传输协议

Git 可以以两种主要的方式跨越两个仓库传输数据：基于HTTP协议之上，和 file://, ssh://, 和 git:// 等智能传输协议。这一节带你快速浏览这两种主要的协议操作过程。

哑协议

Git 基于HTTP之上传输通常被称为哑协议，这是因为它在服务端不需要有针对 Git 特有的代码。这个获取过程仅仅是一系列GET请求，客户端可以假定服务端的Git仓库中的布局。让我们以 simplegit 库来看看 http-fetch 的过程：

$ git clone http://github.com/schacon/simplegit-progit.git

它做的第1件事情就是获取 info/refs 文件。这个文件是在服务端运行了 update-server-info 所生成的，这也解释了为什么在服务端要想使用HTTP传输，必须要开启 post-receive 钩子：

=> GET info/refs

ca82a6dff817ec66f44342007202690a93763949 refs/heads/master

现在你有一个远端引用和SHA值的列表。下一步是寻找HEAD引用，这样你就知道了在完成后，什么应该被检出到工作目录：

=> GET HEAD

ref: refs/heads/master

这说明在完成获取后，需要检出 master 分支。 这时，已经可以开始漫游操作了。因为你的起点是在 info/refs 文件中所提到的 ca82a6 commit 对象，你的开始操作就是获取它：

=> GET objects/ca/82a6dff817ec66f44342007202690a93763949

(179 bytes of binary data)

然后你取回了这个对象 － 这在服务端是一个松散格式的对象，你使用的是静态的 HTTP GET 请求获取的。可以使用 zlib 解压缩它，去除其头部，查看它的 commmit 内容：

$ git cat-file -p ca82a6dff817ec66f44342007202690a93763949

tree cfda3bf379e4f8dba8717dee55aab78aef7f4daf

parent 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

author Scott Chacon <schacon@gmail.com> 1205815931 -0700

committer Scott Chacon <schacon@gmail.com> 1240030591 -0700

changed the version number

这样，就得到了两个需要进一步获取的对象 － cfda3b 是这个 commit 对象所对应的 tree 对象，和 085bb3 是它的父对象；

=> GET objects/08/5bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

(179 bytes of data)

这样就取得了这它的下一步 commit 对象，再抓取 tree 对象：

=> GET objects/cf/da3bf379e4f8dba8717dee55aab78aef7f4daf

(404 - Not Found)

Oops - 看起来这个 tree 对象在服务端并不以松散格式对象存在，所以得到了404响应，代表在HTTP服务端没有找到该对象。这有好几个原因 － 这个对象可能在替代仓库里面，或者在打包文件里面， Git 会首先检查任何列出的替代仓库：

=> GET objects/info/http-alternates

(empty file)

如果这返回了几个替代仓库列表，那么它会去那些地方检查松散格式对象和文件 － 这是一种在软件分叉之间共享对象以节省磁盘的好方法。然而，在这个例子中，没有替代仓库。所以你所需要的对象肯定在某个打包文件中。要检查服务端有哪些打包格式文件，你需要获取 objects/info/packs 文件，这里面包含有打包文件列表（是的，它也是被 update-server-info 所生成的）；

=> GET objects/info/packs

P pack-816a9b2334da9953e530f27bcac22082a9f5b835.pack

这里服务端只有一个打包文件，所以你要的对象显然就在里面。但是你可以先检查它的索引文件以确认。这在服务端有多个打包文件时也很有用，因为这样就可以先检查你所需要的对象空间是在哪一个打包文件里面了：

=> GET objects/pack/pack-816a9b2334da9953e530f27bcac22082a9f5b835.idx

(4k of binary data)

现在你有了这个打包文件的索引，你可以看看你要的对象是否在里面 － 因为索引文件列出了这个打包文件所包含的所有对象的SHA值，和该对象存在于打包文件中的偏移量，所以你只需要简单地获取整个打包文件：

=> GET objects/pack/pack-816a9b2334da9953e530f27bcac22082a9f5b835.pack

(13k of binary data)

现在你也有了这个 tree 对象，你可以继续在 commit 对象上漫游。它们全部都在这个你已经下载到的打包文件里面，所以你不用继续向服务端请求更多下载了。 在这完成之后，由于下载开始时已探明HEAD引用是指向 master分支， Git 会将它检出到工作目录。

整个过程看起来就像这样：

$ git clone http://github.com/schacon/simplegit-progit.git

Initialized empty Git repository in /private/tmp/simplegit-progit/.git/

got ca82a6dff817ec66f44342007202690a93763949

walk ca82a6dff817ec66f44342007202690a93763949

got 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

Getting alternates list for http://github.com/schacon/simplegit-progit.git

Getting pack list for http://github.com/schacon/simplegit-progit.git

Getting index for pack 816a9b2334da9953e530f27bcac22082a9f5b835

Getting pack 816a9b2334da9953e530f27bcac22082a9f5b835

which contains cfda3bf379e4f8dba8717dee55aab78aef7f4daf

walk 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

walk a11bef06a3f659402fe7563abf99ad00de2209e6

智能协议

这个HTTP方法是很简单但效率不是很高。使用智能协议是传送数据的更常用的方法。这些协议在远端都有Git智能型进程在服务 － 它可以读出本地数据并计算出客户端所需要的，并生成合适的数据给它，这有两类传输数据的进程：一对用于上传数据和一对用于下载。

