**[Miao.wang.stat@gmail.com](mailto:Miao.wang.stat@gmail.com)**

**5709wanggit**

**git init:** Initializes a new Git repository. Until you run this command inside a repository or directory, it’s just a regular folder. Only after you input this does it accept further Git commands.

**git config:** Short for “configure,” this is most useful when you’re setting up Git for the first time.

**git help:** Forgot a command? Type this into the command line to bring up the 21 most common git commands. You can also be more specific and type “git help init” or another term to figure out how to use and configure a specific git command.

**git status:** Check the status of your repository. See which files are inside it, which changes still need to be committed, and which branch of the repository you’re currently working on.

**git add:** This does not add new files to your repository. Instead, it brings new files to Git’s attention. After you add files, they’re included in Git’s “snapshots” of the repository.

**git commit:** Git’s most important command. After you make any sort of change, you input this in order to take a “snapshot” of the repository. Usually it goes git commit -m “Message here.”The -mindicates that the following section of the command should be read as a message.

**git branch:** Working with multiple collaborators and want to make changes on your own? This command will let you build a new branch, or timeline of commits, of changes and file additions that are completely your own. Your title goes after the command. If you wanted a new branch called “cats,” you’d type git branch cats.

**git checkout:** Literally allows you to “check out” a repository that you are not currently inside. This is a navigational command that lets you move to the repository you want to check. You can use this command as git checkout masterto look at the master branch, or git checkout catsto look at another branch.

**git merge:** When you’re done working on a branch, you can merge your changes back to the master branch, which is visible to all collaborators. git merge catswould take all the changes you made to the “cats” branch and add them to the master.

**git push:** If you’re working on your local computer, and want your commits to be visible online on GitHub as well, you “push” the changes up to GitHub with this command.

**git pull:** If you’re working on your local computer and want the most up-to-date version of your repository to work with, you “pull” the changes down from GitHub with this command.

1.要检查已有的配置信息，可以使用 git config --list 命令：

2.也可以直接查阅某个环境变量的设定 git config user.name

## 3. 获取帮助

想了解 Git 的各式工具该怎么用，可以阅读它们的使用帮助，方法有三：

$ git help <verb>

$ git <verb> --help

$ man git-<verb>

比如，要学习 config 命令可以怎么用，运行：

$ git help config

### 4. [从现有仓库克隆](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E5%8F%96%E5%BE%97%E9%A1%B9%E7%9B%AE%E7%9A%84-Git-%E4%BB%93%E5%BA%93#从现有仓库克隆)

 git clone [url]

在克隆的时候，自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：

$ git clone git://github.com/schacon/oldproject.git myprojectname

使用的是 git:// 协议，不过你也可以用 http(s):// 或者user@server:/path.git 表示的 SSH 传输协议

### 5. [在工作目录中初始化新仓库](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E5%8F%96%E5%BE%97%E9%A1%B9%E7%9B%AE%E7%9A%84-Git-%E4%BB%93%E5%BA%93#在工作目录中初始化新仓库)

到此项目所在的目录，执行：

$ git init

初始化后，在当前目录下会出现一个名为 .git 的目录, 但我们还没有开始跟踪管理项目中的任何一个文件. 如果当前目录下有几个文件想要纳入版本控制，需要先用 git add 命令告诉 Git 开始对这些文件进行跟踪，然后提交.

$ git add \*.c

$ git add README

$ git commit -m 'initial project version'

### [检查当前文件状态](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#检查当前文件状态)

要确定哪些文件当前处于什么状态，可以用 git status 命令

在状态报告中可以看到新建文件出现在“Untracked files”下面。未跟踪的文件意味着Git在之前的快照（提交）中没有这些文件；Git 不会自动将之纳入跟踪范围

### [跟踪新文件](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#跟踪新文件)

使用命令 git add 开始跟踪一个新文件。所以，要跟踪 README 文件，运行：

$ git add README

只要在 “Changes to be committed” 这行下面的，就说明是已暂存状态。如果此时提交，那么该文件此时此刻的版本将被留存在历史记录中

出现在 “Changes not staged for commit” 这行下面，说明已跟踪文件的内容发生了变化，但还没有放到暂存区。要暂存这次更新，需要运行 git add 命令 (again)

