CHAPTER 1
Installing mod_perl

Introduction

The first step to a successful mod_perl server is the installation of mod_perl itself. As you will quickly find out, mod_perl is incredibly powerful and flexible in all respects—as with Perl, there is always more than one approach, and installation is no exception.

This chapter gives you the knowledge and resources you need to successfully build, verify, and understand a basic mod_perl server installation. This is important not only for the system administrator who may be responsible for maintaining an installation, but also for the developers who write applications against it. It is also helpful to have a working mod_perl installation to try out the recipes and example code throughout this book.

For the most part, a mod_perl installation consists of two parts: the mod_perl enabled Apache server, and the Perl modules required to support various mod_perl functions. At the end of a typical installation, the Apache side of things will include an `httpd` binary, as well as the all important `httpd.conf` configuration file and the user documentation residing under *`ServerRoot`*`/manual/`. The `httpd` binary may include all the components required for mod_perl from Apache's point of view, or mod_perl may be found in the shared library `mod_perl.so`

that is dynamically incorporated into Apache at runtime. In either case, the end result is a fully functional and persistent perl interpreter embedded into the Apache server.

Having a working perl interpreter is only part of the story. Normal mod_perl operation requires a number of different Perl modules, such as `Apache::Registry` and `Apache::Constants`. These are installed into the `site_perl` directory of your Perl installation, alongside of the various other third-party Perl modules you may have installed.

These two parts are sometimes referred to as the "Apache side" and the "Perl side" of a mod_perl installation. The important thing to understand about this symbiotic relationship is that mod_perl ties in to both environments—because it joins Perl modules with the Apache runtime, it requires a presence in both architectures in order to function properly. If the distinction is not clear now, hopefully it will become clearer as you delve into the mod_perl API in Part II.

With that bit of background behind us, we can progress to the topic at hand. To get started using mod_perl you must first obtain a mod_perl-enabled Apache server. Often, the fastest way to a working server is by enabling a binary distribution from your operating system vendor. This way is useful if you are not accustomed to compiling software from its source code, or are just interested in a functional server to experiment with. Installation of a binary distribution is usually as easy as copying a few files or editing a configuration file.

To get the highest performance, and the ability to tune your server at the most granular level, you will want to compile from the mod_perl and Apache sources. Compiling your own mod_perl-enabled Apache server gives you the ability to customize your installation. While it is somewhat more complex to compile your own server, as you will see in later chapters, the extra effort spent perfecting installation will reap benefits later on.

1.1. Unix Binary Installation

You want to install a binary version of mod_perl on a Unix platform.

Technique

Determine your Unix variant and refer to the following platform-specific instructions.

Comments

Many binary distributions of mod_perl are available for Unix. Each Unix vendor or distribution has its own way of packaging and distributing binaries. This recipe covers the most commonly used Unix distributions. Read on for installation instructions for Linux, BSD variants, and Solaris.

At this time there are no known binary distributions of mod_perl for most other Unix platforms (AIX, HP-UX, IRIX, etc.) Consult your operating system manuals for any mod_perl packages that may have been recently added. Alternatively, see Recipe 1.4 to learn how to install mod_perl from its source code. Keep in mind that binary packages may not have been compiled with `EVERYTHING=1`, so compiling from source may be preferred when testing recipes from this book.

RPM-Based Linux Distributions

Many Linux vendors, such as RedHat, SuSE, Mandrake, and Caldera, distribute binaries via the RPM (RedHat Package Manager) packaging format. You can add mod_perl to your system by having `root` access and a copy of the RPM package for your system. The first step is to find a copy of the appropriate RPM file on your installation media. You may see filenames like the following:

- **RedHat:** `mod_perl-1.24_01-3.i386.rpm`
- **SuSE:** `mod_perl.rpm`
- **Mandrake:** `apache-mod_perl-1.3.19_1.26-3mdk.i586.rpm`
- **Caldera:** `mod_perl-1.24-2-i386.rpm`

If you cannot find an RPM on the installation media, try the vendor's Web site, or `http://www.rpmfind.net/`.

Most RPMs are based on Apache's Dynamic Shared Object (DSO) support. This support allows the vendor to ship a generic Apache RPM with add-on module RPMs for Apache extension modules like mod_perl. Be sure you have the base Apache packages installed in this case. A simple

```
$ rpm -qa | grep -i apache
```

will find any Apache RPMs already installed. In most cases, if you do not have an Apache RPM you will need to install one before installing the mod_perl RPM.

DSO, while great in concept, has had many problems in the past with memory leaks and other oddities. Thus, third-party RPMs that include mod_perl as a statically compiled module in the Apache binary (and override your vendor's version) are also available. See `http://perl.apache.org/download/binaries.html` for a canonical list.

After you have the RPMs, installation is as easy as

```
# rpm -ivh mod_perl-1.24_01-3.i386.rpm
Preparing...                ########################################### [100%]
   1:mod_perl               ########################################### [100%]
```

and uncommenting the `LoadModule` and `AddModule` directives in the `httpd.conf` present on your system.

Debian GNU/Linux Distribution

Debian packages mod_perl as part of a special Apache package called `apache-perl`. This version of Apache contains mod_perl statically compiled into the Apache server. The easiest way to obtain this version is with the `apt-get` program:

```
# apt-get install apache-perl
```

Consult the `apt-get` documentation for more information about downloading and installing Debian packages.

Other Linux Distributions

If you are running Slackware, Stampede, or some other Linux variant, you may be interested in the Alien package converter. Available from `http://www.kitenet.net/programs/alien/`, Alien can convert packages from one packaging format to another. This may allow you to use the RPM or Debian packages.

BSD Variants

FreeBSD and OpenBSD users have two easy ways to install mod_perl. The simplest way is to use a precompiled third-party binary package. A more complex method involves using the *Ports* system to automatically compile and install mod_perl.

