POWERBROWSING

v1.2.0 (en)

{Written by +mala, 2004/09/20}
malattia(at)gmx(dot)net
[http://www.reteam.org -/- http://3564020356.org]

This paper is dedicated to Scott R.Lemmon, Proxomitron's author. What you've started will never stop, and your ideas will be forever alive, around the Net and inside our minds.

A big THANK YOU to Andreageddon, who helped me with the English translation when I didn't have enough time to do it :)

1. INTRO

What is PowerBrowsing? The text you can read around the title is a good explaination of this word: PowerBrowsing means browsing the Web seeing only what you chose to see. Even if this might seem an easy thing to do, it is not and it will become harder and harder in the future... unless, of course, you become "PowerBrowsers".
This text tries to explain how the Web works now and how you can PowerBrowse, using ready made tools or creating new ones which fit your needs.

1.1 How are you browsing now?

How do most of you browse now? Well, you probably use the browser which is installed by default in your system which, most of the times, means Internet Explorer. The chances to customize the way you see Web pages is limited by the options your browser allows you to change, so you probably download all the images inside a website and all the active contents like Flash and Java applets, you see more and more advertisements inside Web pages and you have to close tons of popups when you visit some websites.

I'm sure that some of you are already getting angry for this generalization, because they use another browser... hey, maybe another operating system too! Well, let me try to guess anyway: despite of this, you usually follow the links your browser (whatever it is) shows you, you see pages inside windows chosen by it and you access only the URLs it lets you visit. In some cases, to get the information you are interested in, you have to follow some fixed paths inside a website (like in autogrills or supermarkets... and every website, as a supermarket, wants you to see all of its special offers). And last, but not least, you always download much more HTML code than you really need: maybe you don't see it, but be sure your modem notices it!

Well, anyway, why bother? Maybe it's not exactly what you wanted, but modems

are becoming faster and faster, and after all this is what you're given and you can't change it much.

1.2 How does it work, instead?

Hey, wake up! I've got a piece of news for you: a computer is not a TV! It's supposed to do what _you_ ask, not what _others_ want it to do. Also, things are not always as they look: before the end of this text, you will realize that most of the times you're able to see what a browser doesn't directly show you, and to hide what it shows you by default.

Now, think about downloading one tenth of the data you usually download, skipping all the advertisements, avoiding popups, keeping interesting data (and only them) on your computer and accessing them while you're offline, in a custom, easier and more effective way. Think that, once those information are on your hard disk, you can write programs which work on them to produce new, even more interesting information. Finally, think about the chance of making all these data available to everyone, maybe in an automatical way.

All of this is what I call PowerBrowsing.

1.3 Notes

Well, before you read this text I think I need to write some notes about it. First of all, this paper was born as a lecture i kept at Genova (Italy) in April, 2004. The slides I prepared were then adapted to become a full text, but of course still have some characteristics of the original speech you might not expect.

First of all, the speech was intended for a mixed audience, so it is this text: this means that, while some of you might find some parts of this tute interesting and others too hard, some others might become really bored before they find something they're interested in, and other ones might find nothing which deserves to be remembered here. My suggestion to everybody is to skip all the parts you don't like and, in case nothing is left, no problem: just think you've surely wasted less time reading this text than me writing all of it.

Finally, I think I could not write this paper without the book "Spidering Hacks", by Tara Calishain and Kevin Hemenway (aka Morbus Iff), published by O'Reilly, which gave me many interesting ideas and which everybody who wants to create Web bots should read (and, why not, maybe BUY too, if you think like me that the author deserves it). Between the many interesting info you can find inside this book, there are some nice notes about bot netiquette everyone should read (and follow).

2. TECHNOLOGIES

To understand what happens inside your PC when you download a Web page, you should know the technologies the Web is based on. Inside this text I'll give HTTP and HTML basics for granted, anyway I'll try to explain things while I'm writing about them. Since, of course, most of what I'll write will be hard to understand anyway, here are some links to websites you might find useful:

http://www.w3.org/MarkUp/ (everything you need to know about HTML)
http://www.w3.org/Protocols/ (everything you need to know about HTTP)

2.1 Why HTTP?

To tell you the truth, you won't need to know all the details regarding this protocol. Anyway, in the next sections of this text you will read something about HTTP-related concepts, and knowing their meaning in advance will help you much and let you understand what your Perl bots will be able to do. Moreover, knowing what kind of data your computer will exchange with the servers it connects to will help you to create more stable and more secure bots.

GET and POST

GET and POST "methods" are the first two terms we will learn: they represent the two different ways your browser uses to ask servers for data. Without delving too deeply into the details, the main features of the two methods are the following ones:

- GET can be used both to ask a static page and to send parameters to a dynamically generated one (CGI, PHP and so on). The typical URL of a GET is like

  http://www.web.site/page.php?parameter1=value1&parameter2=value2

  (you probably have read this kind of urls before, inside the address bar of your browser). The amount of data you can send with a GET is quite limited, also you should keep in mind that the params you send with a GET are usually saved inside Web server logs (too).

- POST is used only when you want to send data to the Web server. The amount of byte you can send is higher than with GET and inside server's logs you

will be able to find only the URL you POST to, and not the data you have sent. This is quite important if, for example, you want to create some bots which automatically authenticate and log inside a website: in this case POST is better than GET, but keep in mind that, if the connection is in clear, your login and pass won't be safe anyway (as they aren't now with your browser).

Referer

Among the many parameters that are usually sent together inside a GET or POST request, whenever you reach a page following a link browsers usually send your referer to the server, that is the URL of the website you're coming from. This piece of information can be used by the server itself to check if you're coming from one of its pages (and not, for example, from your hard disk: countless hacks can be made just editing a web page locally!), or to create statistics about how many users come from some particular websites.

User-Agent

The User Agent is the software which connects to a server and communicates with it, asking the pages you've choosen and receiving them. In practice, anyway, most of the times this name is used for something else: it's the name of the string used by the app (which could be a browser or any other piece of software) to identify itself with the server.
Sometimes, this very string is used by the server to allow some programs to access a page and to keep away others: this happens, for instance, with some "optimized for Internet Explorer" websites. The most intelligent browsers (if you ask, NO, Internet Explorer is not one of them) allow you to send different User-Agent strings (custom or standard, ready made ones), and if you plan to write some serious piece of Web software you should add this feature too.

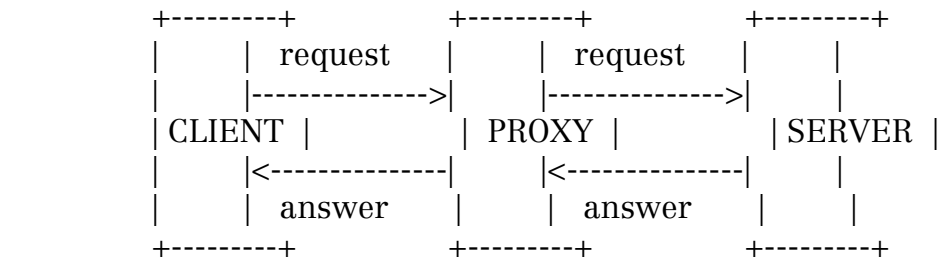
Cookie

Cookies are text files which are automatically saved by your browser on your hard disk. They contain different kind of information, most of the times related to your previous connections to a website or your authentication. For this reason, cookies are often disliked by the ones who want to defend their privacy. However, cookies are used almost everywhere now and some websites don't even work if the application which connects there doesn't support them.

Proxy

Proxies are programs which forward your Web apps requests to the desired servers and return their answers to the clients. Their job can be easily

described with this:



Proxies are very useful for many different reasons:

- the client might not be able to access servers, but it might be authorized to access the proxy: in this case, client apps could reach the server anyway, passing their requests through the proxy

- some proxies have a cache, inside which the most frequently requested files are saved. So, if the connection between the client and the proxy is much faster than the one between client and server, then you might be able to download cached files much faster

- some proxies don't tell the server where requests come from, so the clients are able to connect anonymously to the Net

- later, we'll see how you can use proxies to get many more, and even more interesting, advantages.

