Developing software on an open source stack

Open source provides everything a programmer needs to invent the next big thing

Skill Level: Intermediate

Martin Streicher (martin.streicher@gmail.com)
Chief Technology Officer
McClatchy Interactive

19 Aug 2008

Web developers are enjoying a renaissance. After spending much of the previous decade toiling on server-centric code, programmers are now putting code front-and-center, turning the Web browser into its own computing platform. Much of the renaissance must be attributed to ingenuity. The newest generation of tools and application frameworks automate and simplify the drudgery of building, deploying, and maintaining a Web site. There are also more tools than ever, and all the most innovative tools are open source. This tutorial provides an expansive survey of the free software available to developers to create and deploy Web applications.

Section 1. Before you start

Learn what to expect from this tutorial and how to get the most out of it.

About this tutorial

This tutorial describes the wide variety of programming tools available on Linux®. Further, it demonstrates how quick and easy it is to start developing on the platform and showcases some of the latest innovations in open source tools.

Objectives

Learn how to install a Web server, a database, and several programming languages on Linux. You also learn how to combine those pieces to build an application, first in PHP, then in Ruby on Rails.

Prerequisites

This tutorial is written for all software developers interested in adopting Linux as a development platform and for developers who want to explore the expansive variety of software development tools available for free as open source. To follow this tutorial, you should have a general familiarity with using a Linux command-line shell and some programming experience. Some experience installing and configuring software on Linux is helpful, but not required.

System requirements

To run the examples in this tutorial, you need a Linux box with at least 300 MB of free disk space. Root access to the machine is required to install a number of the software packages. The examples shown in this tutorial were created on Ubuntu Desktop Linux 8.04.1 running as a virtual machine in Parallels on Mac OS X Leopard. Ubuntu is not required; however, the examples use Aptitude, which can be found in any Debian-based distribution.

Section 2. A modern renaissance

At the moment, producers and designers who create content and pages for the World Wide Web are enjoying something of a renaissance. Media is rich and varied, and standards such as Cascading Style Sheets (CSS) enable expressive, captivating pages.

Web developers are enjoying a renaissance as well. After spending much of the previous decade toiling on server-centric code, programmers are now putting code front-and-center, turning the Web browser into its own computing platform. Asynchronous JavaScript + XML, better known as Ajax, techniques lead the transformation. Glance at Basecamp, Google Mail, and Apple's MobileMe (see the Resources section for links). These sites and others—deemed "thick clients" by pundits—now eclipse software packages available on the desktop.

Much of the renaissance can be attributed to entrepreneurship and creativity. Much of it must also be attributed to ingenuity. The newest generation of tools and application frameworks automate and simplify the drudgery of building, deploying, and maintaining a Web site. Further, there are more tools than ever. Software developers can now choose from a large corpus of tools, programming languages, and frameworks—with Ajax built in—to create advanced Web applications.

Better yet, the vast majority of new software development tools are open source, free of charge, readily available from the Internet, and portable to Linux, Mac OS X, and recent flavors of Microsoft® Windows®. Linux has a particular advantage over the other operating systems: It's available free of cost, too.

In this tutorial, you'll dive into open source and Linux and take an expansive survey of the free software available to developers to create and deploy Web applications. In every quarter, free software offers compelling, competitive, even industry-leading features. You'll look at server software, databases, programming languages, editors and integrated development environments (IDEs), and new regimens for propping up code. Along the way, you'll create some simple Web server environments and craft a number of sample applications.

