Perl for Systems Administrators

Simon Cozens

Perl for Systems Administrators
by Simon Cozens

Table of Contents

List of Tables

List of Examples

Chapter 1. Preliminaries

Welcome to NetThink's *Perl for Systems Administrators* training course. This course provides an introduction to Perl by means of examples suitable for Unix systems administrators in their daily work.

1.1. Course Outline

- Introducing Perl
- Perl on the Command Line
- Perl and Unix Database
- Networking and Web Automation
- Win32 Administration
- Larger Projects

1.2. Assumed Knowledge

This course requires no knowledge of Perl programming, but attendees will benefit from the ability to learn by example. This course can be used as a substitute for NetThink's *Introduction to Perl* course.

1.3. Objectives

The aim of this course is to introduce you to programming in Perl by means of examples which are intended to be relevant to your daily work. It intends to teach good programming practices such as the use of CPAN modules, as well as how Perl can be useful for Win32 administration, web automation, and for quick command-line tasks.

1.4. The course notes

These course notes contain material which will guide you through the topics listed above, as well as appendices containing other useful information.

The following typographic conventions are used in these notes:

System commands appear in **this typeface**

Literal text which you should type in to the command line or editor appears as `monospaced font`.

Keystrokes which you should type appear like this: **ENTER**. Combinations of keys appear like this: **CTRL**-**D**

```
Program listings and other literal listings of what appears on the
screen appear in a monospaced font like this.
```

Parts of commands or other literal text which should be replaced by your own specific values appears *`like this`*

> **Note:** Notes and tips appear offset from the text like this.

Notes which are marked "Advanced" are for those who are racing ahead or who already have some knowledge of the topic at hand. The information contained in these notes is not essential to your understanding of the topic, but may be of interest to those who want to extend their knowledge.

Notes marked with "Readme" are pointers to more information which can be found in your textbook or in online documentation such as manual pages or websites.

Chapter 2. Introducing Perl

2.1. What is Perl?

2.2. Example 1: Summarising Apache Vhost Configuration

Recently, we had to rearrange our web servers; as we migrated virtual hosts around, we needed to know the document roots for each vhost. Here's what a colleague came up with.

Example 2-1. Summarising Apache Vhost Configuration

```
while (<>) {                                                    ❶
    next unless /^\s*<VirtualHost\s+([^>\s]+)\s*>/;
    my $vhost = $1;
    while (<>) {
        if (m|\s*</VirtualHost|) {
            die "Could not find document root for $vhost\n";
        }
        if (/^\s*DocumentRoot\s*([^\s]+)/) {
            print "$vhost $1\n";
            last;
        }
    }
}
```

2.3. Example 2: Cluster node list

Our mail cluster has several nodes in several different classes; a frontend, a web server, a mail server, etc. Some machines are in multiple classes. When we're performing administration tasks, we might want to run a command on all machines in a particular class. Given a configuration file listing the node name and its classes, we can use this:

Example 2-2. Cluster node list

```
my $CLUSTER_CONF_PATH = "/etc/cluster.conf";                ❶

if (@ARGV > 1 || $ARGV[0] =~ /^-/) {
    print STDERR "Usage: clist [class]\n";
    exit 2;
}
my $class = $ARGV[0];

open(CONF, $CLUSTER_CONF_PATH) or die "$CLUSTER_CONF_PATH: $!\n";
while (<CONF>) {
    chomp;
    s/#.*//;
    next unless /\S/;

    my ($key, $host, @classes) = split(' ');
    next unless $key eq "node";

    if (!defined($class) || grep($_ eq $class, @classes)) {
        print "$host\n";
    }
}
```

2.4. Example 3: Killing User's Processes

Sometimes we want to kill off a user's mail processes. Here's what we do:

Example 2-3. Killing User's Processes

```
use strict;                                                 ❶

if ($> == 0) {
    my $uid = getpwnam("www-admin") or die "can't find uid for www-admin\n";
    ($<, $>) = ($uid, $uid);
    die "setuid($uid) for www-admin failed: $!\n" unless $< == $uid && $> == $u
d;
}

die "Usage: killwingproc username ...\n"
    unless @ARGV;

my %users = map { $_ => 1 } @ARGV;
```

```
open(PS, "ps axw|") or die "ps: $!\n";
while (<PS>) {
    if (/^\s*(\d+).*maild \((\w+)\)/ && $users{$2}) {
    kill("TERM", $1);
    }
}
```

2.5. Example 4: Mark up plain text as HTML

Here's an example from Tom Christiansen of how to surround things which look like URLs with HTML A tags. We used to try to pretend that it's a useful utility, rather than a cheap example of complicated regular expressions, but we don't bother any more.

Example 2-4. Mark up plain text as HTML

```
#!/usr/bin/perl                                                    ❶
# urlify - wrap HTML links around URL-like constructs

$urls = '(http|telnet|gopher|file|wais|ftp)';
$ltrs = '\w';
$gunk = '/#~:.?+=&%@!\-';
$punc = '.:?\-';
$any  = "${ltrs}${gunk}${punc}";

while (<>) {
    s{
      \b
      (
       $urls     :
       [$any] +?

      )
      (?=
       [$punc]*
       [^$any]
       |
       $
      )
     }{<A HREF="$1">$1</A>}igox;
    print;
}
```

2.6. Example 5: Keeping Up To Date With Perl 5

When I'm making patches to Perl 5, I like to keep my work in Perforce version control; however, I also want to keep up with the changes as they happen. So periodically, I run the following program, which integrates the changes others have made with those I'm in the middle of.

Example 2-5. Keeping Up To Date With Perl 5

```
#!/usr/bin/perl                                                    ❶
chdir("/home/simon/patchbay") or die $!;

# First get the list of changes:
print "rsyncing.\n";
@files =‘rsync -ancvz rsync://ftp.linux.activestate.com/perl-current/
  jarkkoperl/‘;
$ENV{P4CLIENT} = "simon";
$ENV{EDITOR} = "vi";
shift @files until ($files[0] =~ /^receiving file list/); shift
  @files;
pop @files until $files[-1] =~ /^wrote \d+ bytes/; pop @files;
chomp(@files);
$_="jarkkoperl/".$_ for @files;
print "Files to edit: @files\n";
exit unless @files;
print "p4 edit @files\n";
system("p4 edit @files");
system("rsync -acvz rsync://ftp.linux.activestate.com/perl-current/
  jarkkoperl/");
open(SUBMIT, "|p4 submit -i") or die $!;
print SUBMIT form("rsync", @files);
close SUBMIT;

s/jarkkoperl/bleadperl/ for @files;

system("p4 integrate -r -b jarkkoperl");
system("p4 resolve -at");
open(SUBMIT, "|p4 submit -i") or die $!;
print SUBMIT form("Integrate with Jarkko", @files);
close SUBMIT;

sub form {
    my $desc = shift;
    $form = ‘p4 change -o‘;
    $form =~ s/<enter description here>/$desc/;
    return $form;
}
```

Chapter 3. Perl on the Command Line

3.1. Fundamental Switches

3.2. Example 6: Global search-and-replace

In my latest book, I took to marking up functions in the API as `api.function_name`. Unfortunately, this was ill-formed SGML; you're not allowed an underscore in identifiers. So I had to run through the whole text of the book changing `api.function_name` to `api.function.name`. Perl to the rescue!