It's easy to install a binary package. Browse the pages at `http://www.FreeBSD.org/ports/` or `http://www.openbsd.org/ports.html` to find

and download the mod_perl binary package for your flavor of BSD. Once downloaded use the `pkg_add` utility to install it:

```
# pkg_add mod_perl-1.26.tgz
```

You can also use the `-r` option to `pkg_add` to automatically find and download mod_perl, like this:

```
# pkg_add -r mod_perl
```

If you would rather automatically compile and install mod_perl, use the *Ports* system. You will need to have an updated ports tree. This is often found at `/usr/ports/` on most FreeBSD systems. The following output shows the commands that build the mod_perl package and install it for use.

```
# cd /usr/ports/www/mod_perl
# make
>> Attempting to fetch from ftp://gatekeeper.dec.com/pub/plan/perl/CPAN/
➥modules/by-module/Apache/.
Receiving mod_perl-1.26.tar.gz (372859 bytes): 100%
372859 bytes transferred in 3.5 seconds (105.24 kBps)
===>  Extracting for mod_perl-1.26
>> Checksum OK for mod_perl-1.26.tar.gz.
===>   mod_perl-1.26 depends on file: /usr/local/sbin/apxs - not found
===>    Verifying install for /usr/local/sbin/apxs in /usr/ports/www/apache13
===>  Extracting for apache-1.3.20
>> Checksum OK for apache_1.3.20.tar.gz.
===>  Patching for apache-1.3.20
===>  Applying FreeBSD patches for apache-1.3.20
===>  Configuring for apache-1.3.20
Configuring for Apache, Version 1.3.20
 + using installation path layout: FreeBSD
(/usr/ports/www/apache13/files/FreeBSD.layout)
...

# make install
```

Solaris Binary Packages

Sun provides a prebuilt version of Apache compiled with mod_perl in recent versions of Solaris, beginning with Solaris 8. Three packages named `SUNWapchd`, `SUNWapchr`, and `SUNWapchu` are all you need to get mod_perl on your system. You'll find these packages

on the second Solaris 8 software CD-ROM. To install, insert the CD-ROM and execute the following commands:

```
# cd /cdrom/cdrom0/Solaris_8/Product
# pkgadd -d . SUNWapchd
# pkgadd -d . SUNWapchr
# pkgadd -d . SUNWapchu
```

Once done you'll find the installed files in `/usr/apache/`.

Final Touches

After you have installed your package of choice, you can verify that mod_perl is active in your Apache server by following the instructions in Recipe 1.8.

1.2. Windows Binary Installation

You want to install a binary version of mod_perl on Microsoft Windows.

Technique

Download and install the complete Perl binary package or the PPM package.

Comments

Some Perl binary distributions contain mod_perl packaged with them (see `http://perl.apache.org/download/binaries.html` for a link to one such package, as well as `http://www.indigostar.com/` for another). These types of distributions contain relatively detailed installation instructions and also include a collection of popular modules (such as `LWP` and `Net::FTP`) not included in the standard Perl distribution.

For users of ActivePerl or compatible `perl` binaries, mod_perl PPM (Perl Package Manager) packages are available, as well as PPM packages for some other Perl modules often used with mod_perl. For a partial list of links to these PPM packages, see `http://perl.apache.org/download/binaries.html`. These can be installed in one of two ways—directly from the command line as

```
C:\> ppm install http://ppm.example.com/ppmpackages/mod_perl.ppd
```

or, from within the ppm interactive shell, as

```
C:\> ppm
ppm> set repository some_server http://ppm.example.com/cgi-bin/ppmserver?urn:/
➥PPMServer
ppm> install mod_perl
   ...
ppm> set save
ppm> quit
C:\>
```

which assumes http://ppm.example.com/ has installed on it the ppm server from the PPM module from CPAN—doing so has the advantage of also being able to offer a search utility of package and author names of the packages available from the site.

The mod_perl PPM package includes the mod_perl DLL (called mod_perl.so in apache-1.3.15 and later, in accord with the Unix convention). When installed with the ppm utility, a post-install script will offer to install this DLL in your Apache modules/ directory. Installing a PPM package that matches the version of the Apache binary you are running is important for binary compatibility. Also in this regard, at the time of writing, you must be using an ActivePerl version in the 6xx series (or compatible), based on Perl-5.6.x, as earlier ActivePerl binaries in the 5xx series based on Perl-5.005 are not binary-compatible.

Whichever binary you choose, be careful to use versions of Perl, mod_perl, and Apache compiled against each other with the same compiler (generally Visual C++ 6), because rapid changes in the Win32 world mean that often incompatibilities exist between versions. As well, for binary compatibility, do not mix code compiled with Visual C++ 5 and Visual C++ 6 (note that ActivePerl binaries in the 6xx series are compiled with Visual C++ 6).

You can verify that mod_perl is installed by using Recipe 1.8.

1.3. Mac OS X Binary Installation

You want to use mod_perl with Apache on Apple's Mac OS X platform.

Technique

Use the DSO version of mod_perl that Apple ships with Mac OS X.

Comments

The advent of Apple's new Mac OS X operating system has some interesting ramifications for mod_perl and its community. First of all, the Macintosh is primarily marketed as a computer for consumers, but Mac OS X is built on top of a fully functional FreeBSD Unix system. The Unix layer has not been crippled or hidden from the users, either—the "Terminal" application gives access to a command-line shell, and there is at least one version of the X Window system that users can install for running window-based Unix applications.

Best of all, every computer running Mac OS X comes with fully functional versions of Perl, Apache, and mod_perl, making it a potentially attractive development machine. This situation seems to be stable—Apple depends on Perl for many of the installation and maintenance tasks that take place regularly on the computer, and Apache is the Web server used for the operating system's "Web Sharing" features.

The simplest way to install mod_perl on Mac OS X is to enable the DSO module provided with the system. Using whatever text editor you like, add the following lines to the file `/etc/httpd/httpd.conf`.

```
LoadModule perl_module    libexec/httpd/libperl.so
AddModule  mod_perl.c
```

You should add the `LoadModule` directive at the end of all the `LoadModule` directives that already exist in the file, and the `AddModule` directive after all the existing `AddModule` directives. Restart the Web server (by pressing "Stop" and then "Start" in the Web Sharing section of the "System Preferences" Sharing pane), and you should be all set. You can verify that mod_perl is installed by using Recipe 1.8.

1.4. Building mod_perl on Unix

You want to compile and install mod_perl from source on a Unix platform.

Technique

A full recipe for building on Unix would fill most of this chapter. The following recipe gives a reasonably concise overview of the build process for mod_perl on the Apache `1.3` architecture. For full documentation, refer to the `INSTALL` and `INSTALL.apaci` files in the mod_perl distribution.

First, ensure that your system has the following:

- A recent installation of `perl` (`5.005_03` or higher)
- An ANSI C compiler (`gcc`, for instance)
- `make`
- `gzip` and `tar` for uncompressing the source distribution archives

Additionally, it is highly recommended that you install the following CPAN modules so that you can run the mod_perl test suite:

- `libwww-perl`
- `HTML::Parser`

Next, download the Apache and mod_perl source distributions. You can find the latest version of mod_perl at `http://perl.apache.org/dist/`. Go to `http://www.apache.org/dist/httpd/` for the latest version of Apache.

CPAN also contains the mod_perl source distribution. However, some minor releases do not show up on CPAN due to naming conventions (for example, `1.25` and `1.26` show up under `modules/by-module/Apache/`, but `1.25_01` does not).