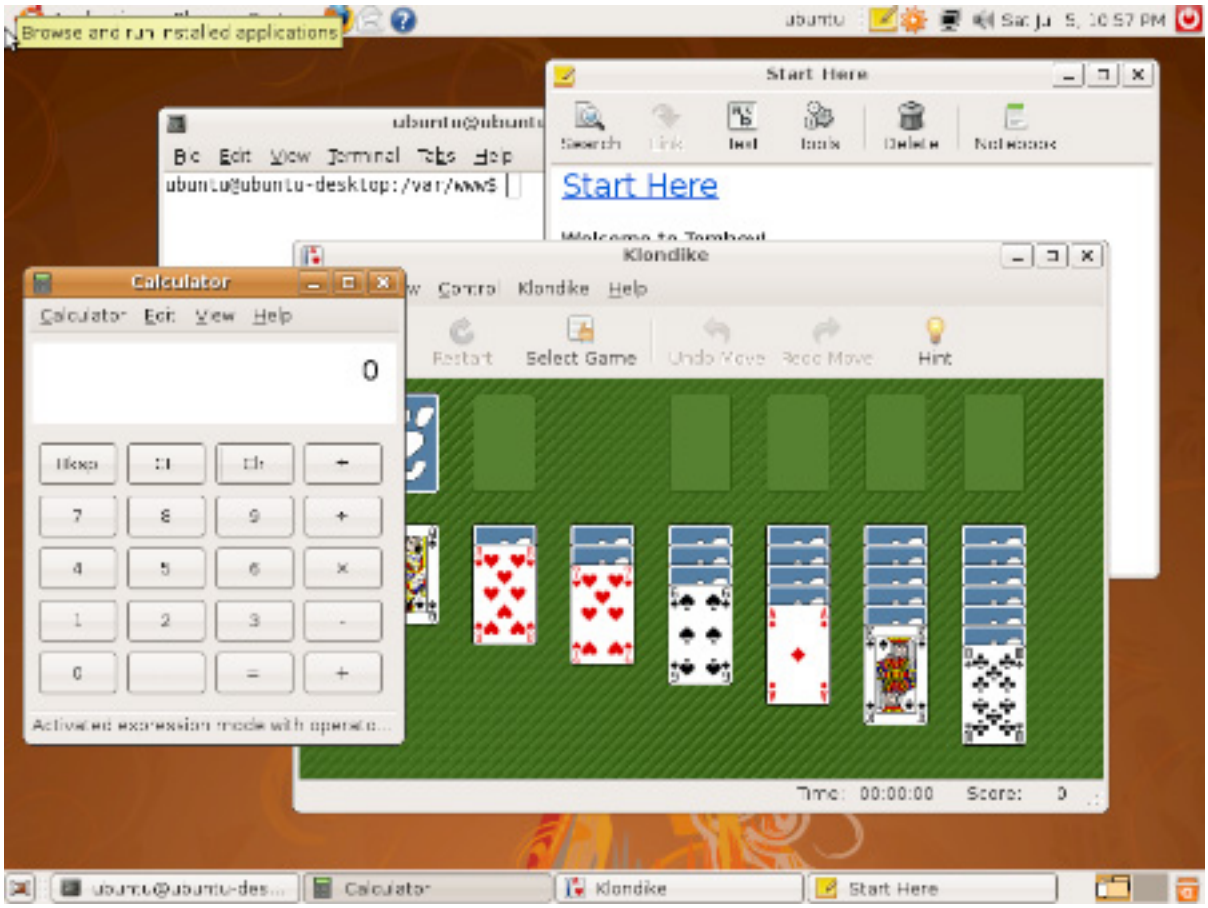
If you have even a modicum of experience in the Linux shell environment, you should find this tutorial easy to follow and perform. If you've never used Linux before, typing commands may seem strange, but persevere, as the step-by-step instructions lead the way.

Much of the software shown here is portable, and you're likely to find pre-built bundles of code ready to install on each of the most popular operating systems. In fact, every copy of Apple's Mac OS X includes most of the software shown here, as well as hundreds of other open source packages.

This tutorial uses Ubuntu Linux (see the Resources section). Ubuntu is available for free, and a novice can quickly download and install it. It's also easy to use and administer. Its desktop is friendly, and a great tool called Aptitude (or "apt") installs and updates software in a snap.

This tutorial is based on a clean install of Ubuntu Desktop 8.04.1 (pictured in Figure 1). Ideally, you have your own Ubuntu machine and have root or superuser access. Superuser access is typically required to install and run system software.