Example 3-1. Global search-and-replace

```
perl -pi -e 's/api\.(\w+)_/api.$1./g' api/*
```

3.3. Example 7: Adding Columns

The `-a` flag autosplits to the `@F` array, and so can be used like awk:

Example 3-2. Adding Columns

```
perl -lane 'print $F[0] + $F[-2]'
```

3.4. Example 8: Watching SSH Logs

Want to know who's connecting to your SSH server? This little one-liner looks for information from **sshd** and prints out the IP address of people connecting.

Example 3-3. Watching IPPL Logs

IPPL is a daemon which watches for connections to TCP/IP ports and logs them to syslog. Let's suppose we're interested in who's making ssh connection attempts, and want to know their hostnames.

```
tail -f /var/log/ippl/all.log |
perl -MSocket -lne
```

```
 'print scalar gethostbyaddr(inet_aton($1),2) if
   /ssh/ && /connection attempt from\D*([\d.]+)/'
```

3.5. Example 9: MP3 Identification

An extreme example of command-line Perl came up recently when rearranging my MP3 collection...

Example 3-4. MP3 Identification

```
find . -name '*mp3' | perl -nle '                          ❶
   $orig=$_;
   tr/_/ /;
   s/..//;
   s/pizzicato five\s*(-\s*)?//i;
   s|^([^/]+)/||;
   $album=$1;
   s/.mp3//;
   s/.*?\d+(\s*-\s*)?//;
   $orig=~tr/ /?/;
   system(qq|id3 -t "$_" $orig|);
   system(qq|id3 -a "Pizzicato Five" $orig|);
   system(qq|id3 -A "$album" $orig|)'
```

3.6. Example 10: TCP/IP client

We'll see in the "networking" section how to construct a simple TCP/IP server. For now, here's the client which talks to it. It sends the text found on the command line, and then prints out the reply it gets from the server.

Example 3-5. Simple TCP/IP client

```
perl -MIO::Socket -e '$r=IO::Socket::INET->new(PeerAddr => "rt:9000")
or die "cannot connect"; print $r "@ARGV\n"; while (<$r>){print}
```

Chapter 4. Perl and Unix Databases

4.1. Perl's Built-In Unix Functions

4.2. Example 11: Summarising the Password Database

To consolidate usernames for various different systems, we take a snapshot of the password database on each system and compare the results. Summarising the password database is just a few lines of code, using Perl's built-in Unix functions.

Example 4-1. Summarising the Password Database

```
# Ignore UIDs outside this range                          ❶
my $minuid = 1000;
my $maxuid = 60000;

setpwent;

while (my ($name, $uid, $gid, $home) = (getpwent)[0,2,3,7]) {
    next unless $uid >= $minuid && $uid <= $maxuid;
    print "$name $uid $gid $home\n";
}

endpwent;
```

4.3. Example 12: Checking a User's Password

Here's another example from the *Perl Cookbook* which helps to ensure that a user is who you think they are, by prompting for their password.

Example 4-2. Checking a User's Password

```
                                                          ❶
use Term::ReadKey;
```

```
print "Enter your password: ";
ReadMode 'noecho';
$password = ReadLine 0;
chomp $password;
ReadMode 'normal';

print "\n";

($username, $encrypted) = ( getpwuid $< )[0,1];

if (crypt($password, $encrypted) ne $encrypted) {
    die "You are not $username\n";
} else {
    print "Welcome, $username\n";
}
```

4.4. Example 13: Group to LDIF

For migrating Unix group information to an LDAP database, the following script due to Jeremy Mates will do the trick.

Example 4-3. Group to LDIF

```
use strict;                                                    ❶

while (my ($name,$passwd,$gid,$members) = getgrent()) {
    print "dn: cn=$name\n";
    print "cn: $name\n";
    print "objectclass: posixGroup\n";

    # default to crypt, but also handle md5,blowfish crypts
    # in RFC 2307 fashion
    my $pw_hash = 'crypt';
    $pw_hash = 'md5' if q/$1$/ eq substr $passwd, 0, 3;
    $pw_hash = 'altscheme' if q/$2$/ eq substr $passwd, 0, 3;

    print "userPassword: {$pw_hash}$passwd\n";
    print "gidnumber: $gid\n";
    print "memberuid: $_\n" for split /\s+/, $members;
    print "\n";
}
```

4.5. Example 14: Moving User's Directories

As part of the web server migration mentioned in our very first example, we needed to go through each user's home directory, and relink the `public_html` directories to their new location on the filesystem. We created a handy module called `AccountHash` to return the new-style path for a given user, and then came up with a program like this:

Example 4-4. Moving User's Directories

```
use AccountHash qw(hashpath);                                    ❶
use User::pwent; # Make getpwent sane

while ($_ = getpwent) {
    next if $_->uid < 1000;
    next unless -d $_->dir."/public_html" or -d $_->dir."/cgi";
    print "rm ".$_->dir."/public_html ".$_->dir."/cgi\n";
    print "ln -s /webnfs/web/users".hashpath($_->name)."/public_html ".
        $_->dir."/public_html\n";
    print "ln -s /webnfs/web/users".hashpath($_->name)."/cgi ".
$_->dir."/cgi\n";
};
```

4.6. Example 15: Moving a User's Group

Occasionally, we have to move a user from one group to another; since the filestore is arranged by group, this means also moving their home directory and so on. Here's an abridged example of the utility we use to do this.

Example 4-5. Moving a User's Group

```
use Fcntl qw(:DEFAULT :flock);                                   ❶
use strict;

my $lockfile = "/var/lock/webadmind.LCK";
my $filesystem = "/w0";

die "Usage: moveuser name oldgroup newgroup newgid\n"
```

```
    unless @ARGV == 4;

my ($name, $oldgroup, $newgroup, $newgid) = @ARGV;

sysopen(LOCK, $lockfile, O_CREAT|O_RDWR, 0600)
    or die "Can't open $lockfile: $!";

# Take out an exclusive lock, blocking if necessary
flock(LOCK, LOCK_EX) or die "flock failed: $!\n";

die "bad name: $name\n" unless $name =~ /^\w{1,32}$/;
die "bad oldgroup: $oldgroup\n" unless $oldgroup =~ /^\w{1,32}$/;
die "bad newgroup: $newgroup\n" unless $newgroup =~ /^\w{1,32}$/;
die "bad newgid: $newgid\n" if $newgid =~ /\D/;
die "bad newgid: $newgid does not match newgroup\n"
    unless $newgid == getgrnam($newgroup);

my ($uid, $gid, $gecos, $home, $shell) = (getpwnam($name))[2,3,6,7,8]
    or die "No such username: $name\n";
my $oldgid = getgrnam($oldgroup)
    or die "bad oldgroup: $oldgroup does not exist\n";
die "user $name is already in group $newgroup\n"
    if $oldgid == $newgid;
die "bad oldgroup: $oldgroup does not match current group for $name\n"
    unless $gid == $oldgid;

my $newhome = "$filesystem/$newgroup/$name";
die "$newhome already exists\n" if -e $newhome;

#
# Move files to new home
#
rename($home, $newhome) or die "rename $home $newhome failed: $!\n";

# Fix up gids of files
system("chgrp -R $newgid $newhome/public_html $newhome/cgi");
die "recursive chgrp failed\n" if $? >> 8;
```

Chapter 5. Network and Web Automation

5.1. Modules for Network Programming

5.2. Example 16: Simple TCP Daemon and Client

Here’s a TCP/IP server which talks to an RT trouble ticketing system:

