHub：

```
$ git push origin master
```

从远程库克隆：

```
$ git clone git@github.com:user-name/repo-name.git
```

要查看远程库的信息，用 `git remote`：

```
$ git remote
origin
```

用 `git remote -v` 显示更详细的信息：

```
$ git remote -v
origin  git@github.com:michaelliao/learn git.git (fetch)
origin  git@github.com:michaelliao/learn git.git (push)
```

我们可以删除已有的GitHub远程库：

```
git remote rm origin
```

推送分支（分支需要指定）：

```
$ git push origin master
```

哪些分支需要推送，哪些不需要呢？ `master` 分支是主分支，因此要时刻与远程同步； `dev` 分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步； `bug` 分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug； `feature` 分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

抓取分支（默认情况下，你的小伙伴只能看到本地的 `master` 分支。）：

```
$ git clone git@github.com:michaelliao/learn git.git
```

你的小伙伴要在 `dev` 分支上开发，就必须创建远程 `origin` 的 `dev` 分支到本地，你的小伙伴已向 `origin/dev` 分支推送了他的提交，而碰巧你也对同样的文件作了修改，并试图推送：

推送失败，因为你的小伙伴的最新提交和你试图推送的提交有冲突，解决办法也很简单，Git已经提示我们，先用 `git pull` 把最新的提交从 `origin/dev` 抓下来，然后，在本地合并，解决冲突，再推送：

```
$ git pull
```

`git pull`也失败了，原因是没有指定本地`dev`分支与远程`origin/dev`分支的链接，根据提示，设置`dev`和`origin/dev`的链接：

```
$ git branch --set-upstream-to=origin/dev dev
```

再pull，但是合并有冲突，需要手动解决，解决的方法和分支管理中的[解决冲突](#)完全一样。解决后，提交，再push。

**rebase:**

rebase操作可以把本地未push的分叉提交历史整理成直线；

```
$ git rebase
```



## 分支管理

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/3 21:00

更新时间： 2019/3/4 10:36

作者： 刘坤鑫

URL: <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

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。

创建并切换到dev分支：

```
$ git checkout -b dev
```

`git checkout`命令加上`-b`参数表示创建并切换，相当于以下两条命令：

```
$ git branch dev
$ git checkout dev
```

查看当前分支：（当前分支前面会标一个\*号）

```
$ git branch
```

切换分支：

```
$ git checkout master
```

合并分支dev到当前分支：

```
$ git merge dev
```

合并完成后，就可以放心地删除`dev`分支了：

```
$ git branch -d dev
```

解决冲突：

修改文件有冲突时，Git告诉我们，`readme.txt`文件存在冲突，必须手动解决冲突后再提交。`git status`也可以告诉我们冲突的文件：

```
git status
```

用带参数的`git log`也可以看到分支的合并情况：

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

Fast forward合并模式下，只是简单的把`master`指针指向分支，但这种模式下，删除分支后，会丢掉分支信息。如果要强制禁用`Fast forward`模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

准备合并`dev`分支，请注意`--no-ff`参数，表示禁用`Fast forward`：（因为本次合并要创建一个新的commit，所以加上`-m`参数，把commit描述写进去。）

```
$ git merge --no-ff -m "merge with no-ff" dev
```

分支策略：

`master`分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；干活都在`dev`分支上，也就是说，`dev`分支是不稳定的，到某个时候，比如1.0版本发布时，再把`dev`分支合并到`master`上，在`master`分支发布1.0版本；你和你的小伙伴们每个人都在`dev`分支上干活，每个人都有自己的分支，时不时地往`dev`分支上合并就可以了。

Bug分支：

当你接到一个修复一个代号101的bug的任务时，很自然地，你想创建一个分支`issue-101`来修复它，但是，等等，当前正在`dev`上进行的工作还没有提交：

幸好，Git还提供了一个`stash`功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作：

```
$ git stash
```