2.2 Why HTML?

Because everything your browser shows you has its own HTML source, created by hand by someone, generated by some program or by a script. For this reason you shouldn't know only HTML basics, but you should be able to understand if the code has been automatically generated or not, so you can use multiple tag occurrencies to split a page in sections and extract its contents.

Since inside dynamic websites forms are always present, you should spend some time to understand how this technology works. To tell you the truth, it's not such a hard work, anyway some experience in this field will help you not only to easily understand the syntax and the meaning of what you see, but also to understand how a whole website works. And without any particular intrusive technique, but only with forms and html knowledge (and well, yes, some brain too), you'll be able to create bots which can do much more than you can

imagine now.

3. PB TECHNIQUES: tools and basics

Around the Net you can find lots of ready made PowerBrowsing tools for free (or, if you prefer, sold for a lot of money), for any operating system. Describing all of them in detail is impossible, so we'll try to group them in categories and describe their main characteristics.

3.1 Alternative browsers

Alternative browsers are, basically, all the ones which are not Internet Explorer. Even if they're not the definitive solution to any problem, their choice is the first step you can do to free your system from unrequested contents, such as advertising banners, flash menus and popup windows.

Opera, for instance, allows you to toggle image loading in a Web page (or to disable just non-cached images, such as banners) with a simple clic, while the same operation with IE requires you to browse for a while inside all the configuration menus. Opera also allows you to quickly toggle Javascript or other dynamic contents and, in the same way, you can choose a custom view of Web pages, with fonts and colors chosen by you instead the ones decided by the page creators.

Between all the browsers with a GUI, I've heard very good comments about Firebird. I haven't tried it enough to write about it here, so I'm waiting for many feedbacks and "PowerBrowsing hints" about it. Anyway, without going

far from this, even Mozilla is, for many reasons, a better choice than MSIE, and -compared to many newer browsers- it also has a greater compatibility with many websites (which you pay with a heavier and more bloated app).

If, instead, what you want is only the text of a Web page, you might consider the idea of using a textual browser like lynx or links or w3c: the speed of page downloads, without all the heavy contents, will amaze you. Moreover, as you'll be able to see later, loading the contents of a page from the command line gives you the chance to operate with Web information in a more effective -even if maybe less conventional- way.

3.2 Leechers and Teleporters

The programs pertaining to this category allow you to "leech" or "teleport" contents from the Web. Both of them usually download lots of files on your hard disk: the first ones are more simple, getting every file of some kind

or all the contents of a directory; the second ones (whose name derives from the old win application Teleport Pro) allow you to get the contents of a website, automatically following its links and trying to replicate on your disk its inner structure (that is, "mirroring" it).

Even in this case, there are so many applications that it's impossible not only to describe, but even to know all of them. Personally, the ones I liked most during the years were Teleport Pro and Getright under Windows, and wget, curl and lynx under Linux. In fact, the distinction between Windows and Linux apps is not so sharp now, as now you can often run programs of one OS under the other: the result is that now the tools I use most frequently are wget, lynx and GetRight (still an old version, but very useful for its "GetRight Browser", which connects to a Web page and shows all the linked files, allowing you to choose and download only the ones you're really interested in).

Probably you've already noticed that, even if lynx is a browser, it fell into this category too. This happened because, thanks to its -source and -dump switches and since its output can be piped to other programs, it can work as a leecher too. As you will see later (in section 4.3), using it as a leecher you can download files and information in a very efficient way.

3.3 Spiders and scrapers

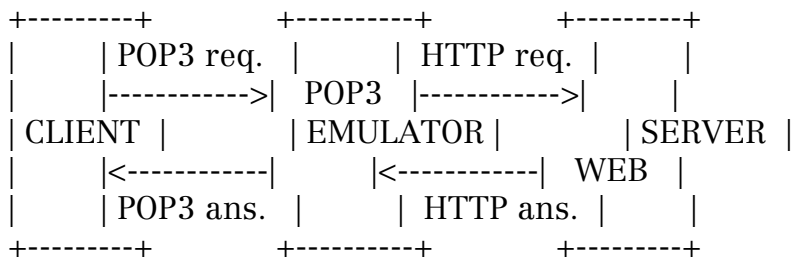
Since, until some months ago, I didn't even know this kind of categorization, I decided to quote (more or less) the two definitions I found on "Spidering Hacks":

- "spiders" are programs which automatically retrieve data from the Web. Usually, they're smarter than teleporters (they don't just follow all the links they find, but they can choose them depending on some algorithm) and they usually download the whole content of the pages they find;

- "scrapers" are programs which extract only some specific parts out of Web pages. In fact they often have to download whole pages anyway, however they can save on your disk only the data you're really interested in and not all the pages' contents.

You won't probably find many ready-to-use spiders and scrapers around the Web so easily, however there are some interesting ones: for instance, liberopop (with all its variants) is a program which allows you to download with your email client the messages you receive in your mailbox, which could otherwise be browsed only on the Web. Of course, the problem of providers which first offer free mail and then close their pop3 servers is not new, and different

solutions have been found, during the years, to fight this trend: just give a look at "Perl@usa.net" paper by Blue (which you can find on any fravia's mirror, in the bot -botstart.htm- section), dated 1999! Anyway, I liked liberopop's structure



which not only reminds a proxy very much, but is also almost identical to the one I designed for an old app of mine (ANO - Another Non-working Offline forum reader), which allowed you to read web forums with your email client. If you're interested in this kind of apps, you might like one of my latest projects, called TWO (The Working Offline forum reader), a scraper I'll describe with a little more detail in section 7.4.

3.4 Proxy-like software

The programs which fall into this category use the same architecture of proxy servers to get results you don't usually expect from an application of this kind. Thanks to their intermediate position between client and server (they are often called "middlewares"), they can transparently work on data travelling in both directions: for instance, they can filter information coming from the server and take only what you're interested in, or changing pages on the fly; in the same way, they can read the data you send to the server and use them later, to automatically authenticate, log and browse into a website.

An example of the first kind of proxy (that is, the one which filters data coming from the server) is Proxomitron, a small but great Windows application which allows you to filter Web pages, cutting away everything you don't like (banners, popups, javascript, spyware) and completely changing their look. Trying to create a platform-independent version of this program, a group of reversers is working on a project, called Philtron, which aims to create a PHP, Proxomitron-compatible application with even more functions. You can find more about this project at these URLs:

http://philtron.sf.net (main project page)
http://fravia.2113.ch/phplab/mbs.php3 (PHPLabs)

http://fravia.2113.ch/phplab/mbs.php3/mb001 (Seeker's messageboard)

For what concerns the second proxy type (the one which works on data you send to the server), a good example is the Web Scraping Proxy: this Perl app can "record" everything you do inside a website, then it automatically creates the source code needed to make a perl bot which will mimic all your actions. To know something more about this program, check the website

http://www.research.att.com/~hpk/wsp

or give a look at the "hack" (number 30) inside Spidering Hacks.

Note: I've recently found another perl package which should do the same thing. It's called HTTP::Recorder and you can easily find it at CPAN (try the search engine at http://search.cpan.org).

4. PB TECHNIQUES: advanced

More advanced PowerBrowsing techniques don't just use a single program, no matter how advanced its options are: they usually take advantage of both the knowledge acquired by users and the functions provided by one or more tools, which can be ready made ones or created ad hoc. This way you get new, more powerful and effective tools. Moreover, reading about more and more complex experiments, you'll notice how we'll move from the simple file download to a more general (and, IMO, much more interesting) _information retrieval_.

Of course, PowerBrowser's skills are fundamental here and the higher is your experience, the better results will be. Anyway, even with the few, simple suggestions which follow you'll be able to find and download much more easily everything you like.