### [忽略某些文件](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#忽略某些文件)

我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式。

我们再看一个 .gitignore 文件的例子：

# 此为注释 – 将被 Git 忽略

# 忽略所有 .a 结尾的文件

\*.a

# 但 lib.a 除外

!lib.a

# 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO

/TODO

# 忽略 build/ 目录下的所有文件

build/

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

doc/\*.txt

# ignore all .txt files in the doc/ directory

doc/\*\*/\*.txt

### [查看已暂存和未暂存的更新](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#查看已暂存和未暂存的更新)

如果要查看具体修改了什么地方，可以用 git diff 命令. 工作目录中当前文件和暂存区域快照之间的差异，也就是修改之后还没有暂存起来的变化内容。

若要看已经暂存起来的文件和上次提交时的快照之间的差异，可以用 git diff --cached 命令

### [提交更新](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#提交更新)

现在的暂存区域已经准备妥当可以提交了。在此之前，请一定要确认还有什么修改过的或新建的文件还没有 git add 过，否则提交的时候不会记录这些还没暂存起来的变化。所以，每次准备提交前，先用 git status 看下，是不是都已暂存起来了，然后再运行提交命令 git commit：

$ git commit

这种方式会启动文本编辑器以便输入本次提交的说明

另外也可以用 -m 参数后跟提交说明的方式，在一行命令中提交更新：

### [跳过使用暂存区域](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#跳过使用暂存区域)

尽管使用暂存区域的方式可以精心准备要提交的细节，但有时候这么做略显繁琐。Git 提供了一个跳过使用暂存区域的方式，只要在提交的时候，给 git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤：

### [移除文件](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#移除文件)

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交。可以用 git rm 命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单中了。

如果只是简单地从工作目录中手工删除文件，运行 git status 时就会在 “Changes not staged for commit” 部分（也就是未暂存清单）

### [移动文件](http://git-scm.com/book/zh/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#移动文件)

不像其他的 VCS 系统，Git 并不跟踪文件移动操作。

要在 Git 中对文件改名，可以这么做：

$ git mv file\_from file\_to

运行 git mv 就相当于运行了下面三条命令：

$ mv README.txt README

$ git rm README.txt

$ git add README

## 查看提交历史

在提交了若干更新之后，又或者克隆了某个项目，想回顾下提交历史，可以使用 git log 命令查看。

Exit log: q

我们常用 -p 选项展开显示每次提交的内容差异，用 -2 则仅显示最近的两次更新：

$ git log -p -2

单词层面的对比

$ git log -U1 --word-diff

git log 还提供了许多摘要选项可以用，比如 --stat，仅显示简要的增改行数统计

但最有意思的是 format，可以定制要显示的记录格式，这样的输出便于后期编程提取分析，像这样：

$ git log --pretty=format:"%h - %an, %ar : %s"

### [修改最后一次提交](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%92%A4%E6%B6%88%E6%93%8D%E4%BD%9C#修改最后一次提交)

有时候我们提交完了才发现漏掉了几个文件没有加，或者提交信息写错了。想要撤消刚才的提交操作，可以使用 --amend 选项重新提交：

$ git commit --amend

此命令将使用当前的暂存区域快照提交。如果刚才提交完没有作任何改动，直接运行此命令的话，相当于有机会重新编辑提交说明，但将要提交的文件快照和之前的一样。

如果刚才提交时忘了暂存某些修改，可以先补上暂存操作，然后再运行 --amend 提交：

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

上面的三条命令最终只是产生一个提交，第二个提交命令修正了第一个的提交内容。

### [取消已经暂存的文件](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%92%A4%E6%B6%88%E6%93%8D%E4%BD%9C#取消已经暂存的文件)

可以使用 git reset HEAD <file>... 的方式, 文件又回到了之前已修改未暂存的状态。

### [取消对文件的修改](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%92%A4%E6%B6%88%E6%93%8D%E4%BD%9C#取消对文件的修改)

$ git checkout –myfileBacktoLastVersion

这条命令有些危险，所有对文件的修改都没有了

### [查看当前的远程库](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E8%BF%9C%E7%A8%8B%E4%BB%93%E5%BA%93%E7%9A%84%E4%BD%BF%E7%94%A8#查看当前的远程库)

要查看当前配置有哪些远程仓库，可以用 git remote 命令，它会列出每个远程库的简短名字。

至少可以看到一个名为 origin 的远程库，Git 默认使用这个名字来标识你所克隆的原始仓库

### [添加远程仓库](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E8%BF%9C%E7%A8%8B%E4%BB%93%E5%BA%93%E7%9A%84%E4%BD%BF%E7%94%A8#添加远程仓库)

要添加一个新的远程仓库，可以指定一个简单的名字，以便将来引用，运行 git remote add [shortname] [url] (eg: my1 https://github.com/miaowangstat/hello.git)