首先确定要在哪个分支上修复bug，假定需要在`master`分支上修复，就从`master`创建临时分支：

```
$ git checkout master  
$ git checkout -b issue-101
```

接着回到`dev`分支干活了

```
$ git checkout dev
```

刚才的工作现场存到哪去了？用`git stash list`命令看看：

```
$ git stash list
```

工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：

一是用`git stash apply`恢复，但是恢复后，stash内容并不删除，你需要用`git stash drop`来删除；

另一种方式是用`git stash pop`，恢复的同时把stash内容也删了：

```
$ git stash pop
```

你可以多次stash，恢复的时候，先用`git stash list`查看，然后恢复指定的stash，用命令：

```
$ git stash apply stash@{0}
```

**Feature分支：**

添加一个新功能时，你肯定不希望因为一些实验性质的代码，把主分支搞乱了，所以，每添加一个新功能，最好新建一个feature分支，在上面开发，完成后，合并，最后，删除该feature分支。

如果要强行删除，需要使用大写的`-D`参数。。

```
$ git branch -D feature-vulcan
```

## 标签管理

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/4 9:28

更新时间： 2019/3/4 9:34

作者： 刘坤鑫

URL: <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

发布一个版本时，我们通常先在版本库中打一个标签（tag），这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的moment的历史版本取出来。所以，标签也是版本库的一个快照。

首先，切换到需要打标签的分支上：

```
$ git checkout master
```

然后，敲命令`git tag <name>`就可以打一个新标签：

```
$ git tag v1.0
```

可以用命令`git tag`查看所有标签：

```
$ git tag
```

给历史版本打标签：

```
$ git log --pretty=oneline --abbrev-commit  
$ git tag v0.9 f52c633
```

可以用`git show <tagname>`查看标签信息：

```
$ git show v0.9
```

还可以创建带有说明的标签，用`-a`指定标签名，`-m`指定说明文字：

```
$ git tag -a v0.1 -m "version 0.1 released" 1094adb
```

注意：标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签。

如果标签打错了，也可以删除：

```
$ git tag -d v0.1
```

如果要推送某个标签到远程，使用命令`git push origin <tagname>`：

```
$ git push origin v1.0
```

或者，一次性推送全部尚未推送到远程的本地标签：

```
$ git push origin --tags
```

如果标签已经推送到远程，要删除远程标签就麻烦一点，先从本地删除：然后，从远程删除。删除命令也是push，但是格式如下：

```
$ git tag -d v0.9  
$ git push origin :refs/tags/v0.9
```

## 使用GitHub & 码云

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/4 9:35

更新时间： 2019/3/4 9:52

作者： 刘坤鑫

URL: <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

点“Fork”就在自己的账号下克隆了一个bootstrap仓库，然后，从自己的账号下clone：如果你想修复bootstrap的一个bug，或者新增一个功能，立刻就可以开始干活，干完后，往自己的仓库推送。

如果你希望bootstrap的官方库能接受你的修改，你就可以在GitHub上发起一个pull request。当然，对方是否接受你的pull request就不一定了。

如果我们希望体验Git飞一般的速度，可以使用国内的Git托管服务——[码云](https://gitee.com) ([gitee.com](https://gitee.com))。

和GitHub相比，码云也提供免费的Git仓库。此外，还集成了代码质量检测、项目演示等功能。对于团队协作开发，码云还提供了项目管理、代码托管、文档管理的服务，5人以下小团队免费。

- 1.使用码云和使用GitHub类似，我们在码云上注册账号并登录后，需要先上传自己的SSH公钥。选择右上角用户头像 -> 菜单“修改资料”，然后选择“SSH公钥”，填写一个便于识别的标题，然后把用户主目录下的`.ssh/id_rsa.pub`文件的内容粘贴进去：
- 2.如果我们已经有了一个本地的git仓库，我们在码云上创建一个新的项目，选择右上角用户头像 -> 菜单“控制面板”，然后点击“创建项目”：项目名称最好与本地库保持一致：
- 3.然后，我们在本地库上使用命令`git remote add`把它和码云的远程库关联：之后，就可以正常地用`git push`和`git pull`推送了！
- 4.如果本地库已经关联了`origin`的远程库，并且，该远程库指向GitHub。我们可以删除已有的GitHub远程库：先关联GitHub的远程库：再关联码云的远程库：

```
git remote add github git@github.com:user-name/learngit.git
git remote add gitee git@gitee.com:user-name/learngit.git
```