Speaking about examples, keep in mind that they are not supposed to be very useful in practice. Moreover, given the speed with which Web information change, some of them even might not work anymore when you read this document. Don't worry too much about this: read them, understand them, try to change them and be sure that you'll have something more than some lines of code.

4.1 Learn to search

As you probably have understood yet, if you want to see only what you're interested in you first have to _find_ what you're interested in. On the other side, in some cases you might already know where that file you wanted to download resides, but for some reason you don't want to connect to the

website it's stored in (for instance, because you have to pay to do that): even in this case, knowing how to search will help you to find alternative sites from which you'll be able to get the same file.

A first suggestion I'd give to anyone is to visit the good Searchlores site (http://searchlores.org). Inside it you will be able to find many tutorials about Web research and technologies, inside a place which is completely free from advertisements, banners and commercial stuff. In particular, I suggest you to give a look at "search webbits", ad hoc search strings for some file or information categories.

Between these, the "index of" trick is one of the best ones, even if some commercial websites are already trying to use it to attract searchers to their pages. In practice, you can use it to restrict the Web search to those directories which are open on the Web: these ones are nothing more than long file lists, and always have at their beginning the "Index of <dir name>" header and a link to their "parent directory".

Now, if for instance you want to download ringtones without paying a cent, why do you have to get lost between dialer and commercial websites when you can download all the songs you like in midi format? To find the websites which share them freely, you just need to feed google with the following string:

"index of" "parent directory" ringtones .mid

Another example is about those funny videos you download when you don't know how to spend time inside your office. With the string

"index of" "parent directory" fun .mpg

(change .mpg with your favorite video format) you can find all the videos you like and, if you're lucky, even some websites which are regularly updated with new funny resources.

If, instead, you want to try the same trick with mp3s, you'll probably find lots of wrong results, which link you to commercial websites. This happens because, as I told you before, once you start using frequently a trick "on this side", this becomes used "on the other side" (the choice of which side is the dark one is left as an exercise to the reader) to attract people where they don't want to go. Fortunately, regardless of how many techniques they'll try to use to trick us, we will always be one step before them ;)

Quite banally, if you want to take away most of the fake results from your search, you can try to cut away the classic web pages extensions. If you see

you still get too many results, you can add some filters on specific terms:

"index of" "parent directory" .mp3 Iron Maiden -.html -.htm -faq

Here, for instance, we're searching some Iron Maiden mp3s, cutting away HTML pages (which are not interesting for this research) and the results which contain the "FAQ" word, because in some newsgroup's FAQ somebody talked about both Iron Maiden and mp3s and someone else had the great idea of mirroring them almost anywhere.

If, finally, you even know the song's title, you can try to insert its last word, joined to the file extension, or add a part of the title to the search strings:

"index of" "parent directory" Metallica frantic.mp3 -.html -.htm

Another suggestion I can give you about web searching is to use webtrackers. Many Web sites use them to have access stats to study: what many don't know, anyway, is that many commercial trackers are open to anyone and let you see, between many different stats, referrers too. For instance, try to give a look at my webtracker:

http://extremetracking.com/open?login=alam

As I've told you previously, referrers are the URLs from which users came when they hit a website, in this case the one which uses the webtracker. Of course, there's a high chance that many referrer websites will be devoted to the same topic: so, you'll just have to search on google the topic you're interested in and the name of a webtracker (or its URL, or a string which uniquely identifies it) to start delving deep inside a mine full of potentially interesting links.

A last suggestion, if you're searching a particular file and especially if it's a copyrighted one, is to give a look to peer to peer channels first: the download will probably be slower than the one from a Web site, but you'll have much more chances to find a movie or a book, even if you have only a couple of words from its title... And, once you have the filename, you might even decide to get back to normal Web search.

If, while searching for a particular file on a p2p network, you happen to find the names of groups which periodically release files of the same kind (for instance, horror movies or comics or tv series or whatever else), write them down: next time you'll have more chances to find what you're searching for, using these names between your search strings. In the same way, following (while paying attention, of course) the links you can find inside

the classic .nfo files, I happened to find some monothematic communities, much more specialized and full of contents than any search engine I used.

4.2 Experiments with curl

Inside some websites you might happen to find long lists of files of the same type, whose filename has always the same prefix followed by an incremental number. If the file list is visible (that is, browseable) from the Web, with an HTML page, or because the directory where the files are stored is accessible, the easiest method to download all the files is always wget:

wget -m -np http://web.site.url/directory/

If you want, you can specify the extension of the files you want to download:

wget -m -np -A <extension> http://web.site.url/directory/

Unfortunately, direct access to directories is often closed and many sites don't give you a file index, but force users to download at least as many pages (with unuseful images, banners and popups) as the files you want to download. For this task, the most useful utility you can find is curl.

Curl allows you to choose, through the command line, one or more URLs, and to specify which parts of them change (putting them between curly brackets) or which are the sequencies of alphanumeric characters the program has to try (writing them between square brackets). For instance,

http://web.site.url/directory/file[1-100].txt
http://web.site.url/directory/file[001-100].txt
http://web.site.url/directory/file[a-z].txt
http://web.site.url/directory/file[1-4]part{One,Two,Three}.txt

allow you to download all the files which begin the same way and continue with, respectively,

- numbers from 1 to 100
- numbers from 1 to 100 (padded with zeroes to always have 3-digit numbers)
- letters from "a" to "z"
- numbers from 1 to 4, followed by "partOne", "partTwo", "partThree"

Curl has many more options than the ones described here. It also supports various protocols (HTTP, HTTPS, FTP, GOPHER, DICT, TELNET, LDAP, FILE) and can be used for file upload too (to know more about this, write "man curl" inside your shell). However, it still has some limitations: for instance, it

still cannot efficiently manage filenames which contain dates inside them.
If you run this command

curl -LO http://web.site.url/dailystrips/[2000-2004][01-12][01-31].gif

you will download all the images you're interested in, but you'll send the server more requests than you need, trying for example to download images dated February, 30th or June 31st...

There are many techniques to solve this problem: between them, there are some you will see inside the next section and that will allow you to automatically download, every day, your favorite daily strips.

4.3 Wget and lynx oneliners

Inside this section you will have the chance to see some oneliners which make use of wget and lynx. They are the result of an old project I named "Browsing the Web from the command line", which had some contributors inside RET forum (http://www.reteam.org). The project has been inactive for a long time, but since it's part of PowerBrowsing now you are free to contribute, sending comments, requests or new experiments. Thanks in advance :)

The first oneliner we'll see allows you to download daily strips from "kevinandkell.com" website (but it won't be much different for other sites).
I took it from http://www.rollmop.org/junk/oneliners.html, which has a nice collection you might want to check (the whole website is quite interesting indeed).

The oneliner gets the current date using the "date" command: to get more information about it, run "date --help" at the prompt.

wget http://www.kevinandkell.com/`date +"%Y"`/strips/kk`date +"%Y%m%d"`.gif

Note how date is called twice: it's between backquotes (in this way, the program output becomes part of the URL string) and it's passed the +"" param, inside which the desired date format is specified. The codes used to specify the date format are the following ones (taken from the help):

```
%Y   year (1970...)
%m   month (01..12)
%d   day of month (01..31)
```

So, if you run this command, for instance, on April, 2nd, 2004, wget would try to download

http://www.kevinandkell.com/2004/strips/kk20040402.gif

One of the most interesting features of these commands is that you can make them run automatically every time you power on your PC, or every day if your computer is always on: this way, you'll have all the files you want always ready on your hard disk. If between the websites you want to periodically check there are some which are not updated daily, you can change the behavior of your script: for instance, you can add some code to tell you that on a particular day the script could not download any file:

```
if [ `date +"%w"` -lt 6 ] && \
   [ $((`date +"%w"` % 2)) = 1 ];
then wget http://www.goats.com/comix/`date +"%y%m"`/goats`date
+"%y%m%d"`.png;
else echo no goats today;
fi;
```