Example 5-1. RT Daemon

```
#!/usr/bin/perl -w                                              ❶
use strict;
use Text::Wrapper;
use strict;
use Carp;
use Getopt::Long;
use Authen::Libwrap qw(hosts_ctl STRING_UNKNOWN);

use lib "/opt/rt2/lib";
use lib "/opt/rt2/etc";

# RT stuff here:
use RT::Interface::CLI  qw(CleanEnv LoadConfig DBConnect
                           GetCurrentUser GetMessageContent);
use RT::Queues;
use RT::Tickets;
CleanEnv();
LoadConfig();
DBConnect();
RT::DropSetGIDPermissions();
my $CurrentUser = $RT::SystemUser;

# Ordinary networking stuff here:
use IO::Socket;
my $PORT = 9000;                 # pick something not in use

my $server = IO::Socket::INET->new( Proto     => ’tcp’,
                                    LocalPort => $PORT,
                                    Listen    => SOMAXCONN,
                                    Reuse     => 1);

die "can’t setup server" unless $server;
```

```
$|=1;
#print "[Server $0 accepting clients]\n";

while (my $client = $server->accept()) {
    $client->autoflush(1);
    my $client_ip = $client->peerhost();
    my $client_addr = $client->peeraddr();

    my $client_name = gethostbyaddr($client_addr, AF_INET);
    unless ($client_name) {
        print $client "Couldn't find you in DNS\n";
        $client->close;
        next;
    }

    unless (hosts_ctl("rtd", $client_name, $client_ip, STRING_UNKNOWN)) {
        print $client "Couldn't authenticate $client_name / $client_ip\n";
        $client->close;
        next;
    }
    my $command = <$client>;
    print $client $_,"\n" for respond($command);
}

sub respond  {
    # ... complex interaction with RT here ...
}
```

5.3. Example 17: Web Page Mirrorer

Here's an example of what you can do with some CPAN modules if you're in a hurry; we can dump a recursive mirror of a web site with the `WWW::SimpleRobot` module and a little bit of glue. For a more sophisticated version, see the `mirror` program in the CPAN scripts repository.

```
use File::Path;                                          ❶
use File::Basename;
use WWW::SimpleRobot;

my $robot = WWW::SimpleRobot->new(
    URLS            => [ 'http://www.perl.org/' ],
    FOLLOW_REGEX    => "^http://www.perl.org/",
    DEPTH           => 5,
    TRAVERSAL       => 'depth',
```

```
    VISIT_CALLBACK  =>
        sub {
             my ( $url, $depth, $html, $links ) = @_;
     print "Visiting $url\n";
             $url =~ s|^.*://||;
             mkpath(dirname($url), 0, 0777);
             open(OUT, ">$url") or die $!;
             print OUT $html;
             close OUT;
        }
);
$robot->traverse;
```

5.4. Example 18: Web Link Checker

Here's an example taken from *The Perl Cookbook*, which uses some of the web automation modules to check the links in a document.

Example 5-2. Web Link Checker

```
#!/usr/bin/perl -w                                                    ❶
# churl - check urls

use HTML::LinkExtor;
use LWP::Simple qw(get head);

$base_url = shift
    or die "usage: $0 <start_url>\n";
$parser = HTML::LinkExtor->new(undef, $base_url);

$parser->parse(get($base_url));
@links = $parser->links;
print "$base_url: \n";

foreach $linkarray (@links) {
    my @element  = @$linkarray;
    my $elt_type = shift @element;
    while (@element) {
        my ($attr_name , $attr_value) = splice(@element, 0, 2);
        if ($attr_value->scheme =~ /\b(ftp|https?|file)\b/) {
            print "  $attr_value: ", head($attr_value) ? "OK" : "BAD",
    "\n";
        }
    }
}
```

5.5. Example 19: HTML Processor and FTP Uploader

My personal website is statically generated from a set of templates, using the `HTML::Mason` templating utility. Mason is meant to be used dynamically, but at the moment I can only upload static HTML to my ISP via FTP. So I write a little utility to process the templates, then automatically upload the new files.

Example 5-3. HTML Processor and FTP Uploader

```
use HTML::Mason;                                                    ❶
use Net::FTP;
use File::Find;
use File::Path;
$ftp = Net::FTP->new("simon-cozens.org", Debug => 0);
$ftp->login("simon",$password) or die $@;

my $template_dir = "/home/simon/website/templates";
my $outdir = "/home/simon/website/output";
my $ftpdir = "public_html/";
my $base = "http://simon-cozens.org/";
my $parser = new HTML::Mason::Parser;
my $interp = new HTML::Mason::Interp (parser=>$parser,
                                      comp_root=>$template_dir,
                                      data_dir=>"/home/simon/website/data",
                                      out_method=>\$outbuf);

find (sub {
    $outbuf="";
    $_ =  $File::Find::name;
    return unless /.html$/;
    s/$template_dir//;
    $outfile = $outdir. $_;
    $ftpfile = $ftpdir .$_;
    my $dir = $outfile;
    if ( -f $outfile and -M $outfile < -M $template_dir.$_) {
        print "Skipping $_, output is newer\n";
        return;
    }
    $dir =~ s|[^/]+$||;
    unless (-d $dir) {
        print "Making $dir\n";
        mkpath $dir;
    }
    print "Processing $_ as $outfile. \n";;
    @params = ( $_, "Base"   => $base, "WhoAmI" => $_);
```

```
    my $retval = $interp->exec(@params);
    open FH, ">$outfile" or die "Couldn't create $outfile: $!";
    print FH $outbuf;
    close FH;
    $ftp->put($outfile, $ftpfile);
    },
$template_dir);
```

5.6. Example 20: XML-RPC Client/Server

Just to prove that Perl can keep up with all the latest fads in network programming:

Example 5-4. XML-RPC Server

```
use Frontier::RPC;                                          ❶
sub sumAndDifference {
    my ($x, $y) = @_;

    return {'sum' => $x + $y, 'difference' => $x - $y};
}

$methods = {'sample.sumAndDifference' => \&sumAndDifference};
Frontier::Daemon->new(LocalPort => 8080, methods => $methods)
    or die "Couldn't start HTTP server: $!";
```

Easy, really. And here's the accompanying client:

Example 5-5. RPC-XML Client

```
   use Frontier::Client;                                    ❶

   $server_url = 'http://localhost:8080/';
   $server = Frontier::Client->new(url => $server_url);

   $result = $server->call('sample.sumAndDifference', 5, 3);
   $sum = $result->{'sum'};
   $difference = $result->{'difference'};
```

Chapter 6. Windows Administration with Perl

6.1. Windows Administration Modules

There are a few things that, if you're venturing into the scary world of Windows administration, you should not leave CPAN without:

- `Win32::AdminMisc`
- `Win32::NetAdmin`
- `Win32::EventLog`
- `Win32::TieRegistry`

6.2. Example 21: Accessing the Event Log

Operating system logs are fundamental to any system administration, but Windows' logs aren't easy to get hold of. It would be much nicer if we could get the Event Log into something that looked a bit like syslog.

Example 6-1. Accessing the Event Log

```
use Win32::EventLog;                                    ❶
%type = (1  => "ERROR",
         2  => "WARNING",
         4  => "INFORMATION",

         8  => "AUDIT_SUCCESS",
         16 => "AUDIT_FAILURE");

$Win32::EventLog::GetMessageText = 1;

# open the System event log
$log = new Win32::EventLog("System")
  or die "Unable to open system log:$^E\n";

# read through it one record at a time, starting with the first entry
my $read_type = EVENTLOG_SEQUENTIAL_READ|EVENTLOG_FORWARDS_READ;
while ($log->Read($read_type, 1,$entry)){
    print scalar localtime($entry->{TimeGenerated})." ";
    print $entry->{Computer}."[".($entry->{EventID} & 0xffff)."] ";
    print $entry->{Source}.":".$type{$entry->{EventType}}.":";
```

```
    print $entry->{Message};
}
```

6.3. Example 22: Starting and Stopping Services

Services are the Win32 equivalent of daemons. So what do you do without `init.d`?

