使用 CURL 来发送 HTTP 请求的方法

原文：Online: http://curl.haxx.se/docs/httpscripting.html

翻译：豆连军 doulianjun@gmail.com

本文假设您已经熟悉了 HTML 和基本网络知识。

拥有脚本语言编写能力对于设计一个漂亮的计算机系统非常重要。UNIX 有 shell 脚本和各种 Tools 工具，这些可以用来执行各种自动化命令和脚本，从而大大拓展了 UNIX 的计算能力。这是 UNIX 为什么如此成功的原因之一。

越来越多的应用转向了 WEB，这使得 HTTP 执行脚本变得更频繁和迫切。为了从 web 获得更多的信息，模拟用户浏览、发帖、上传数据等，今天已经成了我们重要工作内容。

Curl 是一个命令行工具，用来执行各种 URL 操作和信息传输。但在本文中将着重描述如何用 Curl 来完美地执行 HTTP 请求。我假定您知道如何调用'curl –help'或者'curl –manual'来获得 curl 的基本信息。

Curl 自然不是所有事情都能干。Curl 只是生成请求包，获得数据，发送数据获得信息。你可以使用脚本语言或者重复手工调用来完成你想要做的所有事情。

1. HTTP 协议

HTTP 用来从 WEB 服务器获得数据。它也是一种建筑在 TCP/IP 之上的简单协议。HTTP 允许从客户端向服务器端发送数据，这些数据有多种不同的操作方法。这里将给予说明。

HTTP 是一些 ASCII 文字行，这些 ASCII 文字从客户端发送给服务器端来请求一个特别的操作。然后，服务器端在发送给客户端的实际请求内容之前回应一些文字行。

客户端，也就是 curl，发送一个 HTTP 请求，该请求包含一个操作方法（比如：GET,POST,HEAD 等），一组请求头，以及某些时候再携带一些请求消息体。HTTP 服务器响应一个状态行（表明操作结果是否成功），响应头，以及大多数情况下还有响应消息体。消息体部分是你请求的实际数据，比如 HTML 或者图片等。

1.1 查看协议

使用 Curl 选项 –verbose(或者-v)可以显示 curl 发送了什么样的命令给服务器端，以及显示其他的一些信息。

--verbose 是一个非常有用的选项，可以用来调试以及历届 curl 和 server 端之间的交互操作。

有时—verbose 选项仍然不满足调试需求，这时—trace 和—trace-ascii 选项可以提供 curl 发送和接收的所有详细信息。下面是示例：

```
# curl --trace-ascii debugdump.txt http://www.example.com/
```

2. URL

URL 是你用来表达一个特定互联网资源如何定位寻址的一种格式。你看到的 URL 比如：

http://curl.haxx.se or https://yourbank.com a million times.

3. 获得一个网页

最简单和最常用的 HTTP 请求操作是 Get 一个 URL。这个 URL 可能指向一个 web 页面，一副图片，或者一个文件。客户端表达一个 GET 请求发送给服务器端，并接收所请的文档。比如：你表达了一个命令行：

```
# curl http://curl.haxx.se
```

在终端窗口中你会获得一个 web 页面，这就是 URL 指向的完整的 HTML 文档。

所有 HTTP 应答都包含了一组响应头，这些响应头通常被隐藏起来。使用 curl 的—include(-i)选项可以显示这些响应头。你可以使用—head（-I）选项来单单请求响应头。这是通过 curl 发送一个 HEAD 请求来实现的。

4. Form 表单

Form 表单的这样一种作用：网站用来提供一个网页，该网页包含一组 Fields，用户需要输入数据，然后点击“OK”或者“确认”按钮，发送这些数据到服务器端。服务器然后使用这些发过来的数据来决定如何做下一步响应。比如使用输入的文字来检索数据库，或者在一个 bug 跟踪系统中登记信息，在一个地图上显示输入的地址，或者校验输入的用户名口令是否合法。

当然，接收你发送数据的服务器端还是存在着某种程序，你不能凭空产生。

4.1 GET

一个 GET 表单使用了 GET 方法，HTML 中是如下描述的：

```
<form method="GET" action="junk.cgi">
  <input type=text name="birthyear">
  <input type=submit name=press value="OK">
</form>
```