[从远程仓库抓取数据](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E8%BF%9C%E7%A8%8B%E4%BB%93%E5%BA%93%E7%9A%84%E4%BD%BF%E7%94%A8#从远程仓库抓取数据)

正如之前所看到的，可以用下面的命令从远程仓库抓取数据到本地：

$ git fetch [remote-name]

fetch 命令只是将远端的数据拉到本地仓库，并不自动合并到当前工作分支，只有当你确实准备好了，才能手工合并。此命令会到远程仓库中拉取所有你本地仓库中还没有的数据。运行完成后，你就可以在本地访问该远程仓库中的所有分支，将其中某个分支合并到本地，或者只是取出某个分支

### [推送数据到远程仓库](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E8%BF%9C%E7%A8%8B%E4%BB%93%E5%BA%93%E7%9A%84%E4%BD%BF%E7%94%A8#推送数据到远程仓库)

项目进行到一个阶段，要同别人分享目前的成果，可以将本地仓库中的数据推送到远程仓库。实现这个任务的命令很简单： git push [remote-name] [branch-name]。如果要把本地的 master 分支推送到origin 服务器上（再次说明下，克隆操作会自动使用默认的 master 和 origin 名字），可以运行下面的命令：

$ git push origin master

### [查看远程仓库信息](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E8%BF%9C%E7%A8%8B%E4%BB%93%E5%BA%93%E7%9A%84%E4%BD%BF%E7%94%A8#查看远程仓库信息)

我们可以通过命令 git remote show [remote-name] 查看某个远程仓库的详细信息，比如要看所克隆的 origin 仓库，可以运行：

$ git remote show origin

### [远程仓库的删除和重命名](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E8%BF%9C%E7%A8%8B%E4%BB%93%E5%BA%93%E7%9A%84%E4%BD%BF%E7%94%A8#远程仓库的删除和重命名)

在新版 Git 中可以用 git remote rename 命令修改某个远程仓库在本地的简称，比如想把 pb 改成paul，可以这么运行：

$ git remote rename pb paul

需要移除对应的远端仓库，可以运行 git remote rm 命令：

$ git remote rm paul

### [列显已有的标签](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#列显已有的标签)

列出现有标签的命令非常简单，直接运行 git tag 即可：

### [含附注的标签](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#含附注的标签)

创建一个含附注类型的标签非常简单，用 -a （译注：取 annotated 的首字母）指定标签名字即可：

$ git tag -a v1.4 -m 'my version 1.4'

而 -m 选项则指定了对应的标签说明，Git 会将此说明一同保存在标签对象中

### [签署标签](http://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#签署标签)

如果你有自己的私钥，还可以用 GPG 来签署标签，只需要把之前的 -a 改为 -s （译注： 取 signed 的首字母）即可：

$ git tag -s v1.5 -m 'my signed 1.5 tag'

**新建**一个 testing 分支，可以使用 git branch 命令：

$ git branch testing

这会在当前 commit 对象上新建一个分支指针

**转换**到新建的 testing 分支：

$ git checkout testing

这样 HEAD 就指向了 testing 分支

回到 master 分支并把它**合并**进来，然后发布到生产服务器。用 git merge 命令来进行合并：

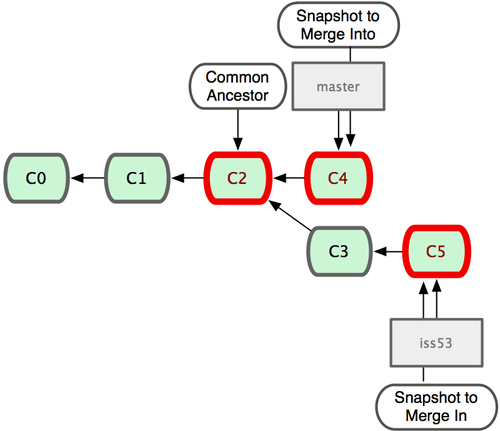
$ git checkout master

$ git merge hotfix

由于当前 hotfix 分支和 master 都指向相同的提交对象，所以 hotfix 已经完成了历史使命，可以删掉了。使用 git branch 的 -d 选项**执行删除操作**：

$ git branch -d hotfix

这次你的开发历史是从更早的地方开始分叉的。由于当前 master 分支所指向的提交对象（C4）并不是 iss53 分支的直接祖先，Git 不得不进行一些额外处理。就此例而言，Git 会用两个分支的末端（C4 和 C5）以及它们的共同祖先（C2）进行一次简单的三方合并计算。图 3-16 用红框标出了 Git 用于合并的三个提交对象



