;login:
THE MAGAZINE OF USENIX & SAGE
February 2003 • volume 28 • number 1

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

practical perl

CPAN, Modules, and the CPAN Shell

by Adam Turoff

Adam is a consultant who specializes in using Perl to manage big data. He is a long- time Perl Monger, a technical editor for *The Perl Review*, and a frequent presenter at Perl conferences.

ziggy@panix.com

If you asked a room full of Perl programmers to name their favorite feature of Perl, the vast majority of them would say that it is CPAN, the Comprehensive Perl Archive Network. Many non-Perl programmers agree that CPAN is the most interesting feature about Perl.

CPAN is a globally distributed library of scripts, modules, documentation, and other resources created by and for Perl programmers. If you need to create an application with a graphical user interface, connect to a database, or access a Web service, you can find a great many modules to help you at your local CPAN mirror. Currently, over 200 public CPAN mirrors, plus countless private mirrors, are available around the world.

Most Perl programmers are familiar with the two primary Web-based interfaces to CPAN: the main Web page, *http://www.cpan.org/* (also available at your local CPAN mirror), and the CPAN search engine, *http://search.cpan.org/*. Both of these sites are excellent resources if you are looking for Perl modules to use. However, finding a module that may help you solve a problem easily is only half of the battle. Fortunately, searching is the difficult part; installing a module once you know its name is much much easier, thanks to the CPAN shell.

The CPAN Shell

One of the many core modules that comes with every version of Perl released since 1997 is Andreas Koenig's CPAN module. This very useful module has a great many features, but its overall purpose is to help you maintain a Perl installation by making it easy to upgrade and install Perl modules from your local CPAN mirror.

The most common way to use the CPAN module is to use the CPAN shell:

```
[ziggy@chimay ~]$ perl -MCPAN -e shell

cpan shell – CPAN exploration and modules installation (v1.63)
ReadLine support enabled

cpan>
```

Within the CPAN shell, installing modules is easy – just type install and a list of modules to install. The install will download, extract, configure, build, test, and install a module.

```
cpan> install LWP::Simple
....
cpan> install Bundle::DBI DBD::mysql DBD::SQLite
....
```

Another way to use the CPAN module is to install modules directly from the command line:

```
[ziggy@chimay ~]$ perl -MCPAN -e 'install LWP::Simple'
....
[ziggy@chimay ~]$
```

The CPAN shell has many other uses. It can act as a basic search tool, report on what modules you have installed, and determine which installed modules are out-of-date. To inspect a module, just type m *modulename* and it will display the current version and, if it is installed, the current location of that module:

```
cpan> m DBI
Module id = DBI
    DESCRIPTION     Generic Database Interface
                    (see DBD modules)
    CPAN_USERID     TIMB (Tim Bunce
                    <dbi-users@perl.org>)
    CPAN_VERSION    1.32
    CPAN_FILE       T/TI/TIMB/DBI-1.32.tar.gz
    DSLI_STATUS     MmcO (mature,mailing-list,
                    C,object-oriented)
    MANPAGE         DBI - Database-independent
                    interface for Perl
    INST_FILE       /opt/perl/lib/site_perl/5.6.1/
                    i386-freebsd/DBI.pm
    INST_VERSION    1.30

cpan> m DBD::Oracle
Module id = DBD::Oracle
    DESCRIPTION     Oracle Driver for DBI
    CPAN_USERID     TIMB(Tim Bunce
                    <dbi-users@perl.org>)
    CPAN_VERSION    1.12
    CPAN_FILE       T/TI/TIMB/DBD-Oracle-1.12.tar.gz
    DSLI_STATUS     MmcO (mature,mailing-list,C,
                    object-oriented)
    INST_FILE       (not installed)

cpan>
```

To search for modules, use the m command with a regular expression to find a list of matching module names:

```
cpan> m /^Sort/
Module  Sort::ArbBiLex      (S/SB/SBURKE/
                            Sort-ArbBiLex-3.4.tar.gz)
```

```
Module  Sort::Array          (M/MI/MIDI/
                             Sort-Array-0.26.tar.gz)
Module  Sort::ArrayOfArrays (E/EA/EARL/Sort-
                             ArrayOfArrays-1.00.tar.gz)
Module  Sort::Fields         (J/JN/JNH/
                             Sort-Fields-0.90.tar.gz)
Module  Sort::Naturally      (S/SB/SBURKE/
                             Sort-Naturally-1.01.tar.gz)
Module  Sort::PolySort       (Contact Author
                             Daniel Macks
                             <dmacks@netspace.org>)
Module  Sort::Versions       (E/ED/EDAVIS/
                             Sort-Versions-1.4.tar.gz)
Module  sort                 (J/JH/JHI/perl-5.8.0.tar.gz)
8 items found

cpan>
```

The CPAN shell has a great many other uses. You can find more documentation bundled with the CPAN module by using perldoc CPAN or man CPAN.