Example 6-2. Starting and Stopping Services

```
use Win32::Service;                                          ❶
use Win32;
my %status;
my $what = shift;
my $state = shift;
Win32::Service::GetStatus( ’’,$what, \%status);
if ($state eq "start") {
    print "Starting $what: ";
    die "already started\n" if ($status{CurrentState} == 4);
    Win32::Service::StartService(Win32::NodeName( ),$what)
      || die "Can’t start service\n";
elsif ($state eq "stop") {
    print "Stopping $what: ";
    die "not running\n" if ($status{CurrentState} == 1);
    Win32::Service::StartService(Win32::NodeName( ),$what)
      || die "Can’t stop service\n";
} else {
    die "Usage: init service start|stop\n";
}
print "$what\n";
```

6.4. Example 23: Keeping a Network’s Clocks in Sync

On Unix, we’ve got **cron** and **ntpd**. But what if you’ve got a network of Windows servers that you want to keep in sync? Here’s a script which originally appeared in the *Perl Journal*:

Example 6-3. Keeping a Network's Clocks in Sync

```
use Win32::AdminMisc;                                          ❶
use Win32::NetAdmin;

srand(time);
$Domain = "SOUTH_PARK";
$Server = "\\\\ServerName";
Win32::NetAdmin::GetServers($Server, $Domain, SV_TYPE_ALL, \@List);

foreach $Machine (@List) {
    $Time = sprintf("3:%02d am", int(rand(60)));
    if (Win32::AdminMisc::ScheduleAdd($Machine,
                $Time,
                0,
        SUNDAY,
                JOB_RUN_PERIODICALLY | JOB_NONINTERACTIVE,
                "net time \\\\server /s /y") {
print "Job added to $Machine for $Time every Sunday.\n";
    } else {
print "Job not added to $Machine.\n";
    }
}
```

6.5. Example 24: Dumping the Registry

Here's a human-readable dump of the Windows registry which can be re-evaluated into a Perl variable to restore the registry in the future.

Example 6-4. Dumping the Registry

```
use Win32::TieRegistry::Dump;
use Data::Dumper;
print Dumper(Win32::TieRegistry::toArray("LMachine"));
```

6.6. Example 25: Starting Perl on Win32

If you're writing Perl background processes and the like on Windows, you'll notice that Windows keeps popping up a DOS box when you run the program. This isn't optimal, so we can get around it - using Perl, of course. Another example from *The Perl Cookbook*.

Example 6-5. Starting Perl on Win32

```
use Win32;                                                    ❶
use Win32::Process;

# Create the process object.

Win32::Process::Create($Win32::Process::Create::ProcessObj,
    'C:/perl5/bin/perl.exe',            # Whereabouts of Perl
    'perl realprogram',                 #
    0,                                  # Don't inherit.
    DETACHED_PROCESS,                   #
    ".") or                             # current dir.
die Win32::FormatMessage( Win32::GetLastError() );
```