在浏览器端，该表单会展现一个文字输入框和“OK”按钮。如果你输入“1905”并点击“OK”按钮，浏览器会生成一个新的 URL，该 URL 将使用 GET 方法。该 "junk.cgi?birthyear=1905&press=OK"衔接在先前的 URL 的 PATH 后面。

如果原表单展现在 www.hotmail.com/when/birth.html 页面上，则第二个页面将是 "www.hotmail.com/when/junk.cgi?birthyear=1905&press=OK".

绝大多数搜索引擎采用这种工作方式。

要使 Curl 做同样的事情，只需要键入如下命令：

```
 # curl
"http://www.hotmail.com/when/junk.cgi?birthyear=1905&press=OK"
```

4.2 POST

GET 方法将把所有输入的 Fields 名称显示在浏览器的 URL 中。这一般是一种比较好的做法，这种做法可以帮助你来保持书签。但是这也是一种显而易见的危险行为，比如你输入了秘密信息，或者输入了大量的 Fields 造成 URL 非常长难以阅读。

HTTP 协议提供了 POST 方法。该方法在客户端发送时，将数据从 URL 中分离出来，因此你不会在 URL 地址栏中看到任何数据内容。

表单与之前描述的表单非常类似：

```
<form method="POST" action="junk.cgi">
  <input type=text name="birthyear">
  <input type=submit name=press value=" OK ">
</form>
```

使用 curl 发送携带了上次数据的表单，我们可以这样做：

```
# curl --data "birthyear=1905&press=%20OK%20"
http://www.example.com/when.cgi
```

这种 POST 操作使用的 Content-Type 为 application/x-www-form-urlencoded ，并且这是一种应用最广泛的 POST 方法。

你发送给服务器端的数据必须是已经被准确地编码了。curl 不会自动帮你做此事。例如：如果你想使用包含了空格的数据，你需要使用%20 来替换空格等。错误的请求将很可能造成你发送的数据错误，并出现混乱。

新版本的 curl 能够执行 URL 编码的 POST 数据，比如：

```
# curl --data-urlencode "name=I am Daniel" http://www.example.com
```

4.3 文件上传 POST

在 1995 年定义了一种通过 HTTP 协议 POST 数据的方法。这就是 RFC 1867。其中 RFC1867-posting 部分将在此被引用。

该方法主要设计来支持文件的上传。一个表单允许用户上传一个文件，如同下面的 HTML 代码：

```
   <form method="POST" enctype='multipart/form-data'
action="upload.cgi">

     <input type=file name=upload>

     <input type=submit name=press value="OK">

   </form>
```

上面代码清楚地描述了将要发送内容的 Content-Type 是 multipart/form-data。

POST 一个表单，如果使用 CURL，则你只需要键入下面的命令：

```
# curl --form upload=@localfilename --form press=OK [URL]
```

4.4 隐藏 Fields

在 HTML 应用中，有一个很常用的在页面之间传输状态信息的方法，就是将隐藏 Fields 放到表单中。隐藏 Fields 事先被填充好了数据。这些 Fields 不会被显示给用户看，它们会跟其他 Fields 一道被传送给服务器端。

同一个可见 Field 用法类似，一个隐藏 Field 和一个 submit 按钮见下述示例：

```
   <form method="POST" action="foobar.cgi">

     <input type=text name="birthyear">

     <input type=hidden name="person" value="daniel">

     <input type=submit name="press" value="OK">

   </form>
```

使用 curl 来发送上述表单，你不必考虑这些 Fields 是隐藏的或者不是隐藏的。实现上述功能的 Curl 用法与上例中是相同的。

```
# curl --data "birthyear=1905&press=OK&person=daniel" [URL]
```

4.5 展示 POST 请求

当你使用 curl 代替浏览器填充一个表单并发送给服务器端时，你一定会对发送和浏览器一样的 POST 请求感兴趣。

一个简单的办法可以看到这个过程。你可以将包含 Form 表单的 HTML 页面存储到本地硬盘上，修改表单的 method 为 GET，然后点击 submit 按钮（当然，你也可以修改 Action URL）。