In this script, the date command used with "%w" parameter returns the day of the week (from 1 to 7). If the day is less than 6 (that is, if it's not saturday or sunday) or if it's odd (that is, only if it's monday, wednesday or friday) then the image is downloaded; otherwise, the program tells you that no pics are available. On the same day used for the previous example, tha is 2004/04/02, the program would download the file

http://www.goats.com/comix/0404/goats040402.png

In both previous examples we used wget in the easiest way we could, that is directly passing it the URL we wanted to download. However, this program can automatically execute more advanced operations, like the following example shows:

```
wget -A mpg,mpeg,avi,asf -r -H -l 2 -nd -t 1 http://url.you.like
```

Here, wget is used alone, but with many different switches. Their meaning is the following:

-A comma-separated list of accepted extensions
-r recursive web-suck
-H go to foreign hosts when recursive
-l 2 maximum recursion depth = 2
-nd don't create directories
-t 1 set number of retries to 1

Let's try to read the single descriptions and join them to understand what

this command really does: wget first connects to the specified URL, then it recursively follows links (-r) to a maximum depth of 2 (-l 2), exiting the main website to connect to external server when needed (-H); then it saves, inside the current directory (-nd), all the files which have mpg, mpeg, avi or asf extensions. Now, if you think about those websites full of links to other sites which contain lots of free videos, you'll understand why this command has been named... "wget-powered porn"!

Few lines ago we were speaking about link collections: what if we wanted to extract all the links contained inside a page, not to follow them but just to save them inside a text file? For this task we can use lynx, joined with some other tools:

```
lynx -dump http://www.reteam.org/links.html \
    | sed 's/^ *[0-9]*\. [^h]*//' \
    | grep '^http'
```

Here we use three programs, piping the output of each one to the input of the following one:

- lynx dowloads the specified page and sends its dump to sed (keep in mind that lynx "dump" contains, at the end of the text, the list of all the links -http or not- present inside a page)

- sed left-aligns all the http links and deletes all the other ones

- grep extracts just the lines which start with "http"

In this way, with just one line of commands, we've extracted all the links inside a page, getting some new information without even opening a browser. Of course, lynx helped us much, giving us a page containing all the links, but things aren't always so easy: a more advanced operation (but not, for this reason, much harder to accomplish) is the following.

```
lynx -nolist -dump 'http://www.reteam.org/board/viewforum.php?f=3' \
    | grep -2 "Browsing the web" \
    | tail -1 \
    | awk '{ print $1 }'
```

In this case, we used four programs:

- lynx connects to the specified URL, where there's a forum, and downloads a page containing the forum's thread list

- grep extracts the thread which contains the words "Browsing the web" with a 2-lines "context" (that is, it takes 2 lines before and 2 lines after the line in which the string "Browsing the web" is found)

- tail gets the last line of the context, that is the second one after the subject line

- awk returns the first word inside this line

Of course, if you want a sequence of programs like this to be useful you don't just have to understand how the single programs work, but you should also know how the forum is organized and how, inside the Web pages sources, "static" and "dynamic" data are stored (the first ones never change, while the other ones usually change, like a message sender or a post subject). As I said previously, this is a skill you can master only with experience: so, before giving you some more suggestions, I'll show you another example and I'll explain it with more detail. Be ready for the Fight Against Flash!

4.4 Fight Against Flash

Why do we have to fight against Flash? Flash is a good technology from the communicative point of view, it allows you to obtain good graphic effects without generating too heavy downloads and it's compatible at least with the most used browsers.

In fact, Flash is not such a big problem. Its programmers, instead, are a HUGE problem when they use it the wrong way: if you have to create a website only for promotional purposes, 100% flash, I can understand that a text-only version doesn't have much sense; however, when inside a website which should provide _information_ you find a flash menu without alternative code to support less used browsers, this becomes a huge problem. And if on one side, fortunately, flash is less trendy than it was some time ago, you just have to search with google files like "menu.swf", "leftmenu.swf" or "navmenu.swf" to understand how many of them are still around.

Flash menus are a problem for different reasons: first of all, I might be interested in the rest of the website and I do not want to just leave the website and switch off my PC; moreover, because of these menus not only text browsers like lynx cannot access site contents, but also special devices like braille bars might have problems. And, even if in my case I might just open another browser, other people cannot choose and so this websites' contents are precluded to them. This is the reason why, in my opinion, it's good to fight against flash now and, more generally, against everything which is going to limit internet use in any way.

To tell you the truth, my fighting skills are not particularly aggressive:
I have simply tried to understand how flash files work, to extract as many info as I could from them and finally to rebuild the contents of the menus which weren't accessible. To do this, I've used three different approaches, each one more difficult and, at the same time, more effective than the previous ones. I still haven't found a universal solution, but I'm sure that if you work on these examples you'll be able to do something better than me.

The easiest way to extract information from an swf file is to show its strings with the "strings" command. If the file we're working on is a menu, we can suppose that links are what we're most interested in: we can then run strings, asking it to open the swf file, and then redirect its output to grep, which will extract only the strings containing "http":

```
lynx -source http://www.reteam.org/top.swf | strings | grep http
```

This first approach already gives you some results, and with some menus it works perfectly. However, it doesn't take into account many other link types: for instance, for the websites where links are managed by some javascript code, you'll have to change the string passed to grep:

```
lynx -source http://www.mypetskeleton.com/mpsmain.swf | strings | grep http

lynx -source http://www.mypetskeleton.com/mpsmain.swf | strings \
| grep javascript
```

Working on strings you can get many other interesting info, but still the links problem isn't solved: as you might have guessed yet, this kind of approach is quite blind, trying to find something interesting inside a lot of data which might or might not contain what we really want. To solve this problem, I've given a look to various swf files, trying to understand which format they use to save links inside them: in all the flash files I've checked, every link was represented by the sequence

```
0x00 0x83 0xLEN 0x00 "string" 0x00
```

where LEN is the string length, and 0x is the prefix I used to show that all the values are saved in hexadecimal.

Once I found this, creating a small script which extracted all the links with a regular expression was just a matter of seconds. If you don't know what a regular expression is, run and study them! Otherwise, here you are the script source:

```
#!/usr/bin/perl

undef $/; # enable slurp mode
$_ = <>;

# SYNTAX IS: 0x00 0x83 0xlen 0x00 "string" 0x00
while (/\x00\x83.\x00(.*?)\x00/gs){
        print "$1\n";
}
```

Yep, it's all there (you understood why you have to study regexps, didn't you?). In fact, it's nothing more than a loop which says "while you find byte sequences like the one I described before, extract the 'string' part and show it". The usage of this script, supposing the swf file is somewhere on the Web, is the following:

lynx -dump http://web.site.url/menuname.swf | perl flash.pl

For instance:

lynx -dump http://www.reteam.org/top.swf | perl flash.pl

Thanks to the script we created, we can now reconstruct the full list of links the menu was pointing to, however we can't retrieve the text of the menu items (also because it's sometimes replaced by images). So, how can we

know in advance where the link we've extracted will take us?

The following script is a _lookahead_ link extractor: it does the dirty work instead of us, following the links it finds inside the Flash file, extracting the title from the Web pages it visits (if it finds any) and using it to rebuild an HTML menu.