When you have the source archives downloaded, a typical mod_perl installation follows the same basic steps as installing any other Perl module, save a few specific arguments when creating the `Makefile`. A simple configuration might look like the following (slightly condensed and stripped of aesthetically unpleasing verbose output):

```
$ gzip -dc apache_1.3.22.tar.gz | tar -xvf -
$ gzip -dc mod_perl-1.26.tar.gz | tar -xvf -

$ cd mod_perl-1.26
$ perl Makefile.PL \
> APACHE_SRC=../apache_1.3.22/src \
> APACHE_PREFIX=/usr/local/apache \
> EVERYTHING=1 \
> DO_HTTPD=1 \
> USE_APACI=1 \
> APACI_ARGS='--enable-module=rewrite, \
>             --enable-module=info, \
```

```
>             --enable-module=expires, \
>             --disable-module=userdir'
Reading Makefile.PL args from ./makepl_args.mod_perl
Will configure via APACI
cp apaci/Makefile.libdir ../apache_1.3.22/src/modules/perl/Makefile.libdir
cp apaci/Makefile.tmpl ../apache_1.3.22/src/modules/perl/Makefile.tmpl
cp apaci/README ../apache_1.3.22/src/modules/perl/README
cp apaci/configure ../apache_1.3.22/src/modules/perl/configure
cp apaci/libperl.module ../apache_1.3.22/src/modules/perl/libperl.module
cp apaci/mod_perl.config.sh ../apache_1.3.22/src/modules/perl/mod_perl.config.sh
cp apaci/load_modules.pl ../apache_1.3.22/src/modules/perl/load_modules.pl
cp apaci/find_source ../apache_1.3.22/src/modules/perl/find_source
cp apaci/apxs_cflags ../apache_1.3.22/src/modules/perl/apxs_cflags
cp apaci/perl_config ../apache_1.3.22/src/modules/perl/perl_config
cp apaci/mod_perl.exp ../apache_1.3.22/src/modules/perl/mod_perl.exp
PerlDispatchHandler.........enabled
PerlChildInitHandler........enabled

...

$ make
(cd ../apache_1.3.22 && PERL5LIB=/home/geoff/src/mod_perl-1.26/lib make)
make[1]: Entering directory `/home/geoff/src/apache_1.3.22'
===> src
make[2]: Entering directory `/home/geoff/src/apache_1.3.22'
make[3]: Entering directory `/home/geoff/src/apache_1.3.22/src'
===> src/regex

...

Manifying blib/man3/Apache::SIG.3
Manifying blib/man3/Bundle::Apache.3
Manifying blib/man3/Apache::Options.3

$ make test
(cd ../apache_1.3.22 && PERL5LIB=/home/geoff/src/mod_perl-1.26/lib make)
make[1]: Entering directory `/home/geoff/src/apache_1.3.22'
===> src
make[2]: Entering directory `/home/geoff/src/apache_1.3.22'
```

```
make[3]: Entering directory `/home/geoff/src/apache_1.3.22/src'
===> src/regex

...

cp t/conf/mod_perl_srm.conf t/conf/srm.conf
../apache_1.3.22/src/httpd -f `pwd`/t/conf/httpd.conf -X -d `pwd`/t &
httpd listening on port 8529
will write error_log to: t/logs/error_log
letting apache warm up...\c
done
/usr/local/bin/perl t/TEST 0
modules/actions.....ok
modules/cgi.........ok

...

internal/taint......ok
All tests successful, 1 test skipped.
Files=34, Tests=457, 26 wallclock secs (20.90 cusr +  1.12 csys = 22.02 CPU)
kill `cat t/logs/httpd.pid`
rm -f t/logs/httpd.pid
rm -f t/logs/error_log

$ su
Password:
# make install
(cd ../apache_1.3.22 && PERL5LIB=/home/geoff/src/mod_perl-1.26/lib make)
make[1]: Entering directory `/home/geoff/src/apache_1.3.22'
===> src
make[2]: Entering directory `/home/geoff/src/apache_1.3.22'
make[3]: Entering directory `/home/geoff/src/apache_1.3.22/src'
===> src/regex

...

make[2]: Leaving directory `/home/geoff/src/apache_1.3.22'
+--------------------------------------------------------+
| You now have successfully built and installed the      |
| Apache 1.3 HTTP server. To verify that Apache actually |
| works correctly you now should first check the         |
```

```
| (initially created or preserved) configuration files   |
|                                                        |
|   /usr/local/apache/conf/httpd.conf
|                                                        |
| and then you should be able to immediately fire up     |
| Apache the first time by running:                      |
|                                                        |
|   /usr/local/apache/bin/apachectl start
|                                                        |
| Thanks for using Apache.       The Apache Group        |
|                                http://www.apache.org/  |
+--------------------------------------------------------+
make[1]: Leaving directory `/home/geoff/src/apache_1.3.22'
Appending installation info to /usr/local/lib/perl5/5.6.1/i686-linux-thread-
multi/perllocal.pod
```

Now, as `root`, issue

```
# /usr/local/apache/bin/apachectl start
```

to start the server.

Comments

Here is a brief explanation of the arguments we passed to the `perl Makefile.PL` portion of the mod_perl build process. You can find a full listing of acceptable options in Appendix A as well as in the `INSTALL` file in the mod_perl source distribution.

Table 1.1 *Arguments Passed to* `perl Makefile.PL`

Option	Description
`APACHE_SRC`	The directory that contains the Apache source headers.
`APACHE_PREFIX`	The directory prefix that is prepended to the Apache installation. Because we added `APACHE_PREFIX` to our example, it is not necessary to `cd` over to the Apache sources and issue `make install`—mod_perl does it for us. If you leave this argument out, you will have to install Apache as well, and the preceding dialogue will look slightly different.
`EVERYTHING`	When true (`EVERYTHING=1`) enables all the available mod_perl hooks. This includes all the advanced features of mod_perl such as authentication and authorization control, configuration of the server with Perl, output filtering, and more.

Table 1.1 *(continued)*

Option	Description
DO_HTTPD	When true (DO_HTTPD=1), mod_perl automatically builds Apache for you; otherwise, it prompts for build instructions.
USE_APACI	When true (USE_APACI=1), mod_perl uses the Apache AutoConf Interface (APACI) for configuration, which is the preferred method. The alternative is to use Apache's manual configuration files, which is becoming rapidly deprecated and is not covered here.
APACI_ARGS	This is a comma-delimited string of APACI commands to pass to Apache.

