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。下面分别介绍一下哪些情形应该使用（或避免使用）这些协议。

值得注意的是，除了 HTTP 协议外，其他所有协议都要求在服务器端安装并运行 Git。

本地协议

最基本的就是*本地协议（Local protocol）*，所谓的远程仓库在该协议中的表示，就是硬盘上的另一个目录。这常见于团队每一个成员都对一个共享的文件系统（例如 NFS）拥有访问权，或者比较少见的多人共用同一台电脑的情况。后面一种情况并不安全，因为所有代码仓库实例都储存在同一台电脑里，增加了灾难性数据损失的可能性。

如果你使用一个共享的文件系统，就可以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

如果在 URL 开头明确使用 file:// ，那么 Git 会以一种略微不同的方式运行。如果你只给出路径，Git 会尝试使用硬链接或直接复制它所需要的文件。如果使用了 file:// ，Git 会调用它平时通过网络来传输数据的工序，而这种方式的效率相对较低。使用 file:// 前缀的主要原因是当你需要一个不包含无关引用或对象的干净仓库副本的时候 — 一般指从其他版本控制系统导入的，或类似情形（参见第 9 章的维护任务）。我们这里仅仅使用普通路径，这样更快。

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

然后就可以像在网络上一样向这个远程仓库推送和获取数据了。

优点

基于文件仓库的优点在于它的简单，同时保留了现存文件的权限和网络访问权限。如果你的团队已经有一个全体共享的文件系统，建立仓库就十分容易了。你只需把一份裸仓库的副本放在大家都能访问的地方，然后像对其他共享目录一样设置读写权限就可以了。我们将在下一节“在服务器上部署 Git ”中讨论如何导出一个裸仓库的副本。

这也是从别人工作目录中获取工作成果的快捷方法。假如你和你的同事在一个项目中合作，他们想让你检出一些东西的时候，运行类似 git pull /home/john/project 通常会比他们推送到服务器，而你再从服务器获取简单得多。

缺点

这种方法的缺点是，与基本的网络连接访问相比，难以控制从不同位置来的访问权限。如果你想从家里的笔记本电脑上推送，就要先挂载远程硬盘，这和基于网络连接的访问相比更加困难和缓慢。

另一个很重要的问题是该方法不一定就是最快的，尤其是对于共享挂载的文件系统。本地仓库只有在你对数据访问速度快的时候才快。在同一个服务器上，如果二者同时允许 Git 访问本地硬盘，通过 NFS 访问仓库通常会比 SSH 慢。

SSH 协议

Git 使用的传输协议中最常见的可能就是 SSH 了。这是因为大多数环境已经支持通过 SSH 对服务器的访问 — 即便还没有，架设起来也很容易。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

如果不指明用户，Git 会默认使用当前登录的用户名连接服务器。

优点

使用 SSH 的好处有很多。首先，如果你想拥有对网络仓库的写权限，基本上不可能不使用 SSH。其次，SSH 架设相对比较简单 — SSH 守护进程很常见，很多网络管理员都有一些使用经验，而且很多操作系统都自带了它或者相关的管理工具。再次，通过 SSH 进行访问是安全的 — 所有数据传输都是加密和授权的。最后，和 Git 及本地协议一样，SSH 也很高效，会在传输之前尽可能压缩数据。

缺点

SSH 的限制在于你不能通过它实现仓库的匿名访问。即使仅为读取数据，人们也必须在能通过 SSH 访问主机的前提下才能访问仓库，这使得 SSH 不利于开源的项目。如果你仅仅在公司网络里使用，SSH 可能是你唯一需要使用的协议。如果想允许对项目的匿名只读访问，那么除了为自己推送而架设 SSH 协议之外，还需要支持其他协议以便他人访问读取。

Git 协议

接下来是 Git 协议。这是一个包含在 Git 软件包中的特殊守护进程； 它会监听一个提供类似于 SSH 服务的特定端口（9418），而无需任何授权。打算支持 Git 协议的仓库，需要先创建 git-daemon-export-ok 文件 — 它是协议进程提供仓库服务的必要条件 — 但除此之外该服务没有什么安全措施。要么所有人都能克隆 Git 仓库，要么谁也不能。这也意味着该协议通常不能用来进行推送。你可以允许推送操作；然而由于没有授权机制，一旦允许该操作，网络上任何一个知道项目 URL 的人将都有推送权限。不用说，这是十分罕见的情况。

优点

Git 协议是现存最快的传输协议。如果你在提供一个有很大访问量的公共项目，或者一个不需要对读操作进行授权的庞大项目，架设一个 Git 守护进程来供应仓库是个不错的选择。它使用与 SSH 协议相同的数据传输机制，但省去了加密和授权的开销。

缺点

Git 协议消极的一面是缺少授权机制。用 Git 协议作为访问项目的唯一方法通常是不可取的。一般的做法是，同时提供 SSH 接口，让几个开发者拥有推送（写）权限，其他人通过 git:// 拥有只读权限。 Git 协议可能也是最难架设的协议。它要求有单独的守护进程，需要定制 — 我们将在本章的 “Gitosis” 一节详细介绍它的架设 — 需要设定xinetd 或类似的程序，而这些工作就没那么轻松了。该协议还要求防火墙开放 9418 端口，而企业级防火墙一般不允许对这个非标准端口的访问。大型企业级防火墙通常会封锁这个少见的端口。

HTTP/S 协议

最后还有 HTTP 协议。HTTP 或 HTTPS 协议的优美之处在于架设的简便性。基本上，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。 Git 的数据是以最基本的静态文件的形式提供的（关于如何提供文件的详情见第 9 章）。

通过 HTTP 进行推送操作也是可能的，不过这种做法不太常见，并且牵扯到复杂的 WebDAV 设定。由于很少用到，本书将略过对该内容的讨论。如果对 HTTP 推送协议感兴趣，不妨打开这个地址看一下操作方法：http://www.kernel.org/pub/software/scm/git/docs/howto/setup-git-server-over-http.txt 。通过 HTTP 推送的好处之一是你可以使用任何 WebDAV 服务器，不需要为 Git 设定特殊环境；所以如果主机提供商支持通过 WebDAV 更新网站内容，你也可以使用这项功能。

优点

使用 HTTP 协议的好处是易于架设。几条必要的命令就可以让全世界读取到仓库的内容。花费不过几分钟。HTTP 协议不会占用过多服务器资源。因为它一般只用到静态的 HTTP 服务提供所有数据，普通的 Apache 服务器平均每秒能支撑数千个文件的并发访问 — 哪怕让一个小型服务器超载都很难。

你也可以通过 HTTPS 提供只读的仓库，这意味着你可以加密传输内容；你甚至可以要求客户端使用特定签名的 SSL 证书。一般情况下，如果到了这一步，使用 SSH 公共密钥可能是更简单的方案；不过也存在一些特殊情况，这时通过 HTTPS 使用带签名的 SSL 证书或者其他基于 HTTP 的只读连接授权方式是更好的解决方案。

HTTP 还有个额外的好处：HTTP 是一个如此常见的协议，以至于企业级防火墙通常都允许其端口的通信。

缺点

HTTP 协议的消极面在于，相对来说客户端效率更低。克隆或者下载仓库内容可能会花费更多时间，而且 HTTP 传输的体积和网络开销比其他任何一个协议都大。因为它没有按需供应的能力 — 传输过程中没有服务端的动态计算 — 因而 HTTP 协议经常会被称为*傻瓜（dumb）*协议。更多 HTTP 协议和其他协议效率上的差异见第 9 章。