Configuring the CPAN Shell

The first time you run the CPAN shell, it will take you through a quick configuration process, which starts out like this:

```
[ziggy@chimay ~]$ perl -MCPAN -e shell

/home/ziggy/.cpan/CPAN/MyConfig.pm initialized.

CPAN is the world-wide archive of perl resources. It
consists of about 100 sites that all replicate the same
contents all around the globe. Many countries have at
least one CPAN site already. The resources found on
CPAN are easily accessible with the CPAN.pm module. If
you want to use CPAN.pm, you have to configure it
properly.

If you do not want to enter a dialog now, you can answer
'no' to this question and I'll try to autoconfigure. (Note:
you can revisit this dialog anytime later by typing 'o conf
init' at the cpan prompt.)

Are you ready for manual configuration? [yes]
```

I have found that the defaults are quite sensible. To save time, you can answer "no" to this prompt and accept all default values. The one value that cannot be intuited is the URL for your local CPAN mirror. If you do not know where to find your local CPAN mirror, then it is best to go through the manual configuration. You will then be presented a list of CPAN sites that are close to you geographically. The manual configuration is also a good idea if you need to use an FTP or HTTP proxy to connect to a CPAN mirror.

If you do know the URL of a local CPAN mirror, you can accept all of the other defaults (by typing "no" at the manual configuration prompt) and specifying it directly. This can be done with the o conf urllist command:

```
[ziggy@chimay ~]$ perl -MCPAN -e shell

cpan shell – CPAN exploration and modules installation (v1.63)
ReadLine support enabled

cpan> o conf urllist push http://www.cpan.org/

cpan>
```

You can see all of the options used by the CPAN shell through the o conf command. When you change the value of an option, it will be used for the duration of your shell session. To save these values permanently, use the o conf commit command:

```
cpan> o conf commit
commit: wrote /home/ziggy/.cpan/CPAN/MyConfig.pm

cpan>
```

Maintaining Multiple Perls

The CPAN module can be configured in two basic modes. When run as root, the default CPAN configuration will be site-wide and stored globally in the CPAN::Config module. Alternatively, you can use the CPAN shell as an unprivileged user, and the CPAN configuration will be stored as CPAN::MyConfig in your home directory (~/.cpan/CPAN/MyConfig.pm to be precise).

On my personal machines, I tend to have multiple versions of Perl installed. First is the version of Perl that comes with the operating system: FreeBSD 4.x ships with Perl 5.005_03 (released March 28, 1999), and MacOS X ships with Perl 5.6.0 (released March 23, 2000). Both of these versions have been superseded by Perl 5.6.1 (released April 9, 2001). I do most of my work with Perl 5.6.1 and do my best to leave the vendor-installed version of Perl alone.

Last summer, Jarkko Hietaniemi and the perl5-porters released Perl 5.8.0, a very major upgrade that includes many new and improved features. I am currently playing with some of these new features, and maintain this installation alongside my installation of 5.6.1. Keeping multiple releases of Perl around helps when I test to see if my programs will work with older versions.

I have also found that upgrading Perl versions where critical production programs are in use can be problematic. The last thing anyone wants to do is break a critical program by upgrading one of its dependencies (like Perl or some Perl modules). Unfortunately, this means that many developers are constrained to write and deploy software using an older version of Perl. Keeping multiple versions of Perl installed is one way to let programmers use the newer features available in newer releases of Perl when writing new programs without interfering with the

critical programs that are best left alone. This strategy also makes it easier to slowly migrate critical programs to newer versions of Perl in a controlled manner, or test a program against multiple releases of Perl.

Because I keep multiple versions of Perl on the same machine, I want to avoid using the CPAN shell as root. When the CPAN module is configured globally, each specific version of Perl must be configured individually. That is because the configuration stored in CPAN::Config is site-wide, but only for a specific Perl installation, so five Perl installations means that five CPAN::Config files need to be created and maintained.

The alternative is to use a user-level configuration, where the configuration is stored in my home directory. That way, CPAN::MyConfig can be configured once, and that configuration will be used by the CPAN shell with all versions of Perl I have installed:

```
[ziggy@chimay ~]$ /opt/perl/bin/perl5.6.1 -MCPAN -e shell
....

[ziggy@chimay ~]$ /opt/perl/bin/perl5.8.0 -MCPAN -e shell
....
```

Using the CPAN shell as an unprivileged user raises a minor issue. While I can download, extract, configure, build, and test a module, I cannot install it in the system-wide library. To do that, I need to have superuser permissions, just as I would with any other software install. This problem is easily solved by using sudo to run the CPAN shell. A judicious use of sudo can let multiple users maintain Perl installations on a single system, or expressly specify which users can maintain which specific Perl installations.

Testing Modules Locally

By default, the CPAN shell will aid the process in adding modules to the local site library. Usually, this is what you want to do.

What do you do when it is time to upgrade a critical module? Do you take the chance that the upgrade will not break anything, or do you test it first? There are many ways to solve this problem. The most troublesome and labor-intensive solution is to maintain a separate test machine for testing new Perl modules before they are deemed ready to install. A better alternative is to maintain a secondary "scratch" installation of Perl where modules can be installed and tested. This is very easy to do, requiring that one version of Perl be installed in two or more separate locations.