Figure 1. The Ubuntu desktop



Section 3. Join the illuminati of LAMP

To offer a typical Web application, say, an online store, you must have a computer, a connection to the Internet, an operating system, Web server software, a database, and the application itself. On Linux machines, this "stack"—so named, because like a stack of pancakes, the application lays on the database, which lays on the Web server, which lays on the operating system—is called LAMP, an acronym for "*L*inux, *A*pache, *M*ySQL, and *P*erl, *P*HP, *P*ython, or other *p*rogramming language."

Linux

Linux is the operating system and provides all the features required to access the resources of the computer, including the network connection, processor, hard drive, printer, and so on. Linux is a multitasking operating system, capable of running multiple applications at the same time. Linux is composed of a central kernel—the

"brains" of the operation—and thousands of utilities and programs.

Apache

Apache (more formally, the Apache HTTP Server) is an open source Web server. It runs as a perennial application on Linux, idly waiting for requests from Web browsers. When a request is made, Apache accepts the connection, interprets and processes the request (such as "Get the file named my_vacation.html"), and returns the result. Apache multitasks, too, to process multiple requests in parallel to minimize delay.

MySQL

MySQL is an open source database (one of many open source databases available). MySQL persists the data an application requires and collects. For example, a MySQL database might retain a store inventory and all the orders placed through the shopkeeper's Web site. MySQL also runs as a perennial application, or server, and idly listens for requests. As is typical for a server, MySQL multitasks to maximize throughput.

Perl, PHP, and Python

In early LAMP configurations, the "P" stood for Perl, a longstanding open source programming language and arguably the first language to power dynamic (that is, not limited to a collection of static pages) Web applications on Linux. In the early part of this decade, PHP and Python—two other scripting languages coincidentally abbreviated as "P"—emerged, and the acronym LAMP stuck.

Alternative Web servers, databases, and programming languages

Apache is not the only Web server available for Linux. lighttpd (pronounced "lighty"), for example, is small, fast, and simpler to configure than Apache. There are other open source databases, as well. PostgreSQL is a popular open source database preferred for its extensive support for transactions, and SQLite is preferred for its tiny size. (SQLite is embedded in every iPhone.) Of course, a scad of Web programming languages, both compiled and interpreted, are available for Linux, including Java™, Mono, an open source implementation of Microsoft's .NET framework, and Ruby.

Today, LAMP is a catchall to refer to an open source Web stack. It's common to hear developers say, "We're a LAMP shop," meaning their company's Web

infrastructure is based on open source.

Section 4. Installing a Web server, a database, and a programming language

Ubuntu provides the "L" of LAMP for this tutorial, so you need to install "AMP," the rest of the stack. You'll install lighttpd as the Web server, MySQL as the database, and PHP as the programming language. PHP is easy to learn and enjoys a large community of developers and pool of reusable code. The latest revision of lighttpd is 1.4.19. The latest revision of MySQL is MySQL 5.1. The latest version of PHP is PHP 5.2.6.

To begin, open a terminal window in Ubuntu (click **Application > Accessories > Terminal**), move to the shell prompt ($), and become root, the superuser, by typing the command `sudo su -`. When (and whenever) you see the octothorpe (#) prompt, you are root. You can type `whoami` to be sure:

```
$ sudo su -
# whoami
root
```

You can now install lighttpd, MySQL 5, and PHP 5. All the job takes is a few Aptitude commands.