As a result of the installation, you should have an entire directory structure under /usr/local/apache/ that you can configure to meet your needs. You will also have several necessary Perl modules installed in Perl's site_perl directory under the Apache:: namespace. You can verify that mod_perl is active in your Apache server by following the instructions in Recipe 1.8

If you encounter any problems getting mod_perl to compile properly, as well as any runtime problems that appear to be caused by a broken installation, consult the SUPPORT document in the mod_perl distribution for detailed information on the next steps to take.

1.5. Building mod_perl on Windows

You want to compile and install mod_perl from source on Microsoft Windows.

Technique

Have patience.

Comments

At present, mod_perl requires Microsoft's Visual C++ to compile on Win32. You will also need to have compiled Apache from the source distribution, because the Apache headers and library files will be needed. Upon unpacking the mod_perl distribution from CPAN or from http://perl.apache.org/dist/, follow one of the following two paths.

Visual Studio Build

A. Run

```
C:\mod_perl> perl Makefile.PL
C:\mod_perl> nmake
```

which will set up some files needed for the library build.

B. Launch Visual Studio, and open the mod_perl `dsp` via the following

1. Select `File -> Open Workspace ...`.
2. Select `Files of type [Projects (*.dsp)]`.
3. Open `mod_perl-1.26/src/modules/win32/mod_perl.dsp`.

as in Figure 1.1.

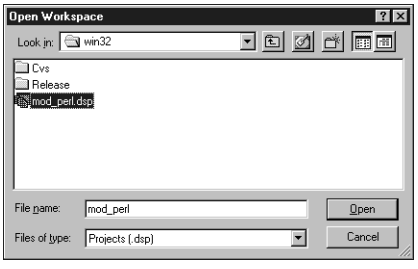

Figure 1.1
Opening the mod_perl project in Visual Studio.

C. You will then need to add some Apache and Perl directories. To add the `include` directories, follow these steps:

1. Select `Tools -> Options -> [Directories]`.
2. Select `Show directories for: [Include files]` and add, as appropriate for your system, the following, as shown in Figure 1.2:
   - `C:\apache_1.3.22\src\include`
   - `C:\apache_1.3.22\src\os\win32` (needed for `apache_1.3.22` and greater) This should expand to `C:\...\mod_perl-1.26\src\modules\perl`.
   - `C:\Perl\lib\Core`

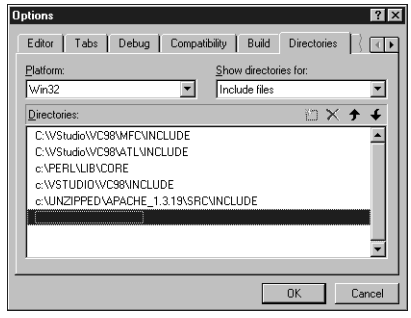

Figure 1.2
Setting options.

D. To include the necessary libraries, select `Project -> Add to Project -> Files`, and add, again as appropriate for your system, one of the following, as shown in Figure 1.3:

- `perl56.lib` (or `perl.lib`) (for example, `C:\Perl\lib\Core\perl56.lib`)
- `ApacheCore.lib` (for example, `C:\apache_1.3.22\src\Release\ApacheCore.lib`)

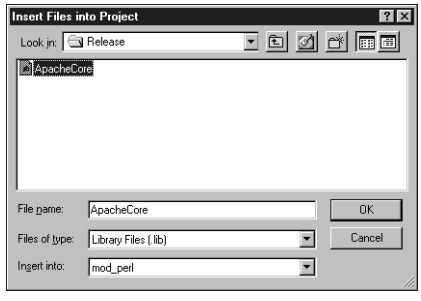

Figure 1.3
Adding `ApacheCore.lib` *to a project.*

E. To reduce the size of the resulting DLL, select `Project -> Settings -> [C/C++] -> Category: [Code Generation] -> Use runtime library: [Multithreaded DLL]`.

F. Finally, build the mod_perl DLL (`mod_perl.so`) by following these steps:

1. Select `Build -> Set Active Configuration... -> [mod_perl - Win32 Release]`.
2. Select `Build -> Build mod_perl.so`.

You can then test the results by using

```
C:\mod_perl> nmake test
```

Complete the build by copying `mod_perl.so` to your appropriate Apache modules directory

```
C:\mod_perl> copy src\modules\win32\Release\mod_perl.so \Apache\modules
```

and then issuing the command

```
C:\mod_perl> nmake install
```

to install the necessary Perl modules that support the mod_perl installation.

Command-Line Build

You can also build mod_perl, including `mod_perl.so`, entirely from the command line by generating the `Makefile` as, for example (all on one line),

```
C:\mod_perl> perl Makefile.PL APACHE_SRC=..\apache_1.3.22 INSTALL_DLL=\Apache\
➥modules
```

The arguments accepted include

- `APACHE_SRC`: This gives the path to the Apache sources (for example, `..\apache_1.3.22`). It is assumed that Apache has already been built in this directory.
- `INSTALL_DLL`: This gives the location of where to install `mod_perl.so` (for example, `\Apache\modules`). No default is assumed. If this argument is not given, you must copy `mod_perl.so` manually.
- `DEBUG`: If true (`DEBUG=1`), a version with debugging enabled will be built (this assumes that a corresponding Apache binary with debugging enabled has been built). If false, or not given, a Release version will be built.
- `EAPI`: If true (`EAPI=1`), `EAPI` (`Extended API`) will be defined when compiling. This is useful when building mod_perl against mod_ssl patched Apache sources. If this flag is not defined, a warning is made about a possible crash when starting a mod_ssl patched Apache. If `EAPI` is false, or not given, `EAPI` will not be defined.

After generating the `Makefile`,

```
C:\mod_perl> nmake
C:\mod_perl> nmake test
C:\mod_perl> nmake install
```

completes the build.

If neither of these build procedures succeeds, be sure that you can successfully build other Perl modules requiring a C compiler, to give you confidence that the failure is not due to a misconfiguration of your Perl installation. If you can build other Perl modules, try the mod_perl CVS version, as shown in Recipe 1.16, to see whether any breakage has been fixed there. If this fails, ask for help on the mod_perl mailing list—give your Perl and Apache version, what you tried, and the error that resulted.

You may feel some trepidation in using mod_perl on Win32 if you run into build problems, because these can be particularly frustrating. Don't get too discouraged, however. Once built, mod_perl on Win32 is used very much like its Unix cousin, save for the usual peculiarities and caveats for Perl and Apache in general on this platform (this includes the fact that mod_perl on Win32 is limited to one interpreter at a time, a restriction that will be lifted when Apache-2.0 and the associated mod_perl-2.0 are released).

1.6. Building mod_perl on Mac OS X

You want to compile and install mod_perl from source on Apple's Mac OS X platform.

Technique

Follow the basic Unix installation given in Recipe 1.4, tacking on a few extra steps along the way.