-- Flash Lookahead Link Extractor

```
#!/usr/bin/perl

use LWP::Simple; # used to get linked pages
use URI::URL;    # used to absolutize URLs

sub rel2abs {
        my ($rel,$base) = @_;
```

```
        my $uri = URI->new_abs($rel, $base);
        return $uri->as_string;
}

$url   = $ARGV[0];
$flash = get($url) || die "Couldn't download $ARGV[0]";


# SYNTAX IS: 0x00 0x83 0xlen 0x00 "string" 0x00
while ($flash =~ /\x00\x83.\x00(.*?)\x00/gs){
  my $nextitle;
  my $link = rel2abs ($1,$url);
  my $nextpage = get ($link);
  if ($nextpage =~ /<title>(.*?)<\/title>/i){
    $nextitle = $1;
  }else{
    $nextitle = $link;
  }
  print qq|<a href="$link">$nextitle</a><br>\n|;
}
```

This source is a little more complex than the previous one, but I assure you there's not much more to know to understand it: in the meanwhile, just be aware that the usage of the script from the shell is the following:

perl flash2.pl http://url.you.like/menu.swf

Note that, in this case, we directly pass the URL to the script instead of using lynx: in this way, the program will be able to use the base menu URL to replace all the relative links it finds inside the flash file into absolute ones. This operation is done by the "rel2abs" sub, which uses a ready made command from the URI::URL package. The other package I used is LWP::Simple, a package you'll see more in detail later: for now, just know that it gives you the useful "get" command, which just needs a URL to return you the HTML source of the matching Web page. Inside the script, this source code is saved inside the $nextpage variable, which is matched with the regular expression to extract the page title whenever one exists.

5. Bot Basics

During our war against Flash we've seen the first Perl robot (or more simply bot) example: it browses inside a website for us, following the links it has extracted from a flash menu, and it rebuilds the same menu in HTML, adding to every link the corresponding Web page title.

But what identifies a bot and what should one of these programs be able to do? Well, despite of many historical or rigorous definitions, a good Web bot should at least provide the typical spider and scraper functions: that is, it should be able to move inside a website (following links, filling forms, authenticating itself and so on) and to download full Web pages or just part of them, extracting the most interesting information and thrashing everything else.

Of course a bot isn't able to do everything alone, such as recognizing "what is interesting" or following some links instead of others. The robot, in general, is a machine inside which the man does what he wants: so, even in this case, we first need a human to first visit the websites the bot should be able to browse and to decide what it will consider interesting and what it will have to ignore.

5.1 Detecting Web Patterns

Human intervention while creating a bot is fundamental: this is because the machine isn't able to directly work on reality, but just works on a model, built by man, which is based on a personal interpretation of reality. For instance, while we can recognize, inside forum messages, a message subject (that is, we immediately perceive the page _semantics_, regardless of how it is coded), a bot can just understand that the subject is between some pieces of HTML code (that is, it works on a _syntactical_ level, strictly dependent on how the page is coded).

Of course, all the work which is being done on the "Semantic Web" should allow machines to access the Web more easily. Also, the use of AI-powered wrappers could give some good results... but this is another story: for now, keep in mind that, for some time, you'll still have to do a good part of the work.

What are the Web Patterns we named inside the title? They're nothing more than particular schemas you can detect inside one or more Web pages' contents, or inside the paths we're used to follow when, clicking from one link to another, we move inside a website. They're exactly what we need to understand, if we want to teach a bot to do what we usually do, that is

1) visiting a website and
2) extracting information from it

In the first case, the question we have to answer is: is there a way to make a program follow some links automatically? Of course there is, and the proof are all the softwares which can mirror a website. But is it also possible to

make a bot CHOOSE some links, and if so which ones?

In some cases the answer is easy: for instance, for "wget-powered porn" we just asked to follow all the links and download all the videos found at a depth of two links. This worked because we knew that the structure of the websites we wanted to use wget with was the following one:

```
 Website containing      1    Website containing    2     Video
links to other sites --------> links to videos  --------> Files
```

In other cases the answer is less trivial: for instance, we might have to dowload all the images from a gallery where every thumbnail has a link to the matching image, and in every page there's a link to the following one, but we don't know how many pages we have to follow. In this case, we should tell the bot to collect all and only the files matching the desired images, and to follow the links to the next pages until they find them, or until they find some kind of terminator (for instance, a link to the first page of the gallery).

And what if we wanted to save all the messages which reside in a forum? In this case, we would probably have a structure which is very similar to a gallery, but with a deeper hierarchy (Forum->Threads->Messages) and without a file extension helping us to recognize the information we want.

As you can understand from these examples, being able to recognize some navigational patterns might help you much while developing a bot. Of course, as with advanced PowerBrowsing techniques, some experience is needed here too. Even if they're implemented in different ways from one website to another, navigational patterns are more or less always the same: this means that the more experience you gain the easiest your job will be.

As I told you before, we don't just have navigational patterns but we can also find some schemas occurring frequently inside page contents, in a single website or even inside different ones. Being able to find these patterns allows you to teach your bot how to extract, from each page, only the information you're really interested in.

For instance, from a generic forum HTML page (that is, the one containing the message) we'd probably want to keep only the sender, the subject, the date and the message body. In the full page, instead, you can usually find images, banners, HTML code for table generation and so on, until you get documents which can even become 100 times bigger (it's true, I've counted the bytes!). However, if we find that before and after the pieces of information we're interested in there's always the same HTML code, then we can create wrappers which can recognize and extract them.

If with old hand-made websites we could just suppose -but we couldn't give for granted- some kind of repetitivity inside HTML code, inside dynamic sites all the fixed code is SURELY always identical, because it's generated by a computer. This helps us much when we create a wrapper, because we just have to see the HTML code of one generic page (for instance, one forum message) to find all the parts which never change.

In some cases, there will be some little differences between "particular" and "general" pages, but still you'll just have to see an example for each one of them and then be sure the same patterns will work in all other cases. To make it more clear with an example, think about "unanswered forum messages" and "forum messages which have a reply": their pages might differ slightly, so you'll have to watch both of them, but once you understand which code remains constant, you'll be able to build wrappers which work for both of these two page classes.

As I told you before, thanks to the diffusion of many standards we can now easily find patterns not only inside a website, but also from one site to another one: think about news in RSS format, or all the blogs which even provide an API to access them, or all the different forum softwares which are written in different languages but have the same structure and often generate almost identical HTML code.

When we're finished identifying patterns, we have to teach our bot how to replicate them (in the case of browsing patterns) or to detect them (in the case of wrappers). Inside the next two sections you'll see some thoughts, still independent from programming languages, which might help you to design your bots.

5.2 Website navigation with bots

For what concerns browsing, these are the operations a bot should be able to easily do:

- given an URL, download the matching page (this task, as you'll see in the next chapter, is trivial, especially in Perl)

- link extraction, based on some particular conditions inside the link URL: for instance, the filename has to end with .gif, or all the files have to be inside /textfiles/ directory or have to be generated by the php script viewthread.php?f=1&thread=... This task can be easily done using regular expressions

- link extraction, based on some particular conditions inside the tagged

text: for instance, we might want to choose only the links which are shown as "Next", or the ones which start with "Document". Regular expressions help much in this case too

- follow extracted links until a particular depth level: for instance, follow all the links you can find in a long list, and download all the images you can find in all of the visited websites

- follow extracted links forever, or until a particular condition: for instance, continue "clicking on Next" until this kind of link is present

- collect all the extracted links to use them in a second time, from the last visited page or from all the downloaded ones: for instance, it should be possible to follow links for three levels of depth and then collect all the links to images, or follow all the "Next" ones inside a forum and, for EACH page we found, collect all the links to messages

If you ever decide to create a Perl bot, keep in mind that most of this work has been done by me yet (or by someone else I don't know, maybe before me, probably better too). Inside the chapter devoted to examples, you'll find the description (and the link to source code) of a package, called "common", which contains exactly the functions needed to complete these tasks.

5.3 Data extraction

For what concerns data extraction with bots, the technique I've found most easy and powerful is the one which uses Regular Expressions. There are many other ways to parse and extract information from HTML files, such as XPath or text parsing libraries. In the next chapter I'll name these libs, but I won't write about them in detail: if you want, you can find more information about them inside "Spidering Hacks"... or around the Web.