Chapter 7. Larger Projects

7.1. Example 26: NSS Database Tool

Example 7-1. NSS Database Tool

```
#!/usr/bin/perl                                                    ❶
use DB_File;
use Fcntl;
use strict;

my $PW_DB_PATH = "/var/db/passwd.db";
my $SH_DB_PATH = "/var/db/shadow.db";

my $batchmode;

if ($ARGV[0] =~ /batch/ or $ARGV[0] eq "-b") {
    $batchmode = 1;
    shift @ARGV;
}

my %commands;

{ no strict 'refs';
 %commands = map {$_ => \&$_} qw(
    adduser removeuser addshadow removeshadow mod
);
}
if (not exists $commands{$ARGV[0]}) {
    die <<EOF;

$0: invalid command $ARGV[0].

Valid commands are : @{[ keys %commands ]}

EOF
}

$commands{shift @ARGV}->();

=head1 NAME

nssdbadmin - Administration script for NSS databases

=head1 SYNOPSIS
```

```
    nssdbadmin adduser      username uid gid gecos home shell
    nssdbadmin removeuser   username
    nssdbadmin addshadow    username
    nssdbadmin removeshadow username
    nssdbadmin mod          username [-gecos newgecos | -shell newshell | -home newhome ]
    nssdbadmin info         username
    nssdbadmin enable       username
    nssdbadmin disable      username

    # Batch mode
    nssdbadmin -b adduser
    nssdbadmin -b removeuser
    nssdbadmin -b addshadow
    nssdbadmin -b removeshadow

=head1 DESCRIPTION

Administers NSS DB password and shadow password databases.

NSS is the Linux Name Service Switch. See:

    info libc 'Name Service Switch'

Briefly, you can use it to replace the password and shadow files by
Berkeley databases. It's the same mechanism by which NIS can be used for
user authentication, but this utility helps you create Berkely DB based
NSS databases.

=head1 COMMANDS

=head2 adduser

    adduser username uid gid gecos home shell

Add a user to the passwd database. In batch mode, passwd file entries
are taken in on standard input and added to the database. A warning is
issued if the entry cannot be added.

=cut

sub adduser {
    # Open the passwd db
    my ($new_db_ref, $old_db_ref) = open_db($PW_DB_PATH);
    my @entries;
    my @ent;
    if ($batchmode) {
        while (<STDIN>) {
            chomp;
            my @ent = split /:/, $_;
            splice (@ent,1,1); # Remove password field.
            push @entries, [@ent] if @ent = _check_new_ent($old_db_ref, @ent);
        }
```

```
    } else {
        push @entries, [@ent] if @ent = _check_new_ent($old_db_ref, @ARGV)
    }
    my $n = db_parse($old_db_ref, $new_db_ref, 1, sub { 0 } ); # Copy old to new

    my $ent_r;
    while ($ent_r = pop @entries) {
        my ($username, $pass, $uid, $gid, $gecos, $home, $shell) = @$ent_r;
        $new_db_ref->{"0$n"} =
            $new_db_ref->{".$username"} =
            $new_db_ref->{"=$uid"} = join(":",@$ent_r)."\0";
        $n++;
    }
    untie %$new_db_ref; rename "$PW_DB_PATH.new", $PW_DB_PATH;
    # chmod $PW_DB_PATH, 0644;
}

sub _check_new_ent { # Check a new entry and return it
    my ($db_ref, $username, $uid, $gid, $gecos, $home, $shell) = @_;
    # Check the args, eh?
    return gurgle("No username specified") unless $username;
    return gurgle("Invalid username $username")      unless $username =~ /^\w{0,8}/;
    return gurgle("Invalid UID $uid")           unless $uid+0 >= 1000;
    return gurgle("Invalid GID $gid")           unless $gid+0 >= 100;
    # The rest don't really matter.

    return gurgle("Username $username already exists") if exists $db_ref->{".$username"};
    return gurgle("UID $uid already exists")      if exists $db_ref->{"=$uid"};
    return ($username, "x", $uid, $gid, $gecos, $home, $shell);
}

=head2 addshadow

Add a user to the shadow database. Similar to the above. In command-line
mode, the only parameter is the username: defaults appropriate to our
local system are used. In the future, this will be replaced by something
more generic. For now, use C<mod> to alter the other fields. Or use
batch mode, which takes shadow(5)-style lines from standard input.

=cut

sub SECS_PER_DAY () { 60 * 60 * 24 }
sub TEN_YEARS_IN_DAYS () { 10 * 365 } # close enough
sub INITIAL_EXPIRY_DAYS () { 30 } # new users must change password within this

sub addshadow {
    my ($new_db_ref, $old_db_ref) = open_db($SH_DB_PATH);
    my @entries;
    if ($batchmode) {
        chomp(@entries = <STDIN>);
        $_.="\0" for @entries;
    } else {
        my $cur_days = int(time / SECS_PER_DAY);
```

```
        my $lstchg = $cur_days - TEN_YEARS_IN_DAYS + INITIAL_EXPIRY_DAYS;
        my $max = TEN_YEARS_IN_DAYS;
        @entries = "$ARGV[0]:*:${lstchg}:0:${max}::0::\0";
    }
    my $n = db_parse($old_db_ref, $new_db_ref, 0, sub { 0 } ); # Copy old to new

    for (@entries) {
        my $username = ((split /:/,$_)[0]);
        $new_db_ref->{"0$n"} =
            $new_db_ref->{".$username"} = $_;
        $n++;
    }
    untie %$new_db_ref; rename "$SH_DB_PATH.new", $SH_DB_PATH;
}

=head2 removeuser

Removes a user from the password database. Pass in a username only. In
batch mode, this will spit out the user's old passwd entry onto
standard output, in a form suitable for feeding back into C<adduser>.

The idea is you'd be able to say

    nssdbadmin -b removeuser < tomove | ssh otherhost 'nssdbadmin -b adduser'

to move users from one server to another. That's not for the faint at
heart, though.

=cut

sub removeuser {
    my ($new_db_ref, $old_db_ref) = open_db($PW_DB_PATH);
    my %togo;
    if ($batchmode) {
        while (<STDIN>) { chomp; $togo{$_} = 1 };
    } else {
        $togo{$ARGV[0]} = 1;
    }
    db_parse($old_db_ref, $new_db_ref, 1, sub {
        my ($oldref, $newref, $user, $passwd, $uid, $rest) = @_;
        if (exists $togo{$user}) {
            print "$user:$passwd:$uid:$rest\n" if $batchmode;
            return 1;
        }
        return 0;
    });
    untie %$new_db_ref; rename "$PW_DB_PATH.new", $PW_DB_PATH;
    #chmod $PW_DB_PATH, 0644;
}

=head2 removeshadow

Removes a data from the shadow database. Similar to the above in every
```

```
respect apart from the database (obviously) and the format of the
output.

=cut

sub removeshadow {
    my ($new_db_ref, $old_db_ref) = open_db($SH_DB_PATH);
    my %togo;
    if ($batchmode) {
        while (<STDIN>) { chomp; $togo{$_} = 1 };
    } else {
        $togo{$ARGV[0]} = 1;
    }
    db_parse($old_db_ref, $new_db_ref, 0, sub {
        my ($oldref, $newref, $user, $passwd, $uid, $rest) = @_;
        if (exists $togo{$user}) {
            print "$user:$passwd:$uid:$rest\n" if $batchmode;
            return 1;
        }
        return 0;
    });
    untie %$new_db_ref; rename "$SH_DB_PATH.new", $SH_DB_PATH;
}

=head2 mod

Modifies the user record; you may modify the following items:

    -gecos
    -home
    -shell
    -gid
    -passwd
    -expiry
    -lastchange
    -maxdays

=cut

sub mod {
    my $user = shift;
    die "Incorrect number of arguments to mod!\n"
        if @ARGV & 1; # Deliberate obfuscation to scare Ray.

    my %options = @ARGV;
    my %possible_options;
    my %formats;
    $possible_options{user} = [qw(-gecos -home -shell -gid)];
    $possible_options{shadow} = [qw(-gecos -home -shell -gid)];
    $formats{user}   = [qw(user pwd uid gid gecos home shell)];
    $formats{shadow} = [qw(user pass lastchange maychange mustchange expire disabledate res

    for my $db (qw(user shadow)) {
```

```
        if (grep { defined $options{$_} } @{$possible_options{$db}}) {
            my $file = $db eq "user" ? $PW_DB_PATH : $SH_DB_PATH;
            my ($new_db_ref, $old_db_ref) = open_db($file);
            my $rec = $old_db_ref->{".$user"} or die "No such username: $user\n";
            my %rec;
            @rec{@{$formats{$db}}} = split /:/, $rec;

            # Change if it needs changing.
            defined $options{"-$_"} and $rec{$_} = $options{"-$_"}
               for map {my $x; ($x=$_)=~s/-//; $x} @{$possible_options{$db}};

            # Out with the old.
            my $ix = db_parse($old_db_ref, $new_db_ref, $_ eq "user", sub { return $_[2] eq

            # And in with the new.
            $new_db_ref->{"0$ix"} = $new_db_ref->{".$user"} = join(":", @rec{@{$formats{$db
            untie %$new_db_ref; rename("$file.new", $file);
            #chmod $file, $db eq "user" ? 0644 : 0600;
        }
    }
}

=head2 info

Shows information about the account. Oops. This doesn't appear to be
implemented. There's probably supposed to be a C<disable> and an
C<enable> function here too.

=cut

# Auxilliary functions

sub gurgle { if ($batchmode) { warn "@_\n" } else { die "@_\n" } }

sub db_parse {
    my ($oldref, $newref, $is_user, $callback) = @_;

    # OK. We loop over the database by sequence number, calling the callback
    # for each record. If the callback returns false, we copy the record
    # from the old database into the new database. Otherwise we assume that
    # the callback has dealt with it.

    my $newix = 0;
    for (my $oldix = 0; my $rec = $oldref->{"0$oldix"}; $oldix++) {
        my ($user, $passwd, $uid, $rest) = split(/:/, $rec, 4);
        if (!$callback->($oldref, $newref, $user, $passwd, $uid, $rest)) {
            $newref->{"0$newix"} = $newref->{".$user"} = $rec;
            $newref->{"=$uid"} = $rec if $is_user;
            $newix++;
        }
    }
    return $newix; # so if we're adding, we get the highest entry.
}
```

```
sub MAX_DEADLOCK_TIME () { 30 } # Give up if lock still unavailable after this

sub open_db {
    my (%olddb, %newdb);
    my $DB_PATH = shift;

    # Try up to MAX_DEADLOCK_TIME secs to get lock.
    my $deadtime = 0;

    my $perms;
    if (-e $DB_PATH) {
        die "$DB_PATH is not a regular file!\n" unless -f _;
        $perms = (stat(_))[2] & 07777;
    } else {
        if ($DB_PATH =~ /shadow/) {
            $perms = 0600;
        } else {
            $perms = 0644;
        }
    }

    while (1) {
        sysopen(DBLOCK, "$DB_PATH.new", O_RDWR|O_CREAT|O_EXCL, $perms) and last;
        die "Unexpected error creating lock file $DB_PATH.new: $!\n"
            unless $! =~ /exists/;
        my $towait = rand(2);
        select(undef, undef, undef, $towait);
        $deadtime += $towait;
        if ($deadtime > MAX_DEADLOCK_TIME) {
            die "Deadlock after $deadtime seconds trying to lock $DB_PATH.new\n";
        }
    }
    close(DBLOCK);

    tie %newdb, "DB_File", "$DB_PATH.new", O_RDWR, $perms, $DB_BTREE
        or die "tie to $DB_PATH.new failed: $!";

    tie %olddb, "DB_File", $DB_PATH, O_RDONLY, 0600, $DB_BTREE
        or die "tie to DB_File $DB_PATH failed: $!";
    return (\%newdb, \%olddb);
}

END { unlink $_.".new" for ($PW_DB_PATH, $SH_DB_PATH);  }

=head1 COPYRIGHT

This software is copyright 2001, Simon Cozens, and may be distributed
under the terms of the Artistic License or the GNU Public
License at your choice.
```