Comments

If you expect to do any real development with mod_perl, you will probably want to compile your own version of the server software from source. If you have any experience building mod_perl on another Unix platform, you will find the process very similar on Mac OS X, and the Recipe 1.4 will be your best guide.

Because the Mac OS X platform is fairly new, some platform-specific problems are still being fixed in the mod_perl build process. The most successful way to compile so far seems to be as a static module, using only the `EVERYTHING=1` argument to `perl Makefile.PL`. All build problems should ideally be fixed relatively soon, however—if you run into problems the best sources of support are the mod_perl list and the Mac OS X Perl list, `macosx@perl.org`.

A couple of things to note. First, you will need to install the OS X Developer Tools, available for free download. As of this writing, the Developer Tools are free, but users must register as Apple developers at `http://www.apple.com/`. Additionally many retail versions of Mac OS X include the Developers Tools CD in the box.

The Developer Tools include such essential system components as `make`, the `gcc` compiler (the executable is actually `/usr/bin/cc`, but it's really the standard GNU `gcc` compiler with some Apple enhancements), and other things developers will have a hard time living without.

Second, most users will be running Mac OS X on Apple's HFS+ file system, which, as of this writing, uses case-insensitive filenames. This creates a couple of gotchas for mod_perl development. First, the mod_perl installation process will encourage you to install Perl's `LWP` modules so that it can run a few HTTP requests to test mod_perl before installing it. Unfortunately, at the time of this writing `LWP` installs its `HEAD` script into the `/usr/bin/` directory. Although the case of any filename is preserved on HFS+, files called, say, `FOO` and `foo` cannot exist in the same directory. Thus `LWP`'s `HEAD` program overwrites the Unix file-viewing utility `/usr/bin/head`. Here is a workaround for this problem. You must be an Administrator to perform most of these steps—enter your regular user password when prompted.

```
% cp /usr/bin/head ~/head

  ... install the LWP modules ...

% sudo mkdir /usr/local/bin
% sudo mv /usr/bin/HEAD /usr/bin/GET /usr/bin/POST /usr/local/bin/
% sudo mv /usr/bin/lwp-* /usr/local/bin/
% sudo mv ~/head /usr/bin/
```

This creates a `/usr/local/bin/` directory, which is a more appropriate place than `/usr/bin/` to install the `LWP` utilities. If you already have a `/usr/local/bin/` directory, skip the `mkdir` step. The next few commands move the `LWP` utilities from `/usr/bin/` into `/usr/local/bin/`, then finally put the `head` utility back into `/usr/bin/` where it belongs. If you don't fix the `HEAD` problem properly, you may see verbose error messages like `Usage: HEAD [-options] <url>...` when you try to use the `head` utility.

Finally, the case-insensitive filesystem can create some security holes if you're not careful. If you use configuration directives like

```
<Files "foo.html">
  deny from all
</Files>
```

then a user may still be able to access `foo.html` by requesting `FOO.html`. Apache will see that `FOO.html` doesn't match the `<Files>` directive, so access will be granted. Then the filesystem will deliver the file, because the name `FOO.html` is a valid name for the file `foo.html`.

There is an Apache module called `mod_hfs_apple`, available from `http://www.opensource.apple.com/projects/darwin/darwinserver/` , which attempts to solve this security problem. It may not solve the entire problem yet, however, so check around for security updates before deploying an Apache server on HFS+ in public. And when writing file-handling code in your own modules, try to keep the case-insensitive nature of the HFS+ filesystem in mind so that you don't create any bugs or security holes of your own.

1.7. Building mod_perl as a Shared Library

You want to use mod_perl as a DSO (Dynamic Shared Object).

Technique

Add the `USE_DSO=1` flag to your mod_perl build arguments.

```
$ perl Makefile.PL \
> APACHE_SRC=../apache_1.3.22/src \
> APACHE_PREFIX=/usr/local/apache \
> EVERYTHING=1 \
> DO_HTTPD=1 \
> USE_DSO=1 \
> USE_APACI=1 \
> APACI_ARGS='--enable-module=rewrite, \
>             --enable-module=info, \
>             --enable-module=expires, \
>             --disable-module=userdir' \
```

Comments

Although most people who run production mod_perl environments choose to have mod_perl compiled statically within their `httpd` binary, this option is not the only one.

Most Apache modules are capable of being loaded into the server dynamically, including mod_perl.

Apache's DSO (Dynamic Shared Object) feature allows you to add modules on-the-fly at startup using `httpd.conf` directives. This feature has the advantage of allowing you to adjust your binary based on your immediate needs, dropping and adding modules as you fine-tune your application without recompiling Apache every time. It also makes having a rather lightweight base Apache possible; because some modules (such as mod_perl and mod_rewrite) are quite large, having them burdening your process size when they are not necessary may not be desirable.

Using mod_perl as a DSO is just as easy as adding the `USE_DSO=1` flag at build time and letting mod_perl build Apache. If you look at the resulting `httpd.conf`, you will see that the following lines were automatically added for you:

```
LoadModule perl_module libexec/libperl.so
AddModule mod_perl.c
```

As mentioned earlier, traditionally DSO installations have been considered less stable than statically compiled versions. This situation is improving as both mod_perl and Apache support for DSO matures. Be sure to check for recent mod_perl developments in this area.

1.8. Testing Your Installation

You want to be sure that your Apache is mod_perl-enabled.

Technique

Check the `Server` response header via `telnet`.

```
$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 08 Oct 2001 14:43:18 GMT
Server: Apache/1.3.22 (Unix) mod_perl/1.26
```

```
Last-Modified: Fri, 04 May 2001 00:00:38 GMT
ETag: "1c2cd-5b0-3af1f126"
Accept-Ranges: bytes
Content-Length: 1456
Connection: close
Content-Type: text/html
Content-Language: en

Connection closed by foreign host.
```

Comments

After attempting one of the installations outlined in this chapter, you will want to test to see whether mod_perl was successfully installed. A simple `telnet` session ought to be sufficient to check the `Server` response header and see whether mod_perl is present. Of course, if your `ServerTokens` directive is set to something other than `Full` (the default) you will not see the mod_perl token, even if your install was a success.

If things did not go smoothly and you find yourself here without a working installation, not to worry. Read over the `INSTALL` and `SUPPORT` documents in the mod_perl distribution and scour the mod_perl mailing list archives from your favorite search engine. Also, be sure to read the section on installation in the mod_perl Guide at `http://perl.apache.org/guide/`—it is an invaluable document that addresses most of the problems you might encounter.

1.9. Changing Apache Installation Directories

You want to change the default Apache installation directories.