Git 没有简单地把分支指针右移，而是对三方合并后的结果重新做一个新的快照，并自动创建一个指向它的提交对象（C6）

可以用 git branch --no-merged **查看尚未合并的工作**：

$ git branch --no-merged

testing

由于这些分支中还包含着尚未合并进来的工作成果，所以简单地用 git branch -d 删除该分支会提示错误，因为那样做会丢失数据：

如果你确实想要删除该分支上的改动，可以用大写的删除选项 -D **强制执行**，就像上面提示信息中给出的那样。

可以运行 git fetch origin 来**同步远程服务器**上的数据到本地。该命令首先找到 origin 是哪个服务器（本例为 git.ourcompany.com），从上面获取你尚未拥有的数据，更新你本地的数据库，然后把origin/master 的指针移到它最新的位置上

### [推送本地分支](http://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF#推送本地分支)

你需要把它推送到一个你拥有写权限的远程仓库。你创建的本地分支不会因为你的写入操作而被自动同步到你引入的远程服务器上，你需要明确地执行推送分支的操作。

可以运行 git push (远程仓库名) (分支名)：

$ git push origin serverfix

也可以运行 git push origin serverfix:serverfix 来实现相同的效果，它的意思是“上传我本地的 serverfix 分支到远程仓库中去，仍旧称它为 serverfix 分支”。通过此语法，你可以把本地分支推送到某个命名不同的远程分支：若想把远程分支叫作 awesomebranch，可以用 git push origin serverfix:awesomebranch 来推送数据。

当你的协作者再次从服务器上获取数据时，他们将得到一个新的远程分支 origin/serverfix，并指向服务器上 serverfix 所指向的版本：

$ git fetch origin

值得注意的是，在 fetch 操作下载好新的远程分支之后，你仍然无法在本地编辑该远程仓库中的分支。换句话说，在本例中，你不会有一个新的 serverfix 分支，有的只是一个你无法移动的origin/serverfix 指针。

如果要把该远程分支的内容合并到当前分支，可以运行 git merge origin/serverfix。如果想要一份自己的 serverfix 来开发，可以在远程分支的基础上分化出一个新的分支来：

$ git checkout -b serverfix origin/serverfix

这会切换到新建的 serverfix 本地分支，其内容同远程分支 origin/serverfix 一致，这样你就可以在里面继续开发了。

### [跟踪远程分支](http://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF#跟踪远程分支)

从远程分支 checkout 出来的本地分支，称为 跟踪分支 (tracking branch)。跟踪分支是一种和某个远程分支有直接联系的本地分支。在跟踪分支里输入 git push，Git 会自行推断应该向哪个服务器的哪个分支推送数据。同样，在这些分支里运行 git pull 会获取所有远程索引，并把它们的数据都合并到本地分支中来。

### [删除远程分支](http://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF#删除远程分支)

可以用这个非常无厘头的语法来删除它：git push [远程名] :[分支名]。如果想在服务器上删除 serverfix 分支，运行下面的命令：

$ git push origin :serverfix

有种方便记忆这条命令的方法：记住我们不久前见过的 git push [远程名] [本地分支]:[远程分支] 语法，如果省略 [本地分支]，那就等于是在说“在这里提取空白然后把它变成[远程分支]”。

**分支的衍合**

把一个分支中的修改整合到另一个分支的办法有两种：merge 和 rebase（译注：rebase 的翻译暂定为“衍合”，大家知道就可以了。）

最容易的整合分支的方法是 merge 命令，它会把两个分支最新的快照（C3 和 C4）以及二者最新的共同祖先（C2）进行三方合并，合并的结果是产生一个新的提交对象（C5）

### [基本的衍合操作](http://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E5%88%86%E6%94%AF%E7%9A%84%E8%A1%8D%E5%90%88#基本的衍合操作)

你可以把在 C3 里产生的变化补丁在 C4 的基础上重新打一遍。在 Git 里，这种操作叫做衍合（rebase）。有了 rebase 命令，就可以把在一个分支里提交的改变移到另一个分支里重放一遍。

在上面这个例子中，运行：

$ git checkout experiment

$ git rebase master

它的原理是回到两个分支最近的共同祖先，根据当前分支（也就是要进行衍合的分支 experiment）后续的历次提交对象（这里只有一个 C3），生成一系列文件补丁，然后以基底分支（也就是主干分支master）最后一个提交对象（C4）为新的出发点，逐个应用之前准备好的补丁文件，最后会生成一个新的合并提交对象（C3'），从而改写 experiment 的提交历史，使它成为 master 分支的直接下游.