注意：两个远程库的名字不同，不是原来的origin，推送的时候也要注意名字的变化

## 自定义Git

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/4 9:53

更新时间： 2019/3/4 10:37

作者： 刘坤鑫

URL： <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

比如，让Git显示颜色，会让命令输出看起来更醒目：

```
$ git config --global color.ui true
```

忽略特殊文件：

在Git工作区的根目录下创建一个特殊的`.gitignore`文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。不需要从头写`.gitignore`文件，GitHub已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览：<https://github.com/github/gitignore>

忽略文件的原则是：

1. 忽略操作系统自动生成的文件，比如缩略图等；
2. 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的`.class`文件；
3. 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

配置别名：

给命令起个别名：

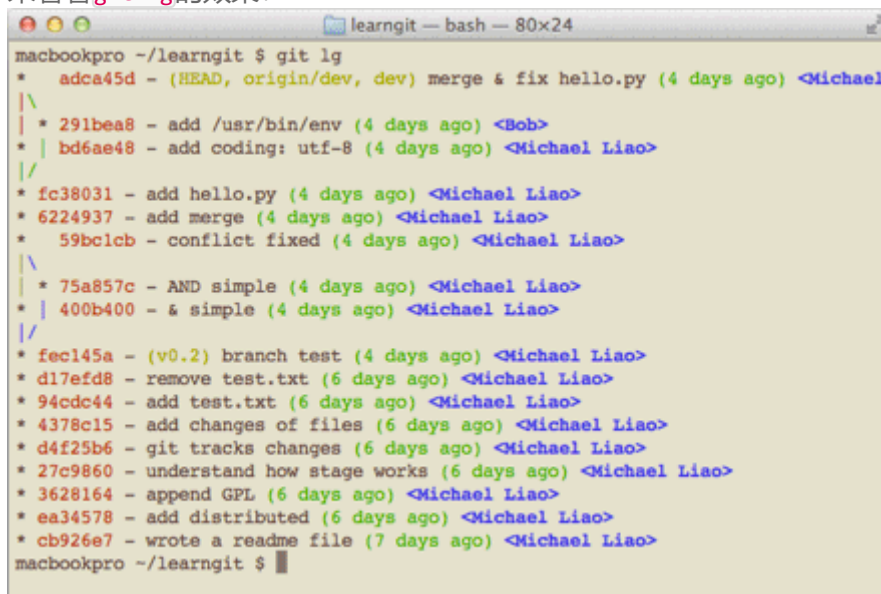
```
$ git config --global alias.st status
$ git config --global alias.co checkout
$ git config --global alias.ci commit
$ git config --global alias.br branch
$ git config --global alias.unstage 'reset HEAD'
$ git config --global alias.last 'log -1'
```

`--global`参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用。

甚至还有人丧心病狂地把`lg`配置成了：

```
git config --global alias.lg "log --color --graph --
pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold
blue)<an>%Creset' --abbrev-commit"
```