Technique

Use the `APACI --with-layout` option with an entry from `config.layout`.

```
$ perl Makefile.PL \
> APACHE_SRC=../apache_1.3.22/src \
> APACHE_PREFIX=/opt/apache \
> EVERYTHING=1 \
> DO_HTTPD=1 \
> USE_DSO=1 \
```

```
> USE_APACI=1 \
> APACI_ARGS='--enable-module=rewrite \
>             --enable-module=info \
>             --enable-module=expires \
>             --disable-module=userdir \
>             --with-layout=opt'
```

Comments

Although not mod_perl-specific, knowing how to tweak your installation is sometimes helpful. By default, Apache uses `--with-layout=Apache`, which installs the `httpd` binary and supporting files and documentation into `/usr/local/apache`. If you want to change this behavior, you can either specify a pre-existing layout from `config.layout`, or add your own layout to the file. In either case, customizing the layout of Apache is then as simple as adding the `--with-layout` argument to `APACI_ARGS` and then matching the `config.layout` prefix option to the `APACHE_PREFIX` argument to `perl Makefile.PL`. Keep in mind that `APACHE_PREFIX` overrides the `--prefix` directive within `config.layout`, so unless the two match exactly you will not end up with the layout you expect.

1.10. Adding mod_perl to an Existing Apache Server

You have an Apache server installed and want to add mod_perl to it.

Technique

Build mod_perl outside of the Apache environment using `APXS`.

```
$ perl Makefile.PL \
> USE_APXS=1 \
> WITH_APXS=/usr/local/apache/bin/apxs \
> EVERYTHING=1
```

Comments

With `USE_DSO=1` in your build arguments, mod_perl not only adds itself to Apache as a DSO, but it also builds Apache at the same time. Because the purpose of using a module as a DSO is to prevent having to rebuild Apache every time you add a module, this feature is convenient but not ideal.

Using the APXS (APache eXtenSion) toolkit, you can build mod_perl as a DSO outside of the Apache source tree and without rebuilding your Apache binary. All that is necessary is to have mod_so.c (which provides DSO support) statically compiled into Apache. Then, after building mod_perl using APXS, you can enable mod_perl using the LoadModule directive, as described in Recipe 1.7.

You may notice the mod_perl build process returning warnings (such as pthreads or uselargefiles warnings) about how your current perl will affect your existing Apache binary. However, the mod_perl build process will usually give you some direction as to steps to take to remedy the situation so that your build will be successful.

1.11. Reusing Configuration Directives

You want to create a file to reuse your configuration directives.

Technique

Store your build arguments in makepl_args.mod_perl.

```
# file makepl_args.mod_perl
APACHE_SRC=../apache_1.3.22/src
APACHE_PREFIX=/usr/local/apache
EVERYTHING=1
DO_HTTPD=1
USE_APACI=1
APACI_ARGS=--enable-module=rewrite, --enable-module=info
```

Comments

To ease the pain of having to type your configuration directives over and over again (or so that you can remember exactly what you typed last month), mod_perl provides a way to supply build arguments from a file. Currently, Makefile.PL will look for its arguments in the following files relative to the mod_perl sources (in the following order):

```
./makepl_args.mod_perl
../makepl_args.mod_perl
./.makepl_args.mod_perl
../.makepl_args.mod_perl
$ENV{HOME}/.makepl_args.mod_perl
```

It is important to note that although we were able to break up the APACI_ARGS argument onto separate lines when building from the command line, makepl_args.mod_perl requires one argument per line. An alternative syntax is to place each APACI argument on a separate line:

```
APACI_ARGS=--enable-module=rewrite
APACI_ARGS=--enable-module=info
```

Also note the absence of enclosing ticks for APACI_ARGS, which is also different from the command-line syntax.

1.12. Re-Creating a mod_perl Installation

You want to know how mod_perl was built so that you can build another similar binary.

Technique

Look at the generated files mod_perl.config and config.status.

Comments

Unfortunately, mod_perl does not stash its compile options away so that you can just port a file to a new machine and make an identical build. However, if you use APACI to install Apache and either APACI or APXS to install mod_perl, two files can help. config.status is found in the root directory of the Apache source tree, while mod_perl.config can be located in one of two places: either the apaci/ directory in the mod_perl source tree (for APXS builds) or in src/modules/perl/ in the Apache source tree (for APACI builds). Between these two files, you can determine which options were enabled at build time and re-create an existing installation. Note that, at the present time, these files are not generated for a Win32 build.

1.13. Distributing mod_perl to Many Machines

You want to prepare mod_perl for distribution across multiple machines.

Technique

Use make targets tar_Apache or offsite-tar (for Unix) or ppd (for Win32).

Comments

Unlike Apache, a mod_perl installation cannot be easily moved from one machine to another—there is more than the `httpd` binary to worry about. If you have many machines that require a mod_perl installation, building Apache and mod_perl from source on all of them can be long and tedious. Under some Unix variants, you have the option of using a third-party packager (such as `rpm`) to roll all the necessary files together. For Windows and the other Unix platforms, this option is not viable. In these cases, mod_perl provides some `make` targets that might help speed things along.

For Unix, the `offsite-tar` target will create a tarball called `mod_perl-1.26.tar.gz` in the mod_perl source directory. It will contain all the required files for a mod_perl build, including the necessary files from the Apache sources. This will allow you to successfully perform an APXS build against an existing Apache installation without also needing to have the full set of Apache sources present on the new machine. Just unpack the file and follow the instructions for an `APXS` build given in Recipe 1.10.

If your `httpd` already contains a static mod_perl, then all you need are the Perl modules that mod_perl installs for you. The `tar_Apache` target will roll these up for you into `Apache.tar`, which can then be extracted into the `site_perl` directory on another machine.

For Win32, the process is a bit different and requires some finesse. To create a mod_perl PPM (Perl package manager) file as used, for example, with ActivePerl, start by running `Makefile.PL` with `BINARY_LOCATION` specified:

```
C:\mod_perl> perl Makefile.PL BINARY_LOCATION=x86/mod_perl.tar.gz ...
```

Build mod_perl in the usual way, and then make the `ppd` file as

```
C:\mod_perl> nmake ppd
```

which will create `mod_perl.ppd`. The binary package for distribution is built as

```
C:\mod_perl>  tar cvf mod_perl.tar blib
C:\mod_perl>  gzip --best mod_perl.tar
```

which, in this example, is to be placed in a directory `x86/` relative to the location you put `mod_perl.ppd`. This can then be installed with the `ppm` utility as discussed in Recipe 1.2.