7.2. Example 27: Perforce Review Daemon

Example 7-2. Perforce Review Daemon

```
#!/usr/bin/perl                                                    ❶
#
# Perforce Review Daemon
#
# $File: //systems/src/perforce/DEVEL/p4reviewd $
# $Revision: #21 $
# $Date: 2002/01/28 $
# $Author: simon $
#

#use warnings;
use strict;
use Getopt::Long;
use IPC::Run qw(run timeout);
use Sys::Syslog qw(setlogsock openlog syslog);

my $p4_path = "p4";

=head1 NAME

p4reviewd - Perforce Review Daemon

=head1 SYNOPSIS

p4reviewd [--help] [--debug] [--config=/config/file/path]

=head1 DESCRIPTION

The review daemon is a daemon that watches over the Perforce
repository, waiting for new commits. On a commit, it looks at each
file submitted and applies a specified set of rules to them. The rules
primarily match against paths in the repository, and are then keyed to
an array of actions. Each action is performed on a given shell
globbing pattern, and can be passed additional arguments specified in
the configuration file. Perforce options, rules and actions are read
from a configuration file in the following format:

    <p4review>
        <p4config>
            <port>localhost:1666</port>
```

```
            <user>p4reviewd</user>
            <client>p4reviewd</client>
            <counter>p4reviewd</counter>
            <errorsto>sysadmin@bofh.net</errorsto>
        </p4config>
        <ruleset depotspec="//web/foo/...">
            <rule pattern="*.xml">
                <callback type="validate" sub="wellformed_xml"/>
                <callback type="sync" sub="remote_sync">
                    <arg>p4user</arg>
                    <arg>p4client</arg>
                    <arg>remoteuser</arg>
                    <arg>remotehost</arg>
                </callback>
            </rule>
            <rule pattern="*.xsp">
                <callback type="validate" sub="send_for_review">
                    <arg>p4review@example.com</arg>
                </callback>
            </rule>
            <rule pattern="*">
                <callback type="sync" sub="local_sync">
                    <arg>p4user</arg>
                    <arg>p4client</arg>
                </callback>
            </rule>
        </ruleset>
        <ruleset depotspec="//cgi/foo/...">
            <rule pattern="*">
                <callback type="validate" sub="send_for_review">
                    <arg>p4review@example.com</arg>
                </callback>
            </rule>
        </ruleset>
    </p4review>
```

The Perforce options must be defined. The callback subroutines must exist in the review daemon.