你将清楚地看到发送的数据附加在 URL 之后，它们之间采用'?'符号分割，就如同 GET 表单的发送过程一样。

5. PUT

上传数据到一个 HTTP 服务器的可能最好的方法是使用 PUT 操作。然后，当然服务器端必须有一个脚本或者程序来接收这些 HTTP PUT 流。

使用 CURL 上传一个文件到 HTTP 服务器，用法如下：

```
# curl --upload-file uploadfile
http://www.example.com/receive.cgi
```

6. HTTP 认证鉴权

HTTP 认证(Authentication)是一种通过校验你的用户名及密码来验证你是否有权执行相关操作的机制。基本的鉴权机制也是缺省的机制是*plain* *text* based, 顾名思义，它发送的用户名和密码看起来稍显杂乱，但仍然可以被读懂。

通过 Curl 来使用用户名口令的鉴权，示例如下：

```
# curl --user name:password http://www.example.com
```

网站可能要求一种不同的鉴权机制（检查服务器端回应的 Header 头），这时，你可以使用--ntlm, --digest, --negotiate 或 --anyauth 等选项。

某些时候，你的 HTTP 访问需要通过 HTTP 代理完成。这是在很多公司存在的。HTTP 代理一般需要验证它自己发放的账户口令来决定是否准许访问 INTERNET。

用 curl 来实现上述要求，可以使用如下命令行：

```
# curl --proxy-user proxyuser:proxypassword curl.haxx.se
```

如果你的代理要求使用 NTLM 方法来认证，则使用—proxy-ntlm 选项。如果要求 Digest，则使用—proxy-digest 选项。

如果你使用任何 user+passord 选项，但是没有标明 password 部分，curl 将提示一个 password 交互窗口。

要注意的是，当一个程序执行时，查看这些进程时，它的参数可能会显示出来。这样，如果你执行一个直白的命令行选项，则其他用户可能会看到你的密码。这里有避免的方法。

这值得注意的是，HTTP 认证时如何工作的。很多很多 WEB 站点不会使用这种登陆认证方式。后面的 WEB 登陆章节会有进一步的细节说明。

7. Referer

HTTP 请求可以包含一个'referer'字段(虽然这个拼写有误)，该字段说明这个 URL 请求是来自于客户端的那个 URL 文档。某些程序/脚本检查请求中的 referer 字段来校验它是否来自于外部站点或者一个未知的页面。当然，检查如此容易的伪造是一个傻瓜做法。许多脚本仍然在这么做。使用 CURL，你可以在 referer 字段中放任何数据，并且非常容易地愚弄服务器端来响应你的请求。

使用 CURL 来设置 referer 字段：

```
# curl --referer http://www.example.come http://www.example.com
```

8. User Agent

与 referer 字段非常类似，所有 HTTP 请求可以设置 User-Agent 字段。这个字段说明了用户使用的是哪种代理（客户端）。许多应用使用该信息来判断如何显示页面。傻瓜 web 编码人员试图为不同的浏览器用户提供不同的页面，以提供给用户最佳的体验。他们通常也使用不同的 javascript,vbscript 等。

有时，你会看到使用 curl 获得的 web 页面与你使用浏览器获得页面有所不同。这时，

你应该知道使用 User-Agent 来糊弄这些服务器端使它们认为你就是那个它认为的浏览器。

使 curl 模拟 Windows 2000 上的 Explorer 5：

```
# curl --user-agent "Mozilla/4.0 (compatible; MSIE 5.01; Windows
NT 5.0)" [URL]
```

使 curl 模拟老 Linux 上的 Netscape 4.73：

```
# curl --user-agent "Mozilla/4.73 [en] (X11; U; Linux 2.2.15 i686)"
[URL]
```