来看看`git lg`的效果：



```
macbookpro ~/learnigit $ git lg
* adca45d - (HEAD, origin/dev, dev) merge & fix hello.py (4 days ago) <Michael Liao>
|
| * 291bea8 - add /usr/bin/env (4 days ago) <Bob>
| * bd6ae48 - add coding: utf-8 (4 days ago) <Michael Liao>
|/
* fc38031 - add hello.py (4 days ago) <Michael Liao>
* 6224937 - add merge (4 days ago) <Michael Liao>
* 59bc1cb - conflict fixed (4 days ago) <Michael Liao>
|
| * 75a857c - AND simple (4 days ago) <Michael Liao>
| * 400b400 - & simple (4 days ago) <Michael Liao>
|/
* fec145a - (v0.2) branch test (4 days ago) <Michael Liao>
* d17efd8 - remove test.txt (6 days ago) <Michael Liao>
* 94cdc44 - add test.txt (6 days ago) <Michael Liao>
* 4378c15 - add changes of files (6 days ago) <Michael Liao>
* d4f25b6 - git tracks changes (6 days ago) <Michael Liao>
* 27c9860 - understand how stage works (6 days ago) <Michael Liao>
* 3628164 - append GPL (6 days ago) <Michael Liao>
* ea34578 - add distributed (6 days ago) <Michael Liao>
* cb926e7 - wrote a readme file (7 days ago) <Michael Liao>
macbookpro ~/learnigit $
```

配置文件：

配置文件放哪了？每个仓库的Git配置文件都放在`.git/config`文件中：

```
$ cat .git/config
```

别名就在[alias]后面，要删除别名，直接把对应的行删掉即可。

## 搭建Git服务器

笔记本： 廖雪峰 - git教程

创建时间： 2019/3/4 10:18

更新时间： 2019/3/4 10:25

作者： 刘坤鑫

URL： <https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c6...>

搭建Git服务器需要准备一台运行Linux的机器，强烈推荐用Ubuntu或Debian，这样，通过几条简单的`apt`命令就可以完成安装。

第一步，安装`git`：

```
$ sudo apt-get install git
```

第二步，创建一个`git`用户，用来运行`git`服务：

```
$ sudo adduser git
```

第三步，创建证书登录：

收集所有需要登录的用户的公钥，就是他们自己的`id_rsa.pub`文件，把所有公钥导入到`/home/git/.ssh/authorized_keys`文件里，一行一个。

第四步，初始化Git仓库：

先选定一个目录作为Git仓库，假定是`/srv/sample.git`，在`/srv`目录下输入命令：

```
$ sudo git init --bare sample.git
```

Git就会创建一个裸仓库，裸仓库没有工作区，因为服务器上的Git仓库纯粹是为了共享，所以不让用户直接登录到服务器上去改工作区，并且服务器上的Git仓库通常都以`.git`结尾。然后，把owner改为`git`：

```
$ sudo chown -R git:git sample.git
```

第五步，禁用shell登录：

出于安全考虑，第二步创建的git用户不允许登录shell，这可以通过编辑`/etc/passwd`文件完成。找到类似下面的一行：

```
git:x:1001:1001:,,,:/home/git:/bin/bash
```

改为：

```
git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell
```

这样，`git`用户可以正常通过ssh使用git，但无法登录shell，因为我们为`git`用户指定的`git-shell`每次一登录就自动退出。

第六步，克隆远程仓库：

现在，可以通过`git clone`命令克隆远程仓库了，在各自的电脑上运行：

```
$ git clone git@server:/srv/sample.git
```

## 管理公钥

如果团队很小，把每个人的公钥收集起来放到服务器的`/home/git/.ssh/authorized_keys`文件里就是可行的。如果团队有几百号人，就没法这么玩了，这时，可以用[Gitosis](#)来管理公钥。

这里我们不介绍怎么玩[Gitosis](#)了，几百号人的团队基本都在500强了，相信找个高水平的Linux管理员问题不大。

## 管理权限

有很多不但视源代码如生命，而且视员工为窃贼的公司，会在版本控制系统里设置一套完善的权限控制，每个人是否有读写权限会精确到每个分支甚至每个目录下。因为Git是为Linux源代码托管而开发的，所以Git也继承了开源社区的精神，不支持权限控制。不

过，因为Git支持钩子（hook），所以，可以在服务器端编写一系列脚本来控制提交等操作，达到权限控制的目的。[Gitolite](#)就是这个工具。

这里我们也不介绍[Gitolite](#)了，不要把有限的生命浪费到权限斗争中。