The real work of the daemon is in the callbacks. There are executed in the order they appear in the configuration file, the next callback being executed as long as the previous was successful. In other words, processing stops with the first callback to reject a file. For instance, a rule like

```
    <rule pattern="*.html">
        <callback type="validate" sub="valid_html"/>
        <callback type="sync" sub="remote_sync">
            <arg>p4user</arg>
            <arg>p4client</arg>
            <arg>remoteuser</arg>
            <arg>www.example.com</arg>
        </callback>
```

```
    </rule>

would (if provided with the appropriate subroutines) check that any
incoming F<.html> documents were valid HTML, sending them to the web
server C<www.example.com> if so, and sending an email back to the author
with comment (but leaving the file in the repository) if not.

=cut

my %options = (
       help => 0,
       debug => 0,
       config => "/etc/p4reviewd.conf"
      );

my $PID_PATH = "/var/run/p4reviewd.pid";
my $config;

use constant REJECT => 0;
use constant ACCEPT => 1;

sub debug(@) { syslog("debug", join "", @_) if $options{debug} }

if (!GetOptions(\%options, qw(help! debug! config=s)) || $options{help}) {
    warn "Usage: p4reviewd [--help] [--debug] [--config=/config/path]\n";
    exit ($options{help} == 1);
}

setup();

my $sysadmin = $config->{p4config}->{errorsto};
unless ($sysadmin) {
    die "Refusing to run without someone to cry to! (set <errorsto> in $options{config})\n"
}


while (1) {
    debug("Entering main loop");
    my $r = p4command("review", "-t", $config->{p4config}->{counter});

    my @changes = @{$r->{info}};
    if (!@changes) {
        sleep 60;
        next;
    }
    my $last_change;
    # First pass is a sync.
    my @toreview;
    for (@changes) {
        my ($change, $author, $email, $fullname) =
            /^Change (\d+) (\S+) <(\S+)> \(([^\)]+)\)/;
        syslog("info", "Looking at change $change by $author");
        my $description =  p4command(qw(describe -s), $change);
```

```
        # What if this fails?

        for (@{$description->{info}}) {
            chomp;
    my ($depot_spec, $action) = split /\s/;
            my $file = get_locally($depot_spec);
            next unless $file;
            push @toreview, {
                perforce_file => $depot_spec,
                local_file    => $file,
action        => $action,
                changeno      => $change,
                author        => $author,
                email         => $email,
                fullname      => $fullname
            };
        }
# Second pass is a review
for (@toreview) {
    review($_);
    # XXX Clean up local file here?
}
        p4command("counter", $config->{p4config}->{counter}, $change);
    }
}

sub review {
    my $job = shift;

    my ($directory, $filename, $changeno) =
        $job->{perforce_file} =~ m|(//.*/)([^/]+)#(\d+)|
            or die "Can't parse filename $_";
    debug("review: $directory, $filename,  $changeno");

    for (reverse sort keys %{$config->{ruleset}}) { # This (believe it or not)
                                                    # guarantees the correct
                                                    # first-match
        debug ("Trying depot spec $_");
        next unless depot_matches($_, $directory);
        syslog("info", "Applying rules for depot spec $_");

foreach my $rule (@{$config->{ruleset}->{$_}->{rule}}) {
    debug("Trying pattern $rule->{pattern}");
            my $pattern = glob2pat($rule->{pattern});
            next unless $filename =~ /$pattern/;
    debug("Matched pattern $rule->{pattern}");
    foreach (@{$rule->{callback}}) {
callback($_, $job) == ACCEPT || return 0;
    }
    return 1; # All callbacks were successful
}
debug ("Didn't match any rule");
    }
```

```
    debug("Didn’t match any rule in any depot specs");
}

#### Helper functions

sub read_config {
    use XML::Simple;

    $config = eval {
XMLin($options{config},
      keyattr => ["depotspec"],
      forcearray => ["rule", "callback", "arg"])
    };
    if ($@) {
syslog("err", "$@");
return 0;
    }

    # Check that Perforce options are defined
    foreach (qw(port user counter client)) {
unless (defined $config->{p4config}->{$_}) {
    syslog("err", "$_ not defined in configuration file");
    return 0;
}
    }

    # Check that required subroutines are defined
    foreach my $depotspec (keys %{$config->{ruleset}}) {
        debug("Processing ruleset for $depotspec");
foreach my $rule (@{$config->{ruleset}->{$depotspec}->{rule}}) {
    debug("Processing rule for pattern $rule->{pattern}");
    foreach my $cb (@{$rule->{callback}}) {
debug("Processing callback type $cb->{type}, sub $cb->{sub}");
unless (defined &{$cb->{sub}}) {
    syslog("err", "Subroutine $cb->{sub} not defined");
    return 0;
}
    }
}
    }
    return 1;
}

sub setup {
    setlogsock("unix") or die "crit: setlogsock failed\n";
    openlog("p4reviewd", "pid", "LOCAL0");

    # Be a good little daemon
    use POSIX;
    fork and exit;
    POSIX::setsid();

    # Write our PID to a file
```

```
    unless (open(PID_FILE, ">$PID_PATH")) {
syslog("err", "error opening $PID_PATH: $!");
exit(1);
    }

    print PID_FILE "$$\n";

    unless (close(PID_FILE)) {
syslog ("err", "error closing $PID_PATH: $!");
exit(1);
    }

    unless ( $) = $( = getgrnam("daemon") ) {
syslog("err", "Can't find deamon group");
exit(1);
    }

    unless ( $> = $< = getpwnam("p4web") ) {
syslog("err", "Can't find p4web user");
exit(1);
    }

    unless (read_config()) {
syslog("err", "Error processing configuration file");
exit(1);
    }

    # read_config checks that these are defined
    $ENV{P4PORT} = $config->{p4config}->{port};
    $ENV{P4USER} = $config->{p4config}->{user};
    $ENV{P4CLIENT} = $config->{p4config}->{client};

}

sub callback {
    my ($callback, $job) = @_;
    my @args = ($job);
    push @args, @{$callback->{arg}} if defined $callback->{arg};

    debug("Callback type $callback->{type}, sub $callback->{sub}");
    no strict "refs";
    return $callback->{sub}->(@args);
}

sub safesystem {
    my @cmd = @_;
    my ($out, $err);
    eval { run(\@cmd, \undef, \$out, \$err, timeout(30)); };
    syslog("err", "Timeout from @cmd") if $@;
    syslog("err", "Error from @cmd: $err") if $err;
    $err ||= $@;

    return unless defined wantarray;
```

```
    debug("Output from @cmd: $out");
    return ($out, $err) if wantarray;
    return $out;
}

sub glob2pat { # From Cookbook recipe 6.9
    my $globstr = shift;
    my %patmap = ( '*' => '.*', '?' => '.', '[' => '[', ']' => ']');
    $globstr =~  s{(.)}{ $patmap{$1}|| "\Q$1\E" }ge;
    return '^'.$globstr.q/$/;
}

sub depot_matches {
    my $globstr = shift;
    $globstr =~  s{(\.\.\.)}{ ".*" }ge;
    # $globstr =~  s{(\.)}{ "\Q$1\E" }ge;
    my $path = shift;
    debug("Trying to match |$path| against |$globstr|");
    return $path =~ /^$globstr/;
}

sub get_locally {
    my $file = shift;
    debug ("Getting $file from perforce");
    my ($path, $revision) = split /#/, $file;
    my $result = p4command("where", $path);
    if (@{$result->{error}} or !@{$result->{info}}) {
        p4death($result);
    }
    my ($depot_spec, $client_spec, $local_path) =
split /\s/, $result->{info}->[0];
    my $result = p4command(qw(sync -f), $file);
    unless ("@{$result->{info}}" =~ / - (?:updating|added as|refreshing|deleted as|file\(s\
p4death($result);
    }
    debug("Returning $local_path");
    return $local_path;
}

sub sendmail {
    # There are a million and one ways to implement this, but...
    use Mail::Mailer;
    my %options = @_;
    my $mailer = new Mail::Mailer 'sendmail';
    syslog("info", "Sending mail to $options{to} re $options{subject}");
    $mailer->open({
   From    => $options{from} || 'Perforce Review Daemon <p4review@perforce.ox.ac.uk>',
   To      => $options{to},
   Subject => $options{subject},
  })
      or syslog("crit", "Aiiie, can't send mail! : $!");
    print $mailer $options{content};
```

```
    $mailer->close();
}

sub p4command {
    local *P4_OUTPUT;
    my ($rc, @err_msg, @warn_msg, @info_msg);
    my @cmd = ($p4_path, "-s", @_);
    $rc = 0;

    my $pid = open(P4_OUTPUT, "-|");
    die "p4_command: fork failed: $!\n" unless defined($pid);
    if (!$pid) {
    exec(@cmd) or exit(255);
    }
    while(<P4_OUTPUT>) {
    chomp;
    if (s/^error:\s//) {
        push(@err_msg, $_);
    }
    elsif (s/^warn:\s//) {
        push(@warn_msg, $_);
    }
    elsif (s/^info\d*:\s//) {
        push(@info_msg, $_);
    }
    elsif (m/^exit:\s(\d+)/) {
        $rc = $1;
    }
    }
    $rc ||= close(P4_OUTPUT); # Catch the case where the fork didn't happen
    my $result = {rc       => $rc,
        command => \@cmd, # For error tracking
        err_msg  => \@err_msg,
        warn_msg => \@warn_msg,
        info_msg => \@info_msg};

    if ($rc) {
        p4death($result);
    }
    return $result;
}

sub p4death {
    my $result = shift;
    my $rc = $result->{rc};
    my @cmd = @{$result->{command}};
    my @err_msg = @{$result->{err_msg}};
    my @warn_msg = @{$result->{warn_msg}};
    my @info_msg = @{$result->{info_msg}};
    syslog("err", "error from p4 review: @err_msg");
    sendmail(to=>$sysadmin, subject=>"$0 exiting", content=> <<EOF );

Perforce exited in a bad way while performing @cmd:
```

```
(Exit status $rc)

Errors:
@err_msg

Warnings:
@warn_msg

Info:
@info_msg

EOF
    exit(1);
}

#### Actual validators and syncers.

=head2 ACTIONS

=over

=item dummy

Does nothing. Simply accepts anything passed to it.

=cut

sub dummy { return ACCEPT; }

=item wellformed_xml

Runs "rxp" on the file to determine whether or not it is well-formed XML.
If so, accepts the file; otherwise rejects it.

=cut

sub wellformed_xml {
    my ($job, $data) = @_;
    my $file = $job->{local_file};

    # Don't validate deleted files
    return ACCEPT if $job->{action} eq "delete";

    syslog("info", "Testing $file for XML wellformedness");
    my ($report,$error) = safesystem(qw(rxp -x -s), $file);
    return ACCEPT if $? == 0;
    if ($? >> 8 > 2) {
        # Something went wrong
        sendmail(to => $sysadmin, subject => "rxp failure", content => <<EOF

Something went wrong with rxp, leaving an exit status of $?.
Here's a possible error message:
$error
```

```
We used the command

    rxp -x -s $file

EOF
        );
    }

    # Send a rejection mail.
    sendmail(
     to      => $job->{fullname}." <".$job->{email}.">",
     subject => "Invalid XML in recent perforce submission",
     content => <<EOF
Hello.

The XML file
      $job->{perforce_file}

you recently submitted to the Perforce repository could not be
validated. The website has therefore not been updated to reflect this
change. The error message received was:

$report
$error
EOF
    );
    return REJECT;
}

=item reject

Rejects a file. Sends mail to the submitter telling them that the file
has been rejected. A reason for rejection should be passed in the
first argument.

=cut

sub reject {
    my ($job, $data) = @_;
    sendmail(
     to      => $job->{fullname}." <".$job->{email}.">",
     subject => "Your Perforce submission has been rejected",
     content => <<EOF

Hello.

A recently-submitted Perforce job contained a file that was rejected
by the review daemon. The file was
      $job->{perforce_file}

There is no cause for alarm, and the following message should explain
the rejection:
```

```
   $data

EOF
    );
    return REJECT;
}

=item send_for_review

This is like C<reject>, but it also sends an additional mail to the
address specified in the first argument.. For instance, if a user checks
in a CGI program, the C<send_for_review> target can alert appropriate
staff to perform a security audit on the code before (manually)
performing a sync to the web server.

=cut

sub send_for_review {
    my ($job, $mailto) = @_;

    # Don't validate deleted files
    return ACCEPT if $job->{action} eq "delete";
    my $content;
    if ($job->{perforce_file} =~ /#1$/) {
        # New file, print whole thing:
        $content = @{(p4command("print", $job->{perforce_file}))->{info}};
    } else {
        # Just show change
        my $prev = $job->{perforce_file};
        $prev--; # Use magic decrement
        $content = @{(p4command("diff2", "-du", $prev, $job->{perforce_file}))->{info}};
    }

    sendmail(
     to      => $job->{fullname}." <".$job->{email}.">",
     subject => "Your Perforce submission has been sent for review",
     content => <<EOF

Hello.

A recently-submitted Perforce job contained a file which was sent to
$mailto for further review. The file was
      $job->{perforce_file}

The recipients of $mailto will deal with it from here.

EOF
    );
    sendmail(
     from     => $job->{fullname}." <".$job->{email}.">",
     to       => $mailto,
     subject  => "Perforce file ".$job->{perforce_file}." for review",
```

```
     content  => <<EOF

Hello, Perforce here.

$job->{fullname} ($job->{email}) recently sent me the file
      $job->{perforce_file}

This was marked as something needing further review, so you should take
a look at it and take over from here.

This is what was changed:
$content
EOF
    );
    return REJECT;
}

=item remote_sync

SSHs to a remote host and syncs the appropriate file. Takes four
arguments: remote Perforce user name, remote Perforce client name,
remote user name, and remote host name. For example, if the callback
in the configuration file is:

    <callback type="sync" sub="remote_sync">
        <arg>p4user</arg>
        <arg>p4client</arg>
        <arg>user</arg>
        <arg>www.example.com</arg>
    <callback>

then the command

    ssh -T user@www.example.com -u p4user -c p4client sync //foo/bar/baz#x

will be run.

=cut

sub remote_sync {
    my ($job, $p4user, $p4client, $user, $host) = @_;
    syslog("info", "Syncing $job->{perforce_file} to $host");
    my ($output, $error)= safesystem("ssh" ,"-T", "$user\@$host",
       "p4 -u $p4user -c $p4client sync \"$job->{perforce_file}\"");
    if ($error) {
        sendmail (to=>$sysadmin,
                  subject=>"Remote sync failed?",
                  content => "Got $error from ssh. Continuing."
                 );
    }
}

=item local_sync
```

```
Runs p4 locally to sync the appropriate file revision. Takes two
arguments: Perforce user name and Perforce client name.

=cut

sub local_sync {
    my ($job, $p4user, $p4client) = @_;
    syslog("info", "Syncing $job->{perforce_file} locally");
    my $result = p4command ("-u", $p4user, "-c", $p4client, "sync", $job->{perforce_file});
    if (@{$result->{error}}) {
        p4death($result);
    }
}
```