This procedure is the standard one for building `ppm` packages in general, but for mod_perl, you would probably also want to include the mod_perl DLL to be installed in the Apache `modules/` directory, and also include a post-install script to install it. To do this, proceed as before in building mod_perl, and then copy `mod_perl-1.26/src/modules/win32/Release/mod_perl.so` to the directory containing the mod_perl `blib/` subdirectory. Create a post-install script (say, `install.ppm`).

Listing 1.1 `install.ppm`

```
#!perl -w

use strict;

my $so = 'mod_perl.so';   # name of the mod_perl dll

# Get the name of the directory to install $so.
my $base =
  GetString ("\nWhere should mod_perl.so be placed in?\n  (q to quit)",
             'C:/Apache/modules') ;
if ($base eq 'q') {
  suggest_manual("Aborting installation ...");
}
$base =~ s/mod_perl.so$//i;
$base =~ s!\\!/!g;
$base =~ s!/$!!;

# If the directory doesn't exist, offer to create it.
if (! -d $base) {
my $ans = GetString("$base does not exist. Create it?", 'no');
  if ($ans =~ /^y/i) {
    mkdir $base;
    suggest_manual("Could not create $base: $!") if (! -d $base);
  }
  else {
    suggest_manual("Will not create $base.");
  }
}

# Copy $so to the indicated directory.
use File::Copy;
move($so, "$base/$so");
suggest_manual("Moving $so to $base failed: $!") if (! -f "$base/$so");
```

Listing 1.1 *(continued)*

```
print "$so has been successfully installed \n\t to $base/$so\n";
sleep(5);   # give the user time to read, before the window closes

# routine to suggest manual installation if user declines
sub suggest_manual {
  my $msg = shift;
  print $msg, "\n";
  print "Please install $so manually\n";
  sleep(5);
  exit(0);
}

# routine to get a string from a prompt, offering a default
sub GetString {
  my ($prompt, $default) = @_;
  printf ("%s [%s] ", $prompt, $default);
  chomp ($_ = <STDIN>);
  /\S/ and return $_;
  /^$/ and return $default;
  return;
}
```

The binary package is then made as

```
C:\mod_perl> tar cvf mod_perl.tar blib mod_perl.so install.ppm
C:\mod_perl> gzip --best mod_perl.tar
```

So that this post-install script runs when the mod_perl package is installed with the `ppm` utility, add `<INSTALL EXEC="perl">install.ppm</INSTALL>` within the `<IMPLEMENTATION>` section of `mod_perl.ppd`, as shown here:

```
<SOFTPKG NAME="mod_perl" VERSION="1,26_01-dev,0,0">
         <TITLE>mod_perl</TITLE>
         <ABSTRACT>Embed a Perl interpreter in the Apache HTTP server</ABSTRACT>
         <AUTHOR>Doug MacEachern &lt;dougm@pobox.com&gt;</AUTHOR>
         <IMPLEMENTATION>
                <OS NAME="MSWin32" />
                <ARCHITECTURE NAME="MSWin32-x86-multi-thread" />
                <CODEBASE HREF="http://ppm.example.com/ppmpackages/x86/
➥mod_perl.tar.gz" />
               <INSTALL EXEC="perl">install.ppm</INSTALL>
       </IMPLEMENTATION>
</SOFTPKG>
```

1.14. Inspecting an Existing Server

You want to know what parts of the mod_perl API are available on an existing installation.

Technique

Check the output from `/perl-status?hooks`, provided by `Apache::Status`.

First, make the required changes to `httpd.conf` to activate `Apache::Status`

```
PerlModule Apache::Status

<Location /perl-status>
   SetHandler perl-script
   PerlHandler Apache::Status
   Order Allow,Deny
   Allow from localhost
   Allow from .example.com
</Location>
```

then restart Apache and fetch `http://`*`www.example.com`*`/perl-status?hooks` from your favorite browser.

Comments

If you are in an environment where you do not have control over how mod_perl is built, you may not have access to the entire mod_perl API. Checking which hooks were enabled at build time may help you determine which phases are available to you, and thus which CPAN modules can offer assistance when building your application.

`perl-status?hooks` just uses the built-in `mod_perl::hook()` and `mod_perl::hooks()` methods. If you are developing a handler that might run in different environments and you need to program intelligently around the availability of a particular hook, you can use these methods. Beware of the spelling of the hooks—they are case sensitive.

```
use mod_perl_hooks;

# Require ALL mod_perl hooks (i.e., EVERYTHING=1)
foreach my $hook (mod_perl::hooks()) {
  die "$hook not enabled!" unless mod_perl::hook($hook);
}
```

Another programmatic option is to check the global hash `%Apache::MyConfig::Setup`. `Apache::MyConfig` is a package that is created when mod_perl is compiled. It contains some important build information, such as enabled hooks and platform-dependent information. You can loop through the hash to find the status of all the various build time options.

```
use Apache::MyConfig;

foreach my $key (sort keys %Apache::MyConfig::Setup) {
  print "$key => $Apache::MyConfig::Setup{$key}\n";
}
```

Finally, if you absolutely require a particular hook, or do not want to program around the availability of one, you can rely on `mod_perl::import()` to catch the availability of the hook at compile time.

```
use mod_perl qw(PerlStackedHandlers PerlLogHandler);

# Now we know we're ok.
$r->push_handlers(PerlLogHandler => \&logger);
```

1.15. Installing Apache Modules from CPAN

You want to install an Apache module you found on CPAN.

Technique

Follow the canonical CPAN installation steps.

```
$ gzip -dc Apache-Module-0.01.tar.gz | tar -xvf -
$ cd Apache-Module-0.01
$ perl Makefile.PL
$ make
$ make test
$ su
Password:
# make install
```

Comments

Part of the power of mod_perl (and Perl in general) is the power of CPAN and its freely available modules. Nearly all the CPAN modules for mod_perl are located under the Apache tree, indicating that they are designed for use only in a mod_perl environment. Of course, you should read the `README` file and any other installation instructions included with the distribution before attempting to install the module. However, the preceding are the typical series of commands used for most of the modules on CPAN, and should prove sufficient to get you on your way.

Another option for installing modules is to use the `CPAN.pm` module, which comes bundled with recent Perl distributions. Upon invoking the interactive shell,

```
$ perl -MCPAN -e shell
cpan>
```

you will (the first time) be taken through a series of questions to set up your configuration. Afterward, building and installing a module is as easy as

```
cpan> install Apache::Module
```

If for some reason the build or the tests (if any) fail, the module won't be installed by default. One nice feature of this way of installing modules is that `CPAN.pm` will, in most cases, automatically detect whether the requested module requires installation of another module, and then offer to install this one for you, as well. For more details on the commands available, type **`h`** at the CPAN shell prompt for a summary, or see `perldoc CPAN` for a more complete description.