上传数据

为了上传数据至远端， Git 使用 send-pack 和 receive-pack 进程。这个 send-pack 进程运行在客户端上，它连接至远端运行的 receive-pack 进程。

举例来说，你在你的项目上运行了 git push origin master, 并且 origin 被定义为一个使用SSH协议的URL。 Git 会使用 send-pack 进程，它会启动一个基于SSH的连接到服务器。它尝试像这样透过SSH在服务端运行命令：

$ ssh -x git@github.com "git-receive-pack 'schacon/simplegit-progit.git'"

005bca82a6dff817ec66f4437202690a93763949 refs/heads/master report-status delete-refs

003e085bb3bcb608e1e84b2432f8ecbe6306e7e7 refs/heads/topic

0000

这里的 git-receive-pack 命令会立即对它所拥有的每一个引用响应一行 － 在这个例子中，只有 master 分支和它的SHA值。这里第1行也包含了服务端的能力列表（这里是 report-status 和 delete-refs）。

每一行以4字节的十六进制开始，用于指定整行的长度。你看到第1行以005b开始，这在十六进制中表示91，意味着第1行有91字节长。下一行以003e起始，表示有62字节长，所以需要读剩下的62字节。再下一行是0000开始，表示服务器已完成了引用列表过程。

现在它知道了服务端的状态，你的 send-pack 进程会判断哪些 commit 是它所拥有但服务端没有的。针对每个引用，这次推送都会告诉对端的 receive-pack 这个信息。举例说，如果你在更新 master 分支，并且增加experiment 分支，这个 send-pack 将会是像这样：

0085ca82a6dff817ec66f44342007202690a93763949 15027957951b64cf874c3557a0f3547bd83b3ff6 refs/heads/master report-status

00670000000000000000000000000000000000000000 cdfdb42577e2506715f8cfeacdbabc092bf63e8d refs/heads/experiment

0000

这里的全'0'的SHA-1值表示之前没有过这个对象 － 因为你是在添加新的 experiment 引用。如果你在删除一个引用，你会看到相反的： 就是右边是全'0'。

Git 针对每个引用发送这样一行信息，就是旧的SHA值，新的SHA值，和将要更新的引用的名称。第1行还会包含有客户端的能力。下一步，客户端会发送一个所有那些服务端所没有的对象的一个打包文件。最后，服务端以成功(或者失败)来响应：

000Aunpack ok

下载数据

当你在下载数据时， fetch-pack 和 upload-pack 进程就起作用了。客户端启动 fetch-pack 进程，连接至远端的upload-pack 进程，以协商后续数据传输过程。

在远端仓库有不同的方式启动 upload-pack 进程。你可以使用与 receive-pack 相同的透过SSH管道的方式，也可以通过 Git 后台来启动这个进程，它默认监听在9418号端口上。这里 fetch-pack 进程在连接后像这样向后台发送数据：

003fgit-upload-pack schacon/simplegit-progit.git\0host=myserver.com\0

它也是以4字节指定后续字节长度的方式开始，然后是要运行的命令，和一个空字节，然后是服务端的主机名，再跟随一个最后的空字节。 Git 后台进程会检查这个命令是否可以运行，以及那个仓库是否存在，以及是否具有公开权限。如果所有检查都通过了，它会启动这个 upload-pack 进程并将客户端的请求移交给它。

如果你透过SSH使用获取功能， fetch-pack 会像这样运行：

$ ssh -x git@github.com "git-upload-pack 'schacon/simplegit-progit.git'"

不管哪种方式，在 fetch-pack 连接之后， upload-pack 都会以这种形式返回：

0088ca82a6dff817ec66f44342007202690a93763949 HEAD\0multi\_ack thin-pack \

side-band side-band-64k ofs-delta shallow no-progress include-tag

003fca82a6dff817ec66f44342007202690a93763949 refs/heads/master

003e085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 refs/heads/topic

0000

这与 receive-pack 响应很类似，但是这里指的能力是不同的。而且它还会指出HEAD引用，让客户端可以检查是否是一份克隆。

在这里， fetch-pack 进程检查它自己所拥有的对象和所有它需要的对象，通过发送 "want" 和所需对象的SHA值，发送 "have" 和所有它已拥有的对象的SHA值。在列表完成时，再发送 "done" 通知 upload-pack 进程开始发送所需对象的打包文件。这个过程看起来像这样：

0054want ca82a6dff817ec66f44342007202690a93763949 ofs-delta

0032have 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

0000

0009done

这是传输协议的一个很基础的例子，在更复杂的例子中，客户端可能会支持 multi\_ack 或者 side-band 能力；但是这个例子中展示了智能协议的基本交互过程。