7.3. Extra Special Bonus Example!

Appendix A. Unix cheat sheet

A brief run-down for those whose Unix skills are rusty:

Table A-1. Simple Unix commands

Action	Command
Change to home directory	**cd**
Change to *directory*	**cd** ***directory***
Change to directory above current directory	**cd ..**
Show current directory	**pwd**
Directory listing	**ls**
Wide directory listing, showing hidden files	**ls -al**
Showing file permissions	**ls -al**
Making a file executable	**chmod +x** ***filename***
Printing a long file a screenful at a time	**more** ***filename*** or **less** ***filename***
Getting help for *command*	**man** ***command***

Appendix B. Editor cheat sheet

This summary is laid out as follows:

Table B-1. Layout of editor cheat sheets

Running	Recommended command line for starting it.
Using	Really basic howto. This is not even an attempt at a detailed howto.
Exiting	How to quit.
Gotchas	Oddities to watch for.

B.1. vi

B.1.1. Running

```
% vi filename
```

B.1.2. Using

- `i` to enter insert mode, then type text, press **ESC** to leave insert mode.
- `x` to delete character below cursor.
- `dd` to delete the current line
- Cursor keys should move the cursor while *not* in insert mode.
- If not, try `hjkl`, `h` = left, `l` = right, `j` = down, `k` = up.
- `/`, then a string, then **ENTER** to search for text.
- `:w` then **ENTER** to save.

B.1.3. Exiting

- Press **ESC** if necessary to leave insert mode.
- `:q` then **ENTER** to exit.
- `:q!` **ENTER** to exit without saving.
- `:wq` to exit with save.

B.1.4. Gotchas

vi has an insert mode and a command mode. Text entry only works in insert mode, and cursor motion only works in command mode. If you get confused about what mode you are in, pressing **ESC** twice is guaranteed to get you back to command mode (from where you press i to insert text, etc).

B.1.5. Help

`:help` **ENTER** might work. If not, then see the manpage.

B.2. pico

B.2.1. Running

```
% pico -w filename
```

B.2.2. Using

- Cursor keys should work to move the cursor.
- Type to insert text under the cursor.
- The menu bar has `^X` commands listed. This means hold down **CTRL** and press the letter involved, eg **CTRL**-**W** to search for text.
- **CTRL**-**O**to save.

B.2.3. Exiting

Follow the menu bar, if you are in the midst of a command. Use **CTRL**-**X** from the main menu.

B.2.4. Gotchas

Line wraps are automatically inserted unless the -w flag is given on the command line. This often causes problems when strings are wrapped in the middle of code and similar. \\ \hline

B.2.5. Help

CTRL-G from the main menu, or just read the menu bar.

B.3. joe

B.3.1. Running

```
% joe filename
```

B.3.2. Using

- Cursor keys to move the cursor.
- Type to insert text under the cursor.
- **CTRL-K** then **S** to save.

B.3.3. Exiting

- **CTRL-C** to exit without save.
- **CTRL-K** then **X** to save and exit.

B.3.4. Gotchas

Nothing in particular.

B.3.5. Help

CTRL-K then **H**.

B.4. jed

B.4.1. Running

```
% jed
```

B.4.2. Using

- Defaults to the emacs emulation mode.
- Cursor keys to move the cursor.
- Type to insert text under the cursor.
- **CTRL**-**X** then **S** to save.

B.4.3. Exiting

CTRL-**X** then **CTRL**-**C** to exit.

B.4.4. Gotchas

Nothing in particular.

B.4.5. Help

- Read the menu bar at the top.
- Press **ESC** then **?** then **H** from the main menu.

Appendix C. ASCII Pronunciation Guide

Table C-1. ASCII Pronunciation Guide

Character	Pronunciation
!	bang, exlamation
*	star, asterisk
$	dollar
@	at
%	percent
&	ampersand
"	double-quote
'	single-quote, tick
( )	open/close bracket, parentheses
<	less than
>	greater than
-	dash, hyphen
.	dot
,	comma
/	slash, forward-slash
\	backslash, slosh
:	colon
;	semi-colon
=	equals
?	question-mark
^	caret (pron. carrot)
_	underscore
[ ]	open/close square bracket
{ }	open/close curly brackets, open/close brace
|	pipe, or vertical bar
~	tilde (pron. “til-duh”, wiggle, squiggle)
`	backtick