You can also use the `CPAN.pm` module to install mod_perl itself; however, because of the number of options available within mod_perl, it is recommended that you familiarize yourself with a manual install first. Having done so, the use of `makepl_args.mod_perl` (described in Recipe 1.11) for saving the arguments passed to `Makefile.PL` is quite useful for use with a `CPAN.pm` install.

1.16. Following mod_perl Development

You want to follow mod_perl development closely.

Technique

Use anonymous CVS to obtain the most recent version of mod_perl.

Comments

If you just cannot wait to get the latest patches or bug fixes, or you like living life on the edge, then anonymous CVS access to the mod_perl sources is for you. First, make sure you have CVS installed on your system, then log in and check out the mod_perl sources (the password is "**anoncvs**" without the quotes).

```
$ cvs -d ":pserver:anoncvs@cvs.apache.org:/home/cvspublic" login
(Logging in to anoncvs@cvs.apache.org)
CVS password:

$ cvs -d ":pserver:anoncvs@cvs.apache.org:/home/cvspublic" checkout modperl
cvs server: Updating modperl
U modperl/.cvsignore
U modperl/.gdbinit
...
```

You will then have a `modperl/` directory, from which you can build mod_perl as described in Recipe 1.4.

To keep your sources current, every once in a while you should

```
$ cvs update -dP
```

Or, to see what has changed since you last updated, issue

```
$ cvs diff -u
```

If you don't have access to a CVS client, tarballs of the latest mod_perl development version are rolled every six hours and placed on `http://perl.apache.org/from-cvs/`.

If you just want to lurk around and watch development for a while, you can subscribe to the development and cvs mailing lists by sending an empty e-mail to `dev-subscribe@perl.apache.org` for discussion of development of mod_perl, or `modperl-cvs-subscribe@perl.apache.org` for automatic messages whenever the mod_perl CVS sources are modified.

1.17. Beyond Simple CVS

Simple CVS access is not enough—you want to live on the bleeding edge.

Technique

Recompile Apache and mod_perl from CVS nightly.

Comments

Building all the essential parts of mod_perl (including Perl!) from development sources is possible, but doing so is not for the faint of heart. Although Perl, mod_perl, and Apache are some of the most stable software products available, such experiments should only be considered in a development environment. CVS versions are not guaranteed to compile, let alone work. To save yourself a few headaches, make certain you can build mod_perl and Apache from a standard, stable distribution first. After that, you can check out Apache CVS in the same base directory as your mod_perl CVS sources.

```
$ cvs -d ":pserver:anoncvs@cvs.apache.org:/home/cvspublic" checkout apache-1.3
```

And install the following scripts in non-`root` and `root` `crontab`s, respectively (see Listings 1.2 and 1.3). Be sure to leave a suitable distance between script execution times to allow for the speed of your connection and machine.

Listing 1.2 `make.ksh`

```
#!/bin/ksh

# Keep Apache and mod_perl up to date.
# Install in non-root crontab.

source="/path/to/your/source"

echo "about to update apache\n"
cd $source/apache-1.3
cp src/CHANGES src/CHANGES.old
cvs update

echo "about to update modperl\n"
cd $source/modperl
make realclean
cp Changes Changes.old
cvs update -dP

echo "about to make modperl\n"
perl Makefile.PL \
```

Listing 1.2 *(continued)*

```
    APACHE_SRC=$source/apache-1.3/src \
    APACHE_PREFIX=/usr/local/apache \
    EVERYTHING=1 \
    DO_HTTPD=1 \
    USE_APACI=1 \
    APACI_ARGS='--enable-module=rewrite \
                --enable-module=info \
                --enable-module=expires \
                --disable-module=userdir'
make && make test
```

Listing 1.3 `install.ksh`

```
#!/bin/ksh

# Keep Apache and mod_perl up to date.
# Install in root crontab
#
# The result is a nice diff of the change logs
# for both mod_perl and Apache, emailed
# directly to you.

source="/path/to/your/source"
email="your@email.address"

/usr/local/apache/bin/apachectl stop

cd $source/modperl
make install

>/usr/local/apache/logs/error_log

/usr/local/apache/bin/apachectl start

sleep 10
today=`date +%b" "%d", "%Y`

cd $source/modperl
echo "\n---- mod_perl Changes ----" > Changes.diff
diff -u Changes.old Changes >> Changes.diff
```

Listing 1.3 *(continued)*

```
cd $source/apache-1.3/src
echo "\n---- Apache Changes ----" > Changes.diff
diff -u CHANGES.old CHANGES >> Changes.diff

cat /usr/local/apache/logs/error_log \
  $source/modperl/Changes.diff \
  $source/apache-1.3/src/Changes.diff \
  | mail -s "httpd $today" $email
```

1.17. Building mod_perl with Different `perl`s

You want to run mod_perl using a different version of `perl` than is the default on the server itself.

Technique

Build mod_perl using the version of `perl` you want mod_perl to use at runtime

```
$ /src/bleedperl/bin/perl5.7.2 Makefile.PL \
> APACHE_SRC=../apache-1.3/src \
> APACHE_PREFIX=/usr/local/apache \
> EVERYTHING=1 \
> DO_HTTPD=1 \
> USE_APACI=1 \
> APACI_ARGS='--enable-module=rewrite, \
>             --enable-module=info, \
>             --enable-module=expires, \
>             --disable-module=userdir'
```

Comments

If you begin to get into mod_perl development, start compiling mod_perl from CVS regularly, or just want to upgrade your installation, you may want to use a more recent version of `perl` than the other applications on your box permit. Perhaps you have some legacy code that has only been tested against `5.005` while your mod_perl application makes copious use of the `our` construct introduced in `5.6.0`. Building and maintaining a mod_perl installation with a specific or separate version of `perl` is actually not as complex as it sounds.

The `perl` binary you use to build mod_perl will be the one it uses at runtime. For instance, the build options shown in the solution code use a current bleeding-edge `perl` binary with the Apache CVS sources. This explanation is somewhat misleading. Because mod_perl embeds the perl interpreter into Apache, it does not invoke `perl` binary you used at build time during normal operation—the perl interpreter mod_perl will use at runtime for its handlers, as well as for `Apache::Registry` scripts, is the embedded interpreter and not the binary sitting in `/usr/bin/perl`.

Where the current Perl installation on your system does come into play is with the files installed into `@INC` during the mod_perl build process. At runtime, mod_perl uses the `@INC` of the perl interpreter it was built with to search for the various Perl modules it needs (like `Apache::Registry` and `Apache::Constants`), as well as any Perl modules your handlers will rely upon (like `Time::HiRes`). This means that you have to use the same `perl` to install new modules as you used to build mod_perl for your mod_perl handlers to have access to them.