Regexps allow you to "cut" the text slicing it in chunks: with a RE like

```
/<initial conditions>.*?<final conditions>/gsi
```

where

- the initial and final conditions are HTML code which surrounds what you are interested in

- .*? is a regular expression which matches (that is, which is satisfied by) the smallest text chunk between the two conditions

- the gsi "modifiers" allow you to make an iterated (g), case-insensitive

(i) search, considering all the text as a single line (s)

you can

- divide a text in chunks, for instance a thread in the messages which are part of it or a long list of generic elements in the single elements (HEAD-TAIL wrapper)

- extract one or more text strings from a line (LEFT-RIGHT wrapper)

Since websites are subject to frequent changes (fortunately, the dynamically generated ones aren't updated so much), creating your wrappers parametrically is always a good choice: that is, they should be able to extract text depending on some strings, which you can change from time to time, containing the regular expressions you want to use. When the website changes, you'll just have to change the regexp and your bot will work again (for an example, see the one about cinemas inside Chapter 8).

6. Perl PowerBrowsing Tools

Inside this chapter we'll speak about PowerBrowsing tools written in Perl. Here, you'll learn how to create bots which connect to websites, downloading what you want and extracting all the information you consider interesting, and all of this while automatically managing identification with servers, authentication, cookies and referrers.

The first section of this chapter explains why I chose Perl as a programming language for bots. In the following ones, you'll have a glimpse of which Perl libraries you can use to create Web-capable programs, then a little more in-depth analysis of the main objects and methods you can use to create your first bots.

6.1. Why Perl?

Why should you use Perl to create bots? Well, the expressive power of many programming languages is almost the same now, but of course everyone is still different from others because of some details, which can be more or less important depending on what kind of application you want to create.

It's one of these details (and the fact I like this language much) which made me choose Perl: its extremely powerful text manipulation functions let you create in few seconds wrappers which, with other languages, might have been harder to build.

To this we have to add the great availability of ready made libraries, which helps us much (as you'll see later) to evolve our simple data-retrieval bots into programs which _share_ information in the most common formats. Moreover, some of these libraries are so easy to use that everybody will be able to program with them.

Finally, we should consider two other important Perl features: first of all, the portability, which lets you make experiments with bots independently from the operating system you use; then, the presence of documents and tutorials about this topic (and I must say that "Spidering Hacks", in this case, has been fundamental giving me hints and new ideas).

Of course, these reasons I used to explain why I decided to use Perl should not prevent you from using another language. Instead, I'd be glad to see other bot implementation and check if they're more or less easy or powerful than the ones I'll describe you later.

6.2. Perl Packages

Perl has many libraries for Web access and for the management of the main objects your bots will have to deal with. Even if you'll probably have to use only a couple of this libraries frequently (LWP::Simple and LWP::UserAgent), here's a small description of the other more commonly used ones:

LWP
: Also known as libwww-perl, it's a collection of modules for Web access

LWP::Simple
: It's the easiest package you can use to download pages from the Web: between the functions it exports, there are the classic "get" (which you have already seen in section 4.4), "getprint", which automatically prints what it downloads, and "getstore", which saves downloaded documents inside a file

LWP::UserAgent
: It's a more advanced library for Web access (we'll see it more in detail later)

HTTP::Request
HTTP::Response
: These are objects used to manage server requests and responses with a higher level of detail

HTTP::Message
HTTP::Headers

These are classes which offer more methods to HTTP::Response

URI
: It's a class which exports methods to operate on Web addresses, for instance to make a relative URL absolute, or to extract domain name or path elements from a link

URI::Escape
HTML::Entities
: These export methods for escaping and unescaping, respectively, URLs and text extracted from an HTML documentm, allowing your programs to manage non-standard characters such as spaces inside URLS or hyphenated letters inside HTML code

HTML::TokeParser, HTML::TreeBuilder, WWW::Mechanize
: The first two classes are specialized in HTML parsing and offer you an alternative way to regular expressions to extract information from Web pages; the last class allows you to automatise the interaction with websites, filling forms, following links and so on. Together these classes might help you much making your bots more simple and stable, but (at least now) they're not described in detail in this text.

6.3. LWP::Simple

LWP::Simple is the easiest package you might use to connect to the Web with Perl. It exports the "get" command, which downloads the page at the specified URL and returns its content, ready to be saved in a variable. The syntax is

```
$content = get ($URL);
```

Once in a variable, you can apply a regular expression to the content, checking for some conditions: as an example next script will, in few lines of code, tell you if when you're executing it there's a Radio Bandita stream online.

```
use LWP::Simple;                # this is the package you want to use

my $url = 'http://radio.autistici.org/';# this is the URL we'll check

# download $url and save its contents in $content
my $content = get($url);

# exit if an error occurs
die "I couldn't download $url" unless defined $content;
```

```
# now search inside $content
if ($content =~ m/bandita/i) {
    print "Radio Bandita is streaming right now!\n";
} else {
    print "No good music online.\n";
}
```

To have a slightly more useful (and surely more international) example, give a look at this more advanced script:

```
use LWP::Simple;                    # this is the package you want to use

my $search_term = $ARGV[0];         # get the seach term from the command line
die "Specify a search string, please!\n" unless defined $search_term;

# this is the base URL of the search script, with the search term appended
my $url = "http://www.shoutcast.com/directory/?s=$search_term";

# download $url and save its contents in $content
my $content = get($url);

# exit if an error occurs
die "I couldn't download $url" unless defined $content;

# now search inside $content

if ($content =~ m/any SHOUTcast streams found/) {
    print "No shoutcast streams containing the term $search_term\n";
} else {
    print "Some shoutcast streams contain the term $search_term.\n";
    print "Here are the songs they're playing:\n";
    while ($content =~ /Playing:<\/font>\s(.*?)<\/font>/gs) {
        print $1."\n";
    }
}
```

In this script, we get a search string from the command line (as we did with the Flash lookahead link extractor before) and then we append it to Shoutcast search engine URL: if the answer contains the string "any SHOUTcast streams found", then no streams contain the searched word; otherwise, we can just apply a simple left-right wrapper to the page source and extract all the songs which are played by the streams which contain the searched term.

LWP::Simple gives you some more functions, such as "getprint" and "getstore", whose usage is more specific and for this reason less frequent than simple get. However, getprint can be particularly useful for oneliners in case you don't have lynx installed in your system: in fact, the command

```
perl -MLWP::Simple -e 'getprint "http://3564020356.org";'
```

works the same way as

```
lynx -source http://3564020356.org
```

6.4. LWP::UserAgent

LWP::UserAgent is a more advanced and, unfortunately, complex package than LWP::Simple, but it offers many other functions which you might need if you decide to build a complete, professional bot. Thanks to this package, you'll be able to identify your UserAgent with the server, add the "Referer" field between request headers, POST data to a form, manage cookies and connect through a proxy. And these are only some of the things you can do with it!

-- LWP::UserAgent - 01 - Basics

```
#!/usr/bin/perl -w
use LWP 5.64; # use LWP and check that the version is recent enough

my $url = 'http://radio.autistici.org/';

# create a new useragent
my $ua = LWP::UserAgent->new;

# GET the specified URL
my $response = $ua->get($url);

# if an error occurs, exit
#(NOTE the more advanced error management)
die "I can't download $url: ", $response->status_line
    unless $response->is_success;

# now, search inside page content
if ($response->content =~ m/bandita/i) {
    print "Radio Bandita is streaming right now!\n";
} else {
    print "No good music online.\n";
```

```
}
```

The script you've just seen is the easiest program example you can write with LWP::UserAgent: it's an evolution of the previous script which was written with LWP::Simple. It's not much more complex than the original one, but we can see a more advanced organization yet: first of all, a UserAgent object which will manage all the communications with the server is created; then, the UA is asked to GET the specified URL and to save it inside $response variable. This variable doesn't contain only the HTML page content anymore, but it's a real object which can tell you if the operation went fine or not (with is_success method) and give you the page content (with content) or the error code (status_line).