The easiest solution is to avoid the problem of updating the site-wide library for a Perl installation and just test modules in a local library area. To do this, I start by creating a "test" account that has very limited access to the system (no group memberships, no sudo access). I then set up a CPAN shell configuration specific to this user that will install all CPAN modules in the /home/test/lib directory. Finally, I tell Perl to look in this directory *before* looking in the site-wide module library areas, so that I can install and test both new modules as well as upgraded versions of previously installed modules.

Of course, there are many ways to tell Perl where to look for modules. The list of directories that contain Perl modules is stored in @INC. Here is one common idiom for adding a directory to the list of directories to search:

```
#!/usr/bin/perl -w

BEGIN {
    unshift(@INC, "/home/test/lib");
}

use strict;
....
```

That technique is rather opaque. Recent versions of Perl now include a use lib pragma to specify an alternate module directory. Using this pragma is preferable to using the old style BEGIN block:

```
#!/usr/bin/perl -w

use strict;
use lib '/home/test/lib';
....
```

If I can modify programs I want to test, the use lib technique will work. If I want to test a program that I cannot modify, or do not wish to modify, I can specify additional library paths when I invoke Perl. One approach is using the PERL5LIB environment variable. Another approach is to use the -I command line switch. I can see impact on the module search path (stored in @INC) simply by printing it out:

```
[test@chimay ~]$ perl   -I/home/test/lib \
>                       -e 'print join("\n", @INC), "\n"'
/home/test/lib/5.6.1/i386-freebsd
/home/test/lib/5.6.1
/home/test/lib/
/opt/perl/lib/5.6.1/i386-freebsd
/opt/perl/lib/5.6.1
/opt/perl/lib/site_perl/5.6.1/i386-freebsd
/opt/perl/lib/site_perl/5.6.1
/opt/perl/lib/site_perl
.
[test@chimay ~]$ PERL5LIB=/home/test/lib \
>                perl -e 'print join("\n", @INC), "\n"'
/home/test/lib/5.6.1/i386-freebsd
/home/test/lib/5.6.1
/home/test/lib/
/opt/perl/lib/5.6.1/i386-freebsd
```

```
/opt/perl/lib/5.6.1
/opt/perl/lib/site_perl/5.6.1/i386-freebsd
/opt/perl/lib/site_perl/5.6.1
/opt/perl/lib/site_perl
.
[test@chimay ~]$
```

Updating the CPAN Configuration

The next thing I need to do is update the CPAN configuration for the "test" user so that it installs modules in /home/test/lib. The best way to specify this is to use the PERL5LIB prefix, because the configure/build process will invoke some Perl subprocesses. If I specify the library path using -I, then the CPAN shell will find modules installed there, but the subprocesses will not. Using the PERL5LIB environment variable fixes this problem.

I start by logging in as "test" and invoking the CPAN shell so that it can find my local module directory:

```
[test@chimay ~]$ PERL5LIB=/home/test/lib perl -MCPAN -e shell
cpan>
```

The vast majority of Perl module distributions are built so that they can be installed easily with CPAN.pm. The process is quite simple (and easily automated by the make and install commands). The configuration process (perl Makefile.PL) generates a makefile that will be used to build and install a module. This auto-generated makefile provides many options for overriding the default configuration parameters.

One of the makefile parameters that can be configured is the PREFIX variable, which defines the root directory of a Perl installation. By setting this variable to /home/test when generating the makefile, the install process will install Perl modules under /home/test/lib, man pages under /home/test/man, and programs under /home/test/bin.

Another parameter that can be configured is the INSTALLDIRS variable. This specifies one of three possible areas where modules can be installed: perl, site, or vendor. In a stock Perl configuration, modules installed under the perl directory go into /usr/local/lib, modules installed under the site directory go into /usr/local/lib/site_perl, and the vendor directory is unused.

Because we may be installing upgrades to Perl core modules, it's important to set the INSTALLDIRS variable when generating a makefile so that all modules go into the same local directory. For convenience, I set this variable to perl, so that all modules will be installed in /home/test/lib instead of /home/test/lib/. Updating the makepl_arg option to specify these two configuration parameters is simple:

```
cpan> o conf makepl_arg "PREFIX=/home/test
                                INSTALLDIRS=perl"
      makepl_arg     PREFIX=/home/test INSTALLDIRS=perl

cpan> o conf commit
commit: wrote /home/test/.cpan/CPAN/MyConfig.pm

cpan>
```

Now, the CPAN shell is configured to find modules in /home/test/lib and install new modules in that location.

Any instance of this version of Perl that starts up with -I/home/test/lib specified (or PERL5LIB=/home/test/lib) will use the modules installed in this location. Alternatively, any program that contains a use lib '/home/test/lib'; declaration will find the upgraded version of the CGI module. All other programs will find the previously installed version.

Conclusion

The CPAN shell is a very useful tool for installing Perl modules on your system. Although it is typically used for managing one Perl installation on a system, it can be used to help maintain multiple parallel Perl installations. The CPAN shell can also be used to aid in testing and evaluating Perl modules before installing them system-wide.