9. Redirects

当向一个服务器请求一个资源时，回应可能包含了一个提示，这个提示说明了浏览器应该继续跳转到另外一个网页上。告诉浏览器做重定向的响应头是 Location：

Curl 不会执行重定向：响应头是先前描述的，消息体也是跟其他回应相同的显示模式。但提供了一个功能选项，可用来执行后继的重定向。

使 curl 执行重定向：

```
# curl --location http://www.example.com
```

如果你使用 curl 来 POST 到一个网址，该网址将重定向到另外一个网页，你可以安全地将—localtion(-L)和—data/--form 等选项一起结合使用。Curl 将只在第一次请求中使用 POST，然后使用 GET 来执行后继操作。

10. Cookies

web 浏览器执行“客户端状态控制”技术是通过 cookies 来实现的。 Cookies 是多个内容组合体。Cookies 由服务器端发送给客户端。服务器端告诉客户端 Cookie 的有效路径和主机名称，以便浏览器返回这些数据。同时，它也发送过期时间和一些其他特性。

客户端跟服务器端通信时，如果服务器名称、路径名都跟以前 Cookie 设定的参数相匹配，则客户端将把 Cookies 和相关内容回传给服务器，除非这些 cookie 过期才不回传。

许多应用和服务器使用这种方式来连接一系列请求为一个逻辑会话。为了使 curl 能够用

于这种场合，我们必须能够记录和回传 cookies。就如同浏览器处理它们一样。

当使用 curl 获取一个网页，最简单的方式就是发送一组 cookies 给服务器端。在命令行中如下所示：

```
# curl --cookie "name=Daniel" http://www.example.com
```

Cookies 跟其他普通的 HTTP 响应头一样传输。这里可以通过 curl 来记录 cookies，跟记录其他响应头一样。要使用—dump-header(-D)选项：

```
# curl --dump-header headers_and_cookies http://www.example.com
```

（注意，下面描述的--cookie-jar 选项是一种更好的存储 cookies 的方法。）

Curl 有一个内置的功能全面的 cookie 解析引擎，如果你想重新连接一个服务器站点，想使用前一个步骤访问站点时存储下来的 cookies（或者人工愚弄服务器端使服务器相信你有一个以前的连接），你可以使用这个功能。使用先前存储的 Cookies，你可以这么使用 Curl：

```
# curl --cookie stored_cookies_in_file http://www.example.com
```

Curl 的 Cookie 引擎在你使用—cookie 选项后开始执行。如果你仅仅想 curl 理解接收到的 cookies，使用—cookie 再跟一个不存在的文件。例如，如果你想让 curl 理解一个网页的 cookie，并且重定向到一个新的网址（当然，curl 会回送它接收到的 cookies），你可以这么引用 curl：

```
# curl --cookie nada --location http://www.example.com
```

Curl 能够读取和生成 Cookie 文件，这文件格式跟 Netscape 和 Mozilla 是一样的。这种特性为 cookies 在浏览器和自动脚本之间共享提供了一种方便性。--cookie（-b）开关自动检测一个给定文件是否为 Cookie 文件，如果是，就解析它。使用—cookie-jar(-c)选项，你可以使 curl 在操作结束时生成一个新的 cookie 文件：

```
# curl --cookie cookies.txt --cookie-jar newcookies.txt
http://www.example.com
```

11. HTTPS

有多种方式来实现 Secure HTTP 传输。最知名的协议时 HTTPS，HTTP over SSL。SSL 将所有网络上发送和接收的数据进行加密，从而避免攻击者窃取敏感信息。

SSL（或者 TLS 是最近的标准）提供一整套高级功能，允许加密和 HTTP 请求密钥加密机制。

感谢 OpenSSL 库的支持，Curl 实现了加密的支持。从一个 HTTPS 站点获得网页，只需要简单地执行如下操作：

```
# curl https://secure.example.com
```

11.1 Certificates

使用 HTTPS，你需要使用证书来证明你是否有权访问，这作为登陆密码的一个补充。Curl 支持客户端证书。所有证书在校验过程中都会被锁定，在证书可以被 curl 使用之前，你需要进入这个过程。这个过程可以在命令行中被指定，或者在 curl 询问时人工输入。curl 在一个 HTTPS 站点上使用一个证书，可以采用如下方式：

```
# curl --cert mycert.pem https://secure.example.com
```