The "get" command itself, used in its easiest version, allows you to specify many other parameters: for instance you can add custom headers, containing the identification string of you UserAgent, the file types you accept, the character set and the language you prefer. To do this, you just have to save these data inside an array and to pass it as the second parameter of "get" method:

-- LWP::UserAgent - 02 - Identify your bot

```
my @ns_headers = (
   'User-Agent' => 'MaLaBot 1.0',
   'Accept' => 'image/gif, image/x-xbitmap, image/jpeg,
               image/pjpeg, image/png,  */*',

   'Accept-Charset' => 'iso-8859-1,*',
   'Accept-Language' => 'en-US',
);

$response = $browser->get($url, @ns_headers);
```

Some websites might refuse to accept data from you, unless you come from a particular URL: to check this, servers give a look at your referer, so you might need to change it to a desired value. To do this, you have to work on "request" object: first you should create it specifying which kind of request (GET or POST) you want to send; then, you should add the referer with the "referer" method. The following example shows you the steps you have to do when you POST data to a form (in this case, $POST_URL and $REFERER_URL are variables which contain, respectively, the URL you want to send your request to and the URL you've decided you are coming from).

-- LWP::UserAgent - 03 - custom referers

```
# create request object, passing method
my $req = HTTP::Request->new(POST => "$POST_URL");
   # this is used for post
   $req->content_type('application/x-www-form-urlencoded');
   # line to post
   # this is an example string which will be sent to a search engine
   $req->content("lang=it&descrizione=$letters&numero=&CheckExt=N");
   # referer URL
   $req->referer("$REFERER_URL");

my $res = $ua->request($req);
```

In the previous example you've seen how to POST form data. In particular, we have used a string (the one specified with the "content" method of the "Request" object) of concatenated elements. However, there's another way to do the same thing: you can give the "post" method of the "UserAgent" object, together with the destination URL, a hash (that is, an array containing key/value pairs) with the parameters you want to send.

-- LWP::UserAgent - 04 - post form data

```
my $ua = LWP::UserAgent->new;

my $url = 'http://url.of.the.form/path/form.php';

my $response = $ua->post( $url,
   [ 'param1' => $value1,
     'param2' => $value2,
     'param3' => $value3,
     'param4' => $value4,
   ]
);
```

Remember (I had to bonk my head over the monitor for a while before I found this) that often, after a POST, you might be redirected to another page. To have your UA return the page you're interested in, and not the one which just contains the redirection to the page you want, you'll have to insert the following line before your request:

```
  push @{ $ua->requests_redirectable }, 'POST';
```

This line tells the UserAgent that POSTS are requests for which the bot needs to automatically follow redirects.

For your authentication inside some websites (or even for a simple access to

other ones) your UserAgent should be able to handle cookies. Nothing easier: you can use one of the following methods and activate UA cookie management.

-- LWP::UserAgent - 05 - use cookies

```
use HTTP::Cookies;

$ua->cookie_jar( HTTP::Cookies->new( # or HTTP::Cookies::Netscape
    'file' => '/some/where/cookies.lwp',  # where to read/write cookies
    'autosave' => 1,                      # save it to disk when done
));
#------------------------------------------------------------------------
$kj = new HTTP::Cookies;
$kj->load("cookies.txt");
$ua->cookie_jar($kj); # activate cookie jar for the useragent
```

Last, but not least (if you're working in an office and you're behind a proxy these lines will be of primary importance for you), the UAs you create with LWP::UserAgent can connect through proxies. You can use the proxy set as an environment variable (env_proxy), so you don't have to reconfigure it, or you can setup a new one, and even choose the protocols you want to use it for.

-- LWP::UserAgent - 06 - connect through a proxy

```
$ua->env_proxy;
#------------------------------------------------------------------------
$ua->proxy(['http', 'ftp'], 'http://proxy.sn.no:8001/');
```

That's all, for now: these examples aren't complete for sure, nor they want to answer all the questions you might ask yourselves when you decide to build your first bot. However, I think that they might make your life easier, at least for the first experiments, and I'm sure that working on them for a while (and, why not, maybe cooperating with someone else, if not with me) you will get great results.

7. Share your data

Now that you can create bots that can go around recovering data for you, the best thing you could do is to find a way to share these data with other people. Of course, this is a choice that everyone should do spontaneously, together with the way you decide to make this sharing (from floppy disk to telepathy, everything is allowed!)

In this section I'd like to give some hints to those among you who would like to share the data harvested with their own bots, but still have no idea of how they can do it.

7.1 Web and RSS

The minimal sharing example I can usually think about is the screen of my computer in text mode, an idea I can replicate quite well by creating Web pages whose contents are enclosed in <PRE> and </PRE> tags.

Without the need to reach this extreme point, you can create light websites, maybe dynamically generated with PHP, which allow you to share your data without losing usability: a little search engine and some links to go from one part of your data to another one will require a relatively small time for the implementation, but they'll add much to the usability of the data you expose.

Alternatively, you can use the Perl package XML::RSS to publish your contents in RSS: while this operation is absolutely trivial for you, it gives a great service to the user, letting him choose the client he likes most to browse the data.

7.2 Mail

When I talk about choosing a client, I can't but think about my good, old, comfortable and hyperused mail client: if it could be consumed, mine wouldn't even exist anymore now! It's become so comfortable to me that, even if it isn't supported anymore, I still continue using it. And, being a Windows app, it's one of the few reasons that still make me create dual boot PCs.

So, if other users are comfortable with their mail clients as I'm with mine, why don't we share our data with them through email? I remember some services (now they're fewer and fewer, but there still are some) that allowed you to see Web pages by mail; others periodically visited a website to check for updates and sent notifications to subscribers; I have created a perl script that, called by procmail whenever it gets a message with a particular subject, sends an email with a list of all the films being played in my city.

There are different ways to communicate data via email: one, perhaps more complex if you don't have a server available, is to use procmail to filter incoming messages and automatically answer, according to particular text strings in the subject or in the message body; aonther one, instead, is to create a virtual POP3 server like the one described in section 3.3. If you want to give a look at some perl source code, you can download my old ANO app from http://3564020356.org/tools/ano099beta.zip.

7.3 Net::Blogger

Among the various techniques you can use to publish your data, one of the easiest and most versatile ones is using a blog. It's easy because you just have to use a ready made library, which uses APIs from most common blogs. It's versatile because, when you publish data on a blog, you're making them available not only in HTML, but also in RSS format; you have them saved in a DB where you can make queries; also, you probably won't even need to own the server which hosts them, because there are so many which give you a free blog now; finally, you'll be able to use these satanic apps for a USEFUL purpose, at last!