请注意，合并结果中最后一次提交所指向的快照，无论是通过衍合，还是三方合并，都会得到相同的快照内容，只不过提交历史不同罢了。衍合是按照每行的修改次序重演一遍修改，而合并是把最终结果合在一起。

我们就可以把基于 client 分支而非 server 分支的改变（即 C8 和 C9），**跳过 server 直接放**到 master 分支中重演一遍，但这需要用 git rebase 的 --onto 选项指定新的基底分支 master：

$ git rebase --onto master server client

这好比在说：“取出 client 分支，找出 client 分支和 server 分支的共同祖先之后的变化，然后把它们在 master 上重演一遍”。

**快进 master 分支**了（见图 3-33）：

$ git checkout master

$ git **merge** client

可以直接把 server **分支衍合**到 master，而不用手工切换到 server 分支后再执行衍合操作 — git rebase [主分支] [特性分支] 命令会先取出特性分支 server，然后在主分支 master 上重演：

$ git rebase master server

然后就可以快进主干分支 master 了：

$ git checkout master

$ git merge server

### [衍合的风险](http://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E5%88%86%E6%94%AF%E7%9A%84%E8%A1%8D%E5%90%88#衍合的风险)

呃，奇妙的衍合也并非完美无缺，要用它得遵守一条准则：

**一旦分支中的提交对象发布到公共仓库，就千万不要对该分支进行衍合操作。**

# 服务器上的 Git - 协议

## 协议

Git 可以使用四种主要的协议来传输数据：本地传输，SSH 协议，Git 协议和 HTTP 协议。

### [本地协议](http://git-scm.com/book/zh/v1/%E6%9C%8D%E5%8A%A1%E5%99%A8%E4%B8%8A%E7%9A%84-Git-%E5%8D%8F%E8%AE%AE#本地协议)

在一个本地文件系统中克隆仓库，推送和获取。克隆的时候只需要将远程仓库的路径作为 URL 使用，比如下面这样：

$ git clone /opt/git/project.git

或者这样：

$ git clone file:///opt/git/project.git

要添加一个本地仓库作为现有 Git 项目的远程仓库，可以这样做：

$ git remote add local\_proj /opt/git/project.git

### [SSH 协议](http://git-scm.com/book/zh/v1/%E6%9C%8D%E5%8A%A1%E5%99%A8%E4%B8%8A%E7%9A%84-Git-%E5%8D%8F%E8%AE%AE#SSH-协议)

SSH 也是唯一一个同时支持读写操作的网络协议。另外两个网络协议（HTTP 和 Git）通常都是只读的，所以虽然二者对大多数人都可用，但执行写操作时还是需要 SSH。SSH 同时也是一个验证授权的网络协议；而因为其普遍性，一般架设和使用都很容易。

通过 SSH 克隆一个 Git 仓库，你可以像下面这样给出 ssh:// 的 URL：

$ git clone ssh://user@server/project.git

或者不指明某个协议 — 这时 Git 会默认使用 SSH ：

$ git clone user@server:project.git

### [Git 协议](http://git-scm.com/book/zh/v1/%E6%9C%8D%E5%8A%A1%E5%99%A8%E4%B8%8A%E7%9A%84-Git-%E5%8D%8F%E8%AE%AE#Git-协议)

打算支持 Git 协议的仓库，需要先创建 git-daemon-export-ok文件.

### [HTTP/S 协议](http://git-scm.com/book/zh/v1/%E6%9C%8D%E5%8A%A1%E5%99%A8%E4%B8%8A%E7%9A%84-Git-%E5%8D%8F%E8%AE%AE#HTTP/S-协议)

只需要把 Git 的裸仓库文件放在 HTTP 的根目录下，配置一个特定的 post-update 挂钩（hook）就可以搞定（Git 挂钩的细节见第 7 章）。此后，每个能访问 Git 仓库所在服务器上 web 服务的人都可以进行克隆操作。下面的操作可以允许通过 HTTP 对仓库进行读取：

$ cd /var/www/htdocs/

$ git clone --bare /path/to/git\_project gitproject.git

$ cd gitproject.git

$ mv hooks/post-update.sample hooks/post-update

$ chmod a+x hooks/post-update

这样就可以了。Git 附带的 post-update 挂钩会默认运行合适的命令（git update-server-info）来确保通过 HTTP 的获取和克隆正常工作。这条命令在你用 SSH 向仓库推送内容时运行；之后，其他人就可以用下面的命令来克隆仓库：

$ git clone http://example.com/gitproject.git

在本例中，我们使用了 Apache 设定中常用的 /var/www/htdocs 路径，不过你可以使用任何静态 web 服务 — 把裸仓库放在它的目录里就行。