CURL 也将执行对证书中的服务器端的校验，该校验是通过比对本地存储的 CA 证书和服务器端的证书进行的。

错误的证书将会导致 CURL 拒绝本次连接。此时，你必须使用—insecure(-k)确保你想告诉 curl 去忽略服务器端的失败校验。

更多关于服务器端证书和 CA 机制，请参阅 SSLCERTS 文档，线上地址为：

http://curl.haxx.se/docs/sslcerts.html

12. 定制请求元素

要想做得漂亮，你需要增加或者修改 curl 的请求元素。例如，你可以修改 POST 请求指向一个 PROPFIND，并且使发送的数据采用”Content-Type: text/xml”（代替缺省的

Content-Type），则相应 curl 用法如下：

```
# curl --data "<xml>" --header "Content-Type: text/xml"
--request PROPFIND url.com
```

你可以通过传入一个不存在的内容来删除一个缺省的 Header。比如你可以去掉一个 Host 头来修改一个请求。

```
# curl --header "Host:" http://www.example.com
```

类似的方式，你可以添加 Headers 头。你的服务器如果要增加一个"Destination:"头，你可以如下方式使用 Curl：

```
# curl --header "Destination: http://nowhere" http://example.com
```

13. Web Login

由于不只是 HTTP 方面的问题，仍然有许多困难。这里详细描述了各种登陆表单如何工作的原理以及如何使用 CURL 来模拟登录。

需要注意的是，要自动登陆，大多数情况下，你需要编写脚本，反复调用 curl。

首先，服务器端大多数都是使用 cookies 来跟踪客户端登陆状态。因此，你应该捕获服务器端的 Cookies 响应，然后，许多站点会在登录页面设置一个特殊的 cookie（请确认你已登陆了）。所以，你应习惯于首先抓获一个登陆表单来捕获其上的 cookies。

某些 web 登陆页使用了功能复杂的 javascript，有时他们使用这些代码来设置和修改 cookie 内容。也有可能他们想通过这种方式来阻挡机器人登陆，就象本文描述的这样……。但不管如何，如果阅读这些代码不足以让你实现自动登陆，你需要通过浏览器来捕获 http 请求，并分析这些 cookies 的发送过程，以找到剥离 javascript 代码的途径。

在实际登陆页的 form 标签中，许多站点会隐藏很多秘密标签或填充一些随机数或 session。你需要首先捕获登陆表单的 HTML 代码，并且抽取所有的隐藏字段，来正确地发送 POST 登陆请求。切记，采用普通 POST 命令发送数据时，内容需要使用 URL 编码。

14. Debug

很多时候，当你在一个站点上使用 curl，你会注意到站点并不像你使用浏览器一样给你想要的响应。

然后，你需要使 curl 可以发出与你浏览器更类似的请求数据包。下面是几个有用的调试命令选项：

* 使用—trac-ascii 选项来存储详细的请求 log 日志，这可以用来辅助分析和理解。

* 检查确认是否有必要使用 cookies（参考—cookie 和—cookie-jar）

* 设置 User-Agent 为最近流行的浏览器类型。

* 设置 referer 为浏览器上的数值。

* 如果你使用 POST，需要确认你发送了所有的字段，并且字段顺序跟浏览器一致。（见 4.5 章节）

一个可以帮助你正确操作的工具是 LiveHttpHeader tool，这可以帮助你在 Mozilla/Firefox 上查看你发送和接收的所有 Headers。（包括你使用 HTTPS 场合）

一个更基本的目的是捕获 HTTP 网络通信数据，这可以使用 ethereal 或者 tcpdump 等工具来做到。通过这些工具可以帮助检查浏览器发送了和接收了哪些 Headers。（HTTPS 不适用）

15. References

RFC 2616 是你理解 HTTP 协议所必须阅读的。

RFC 3986 解释了 URL 语法。

RFC 2109 定义了 cookies 是如何工作的。

RFC 1867 定义了 HTTP post 上传格式。

http://curl.haxx.se 是 curl 的官方网站。