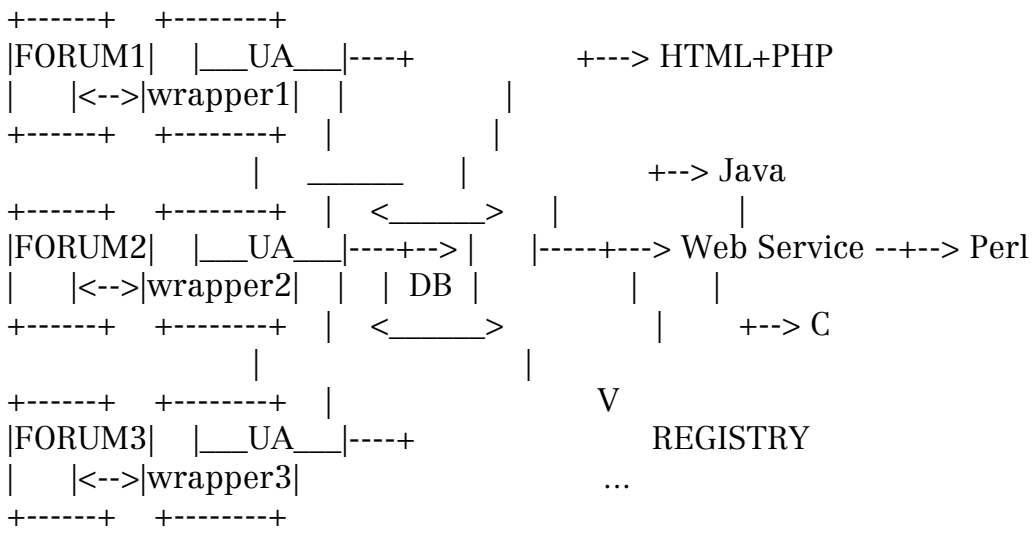
I will not dwell on explanations about this package: you can find all the docs online and the source is self explanatory. Keep in mind that the part related to blog publishing is all in the last twenty rows, while previous ones deal with variable declaration and scraper code (which, in this case, is the same I used inside "Things I've learned from B-Movies" example). If you change the first lines of the script, inserting your blog's API's URL, its name, your login and password, you'll have in few seconds your first working Bloggerbot.

```
#!/usr/bin/perl

use Net::Blogger; # this is used to post articles
use common;       # this is used for LWP related functions (getpage, exturl)

my $debug  = 1;
my $PROXY  = 'http://site.of.your.blog/blog/nucleus/xmlrpc/server.php';
my $BLOG   = 'myblog';
my $LOGIN  = 'login';
my $PASS   = 'password';

my $TITLE  = "Things I've learned from B-movies";
my $CATEG  = "bot";

#-----------------------------------------------------------------
# this is the scraper code

my $BADMOVIES_URL = 'http://www.badmovies.org/movies/';
my $LINK_FORMAT   = '/movies/.*?/index.html';
my @quotes;
```

```
my $idx_content = getpage ($BADMOVIES_URL);
my @movies = exturl ($idx_content,$LINK_FORMAT,'',$BADMOVIES_URL);
my $movies_size = @movies;
my $randurl = $movies[rand($movies_size-1)];
my $mov_content = getpage($randurl);

if ($mov_content =~ /<title>\s*Review for (.*?)\s*\n/si){
        $title = $1; chomp $title;
}

if ($mov_content =~ /learned\.gif>(.*?)<\/font><br>/si){
        my $learned = $1;
        while ($learned =~ /10>\s*(.*?)\s*\n/gsi){
                push @quotes,$1;
        }
}

my $quote_size = @quotes;
my $quote = $quotes[rand($quote_size-1)];
$DATA  = qq|$quote<br>|;
$DATA .= qq|(<a href="$randurl">$title</a>)|;

#---------------------------------------------------------------
# this is the blogger code

$blogger  = Net::Blogger->new(debug=>$debug);

$blogger->Proxy($PROXY);
$blogger->Username($LOGIN);
$blogger->Password($PASS);

# get blogid and assign it to the blogger
my $blogid = $blogger->GetBlogId(blogname=>$BLOG);
$blogger->BlogId($blogid);

# create post text
my $txt  = "<title>$TITLE</title>";
   $txt .= "<category>$CATEG</category>";
   $txt .= "$DATA";

# send and publish the new post
my $id = $blogger->newPost(postbody=>\$txt,publish=>1)
        || die "Error: ".$b->LastError();
```

7.4 TWO

TWO (The Working Offline forum reader) isn't a very active project at the moment, but I worked a lot on it last year and now it enters with full merit among advanced PowerBrowsing techniques. Thanks to a very modular structure, based on plugins, it allows you to download the contents from web forums that have different technologies (at the moment four different forum types are supported, for which google returns me some millions of hits), to save them inside a unified database and to read messages with a Web (HTML and PHP) interface or to share them via Web Services (with ready made clients, even if they are minimal, in Java, Perl and C).



One of the main advantages of TWO, beyond the possibility to share downloaded information with other people, is that it merges data coming from websites with different technologies inside a single database: in this way you can find, with a single search, messages written in different forums. Moreover, the disk space required by the information extracted from the forums is a very small part of the downloaded data, which are just a very small part of what a normal browser would have downloaded.

How much disk space can save TWO, exactly? The following are the results of some tests (you can find the complete version inside TWO's documentation):

Forum data size (KB) 92741

TWO's data size (KB)   7892

Saved space.(KB).....84849
Saved space (perc).....91%   <= !!!
==========

The saved space, of course, is not only an advantage for you, but also for all the people that will download that data from your computer. For instance, if people started to share forums, you could just compress that 8MB database to obtain a file the size of a floppy disk, so everyone in few minutes (or seconds!) could download it and enrich their DB with lots of new information. Using a more advanced technology, if many TWO Web Services shared different databases through a registry, every Web user could make distributed searches on different forums at the same time.

Of course, TWO can still be improved much and the errors that should be corrected are, probably, still many. The source code is provided "as is" and you'll probably need some time to understand how it works and how it can be improved. However, if you are interested, you can download TWO's source code and documentation from http://two.sf.net. Let me know if you can make something good out of it ;)

8. Examples

In this section you'll have the chance to see and try some examples. To save space, I've decided not to publish their source code here, but to insert a link from which you can download them. If you can't connect there, you can send me a mail and I'll answer you with an alternative URL.

- Common lib
  http://3564020356.org/cgi-bin/perlcode.pl?file=common.pm

  common.pm is a package I created when I was developing TWO. It contains all the main functions I used to control a web bot behavior:

  getpage is very similar to LWP::Simple "get" command, but it uses LWP::UserAgent instead to obtain some more advantages: it can manage cookies, a proxy, UserAgent identification and multiple retries before it aborts a download.

  exturl collects, inside an array, all the links it can find inside a Web page and which satisfy one or more regular expressions inside the URL or the tagged text: this allows you to follow links such as "all the files whose name end in .txt" or "all the links whose text matches 'Next'".

walkpages is a recursive function which uses exturl to follow a list of links and collect a list of others. It can work in different ways: it can follow different links depending on the depth and it can collect only the ones it finds in the last page, or all the ones it finds during its travel.

walkpages_loop is the "looped" version of walkpages: that is, it follows the same (or the same list of) links undefinitely (or for a specified depth) until it finds results, collecting all the matching links it finds.

- Cinemaz
http://3564020356.org/cgi-bin/perlcode.pl?file=cinema.pl
NOTE: it uses common.pm

Cinemaz is a program I created for personal use, to automatically gather data from http://www.monzacinema.it. This website shows all the movies you can find this week in different cinemas, but if you want to choose a particular movie, know where and when you have to go to see it, and find a phone number to book a seat, well you have to click far too many times!

So, this bot connects to the main page and downloads all the pages which describe the cinemas, extracting their name, the phone number, the movie name and the timetable. All the information are shown in a good old plain text file, which has everything you need... and only it.

With some little adaptations I managed to use the same script with procmail and now, wherever I am, I just need to send my bot a mail with "cinemaz" as subject to have an answer containing the very same text file :)

- Things I've learned from B-Movies
http://3564020356.org/cgi-bin/perlcode.pl?file=badmovies.pl
NOTE: it uses common.pm

http://www.badmovies.com is a very funny website, containing many B-movies reviews. One of the funniest things, IMO, is that inside every movie page there's a section, called "Things I've learned from B-Movies", with a list of fortunes about (silly) things you can learn from that movie.

When you run the script it downloads the movie list with their links, it chooses and follows a random one, then it extract all the quotes from the "Things I've learned from B-Movies" section and finally, depending on how you ran it, it shows all of them or just a random one, like the "fortune" application.

- Malacomix
  http://3564020356.org/cgi-bin/perlcode.pl?file=comics.pl

  Malacomix connects to http://www.comics.com and lets you see, every single day, your favorite comic strips. Since it uses the common syntax used by the website to find all the different strips, you just have to write inside the URL the names of the strips you want to see and it will generate an HTML page containing all the links to the matching images.

- Happy3 URL Extractor
  http://3564020356.org/cgi-bin/perlcode.pl?file=happy3.pl

  This script has been created with a particular purpose: to let everyone see Happy Tree Friends episodes _the way they like_ (and not in a fixed-size popup window) or download them on their hard disk. The script is very, very easy, but it followed a more advanced "flash reversing" work. Maybe one day, when you are not so tired because you've read this loooong text, I'll explain you this one too ;)

-mala