Install lighttpd, the Web server

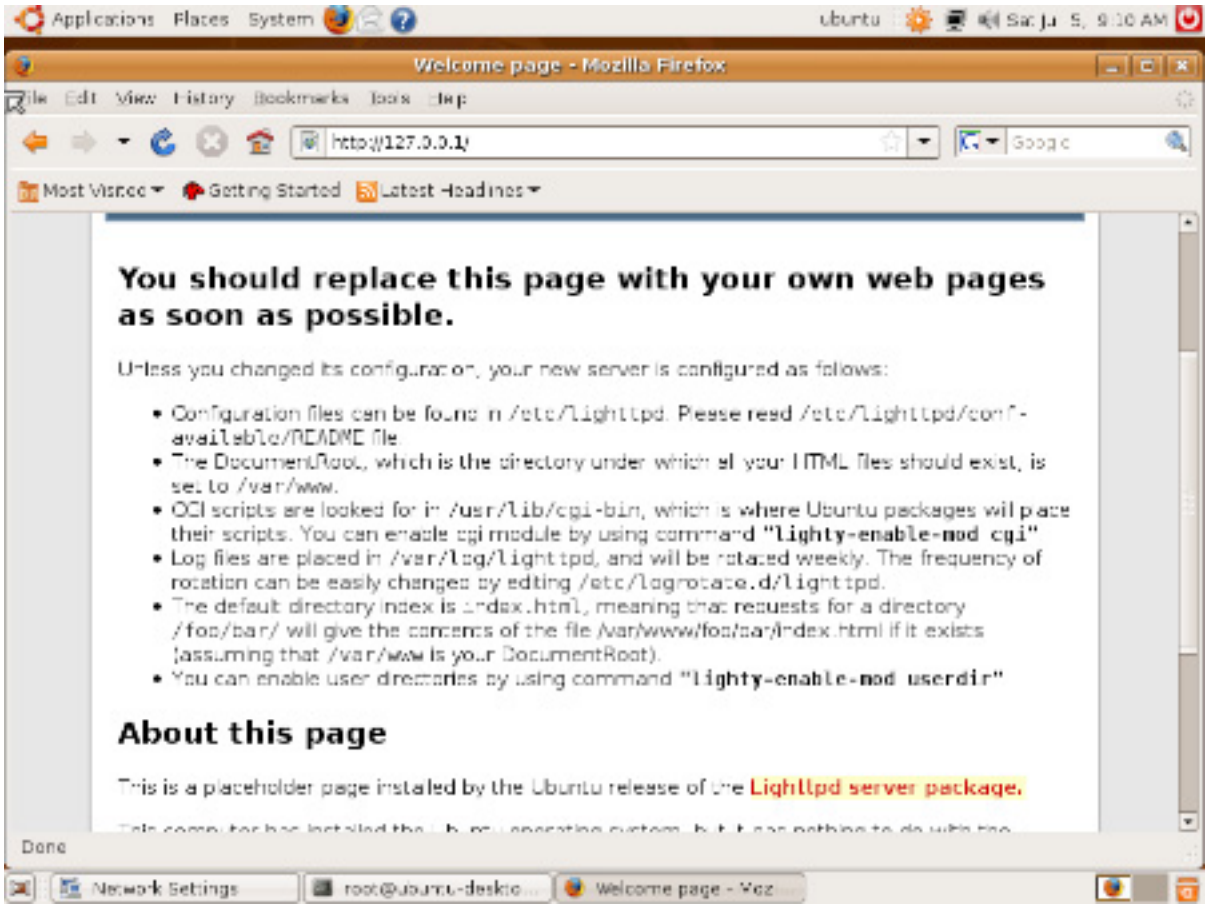
First, install lighttpd:

```
# apt-get install lighttpd
```

`apt-get install` downloads and installs the named package(s). Each time you install a package, Aptitude analyzes the package's dependencies, installs the prerequisites, if any, and then installs the package itself. If Aptitude must install dependencies, it first prompts you to confirm. Read the list of packages provided and press **Y** if you want to continue. If any package provides a server—as lighttpd and MySQL do—Aptitude automatically starts the new server at the end of the process. Nothing could be easier.

After you install lighttpd, open the Firefox browser (there is a link in the Ubuntu menu bar at the top of the screen) and enter `http://127.0.0.1` in the address bar. You should see a page similar to Figure 2. This indicates lighttpd is operational, but still needs configuration, which you will do momentarily.

Figure 2. Indication that lighttpd is running



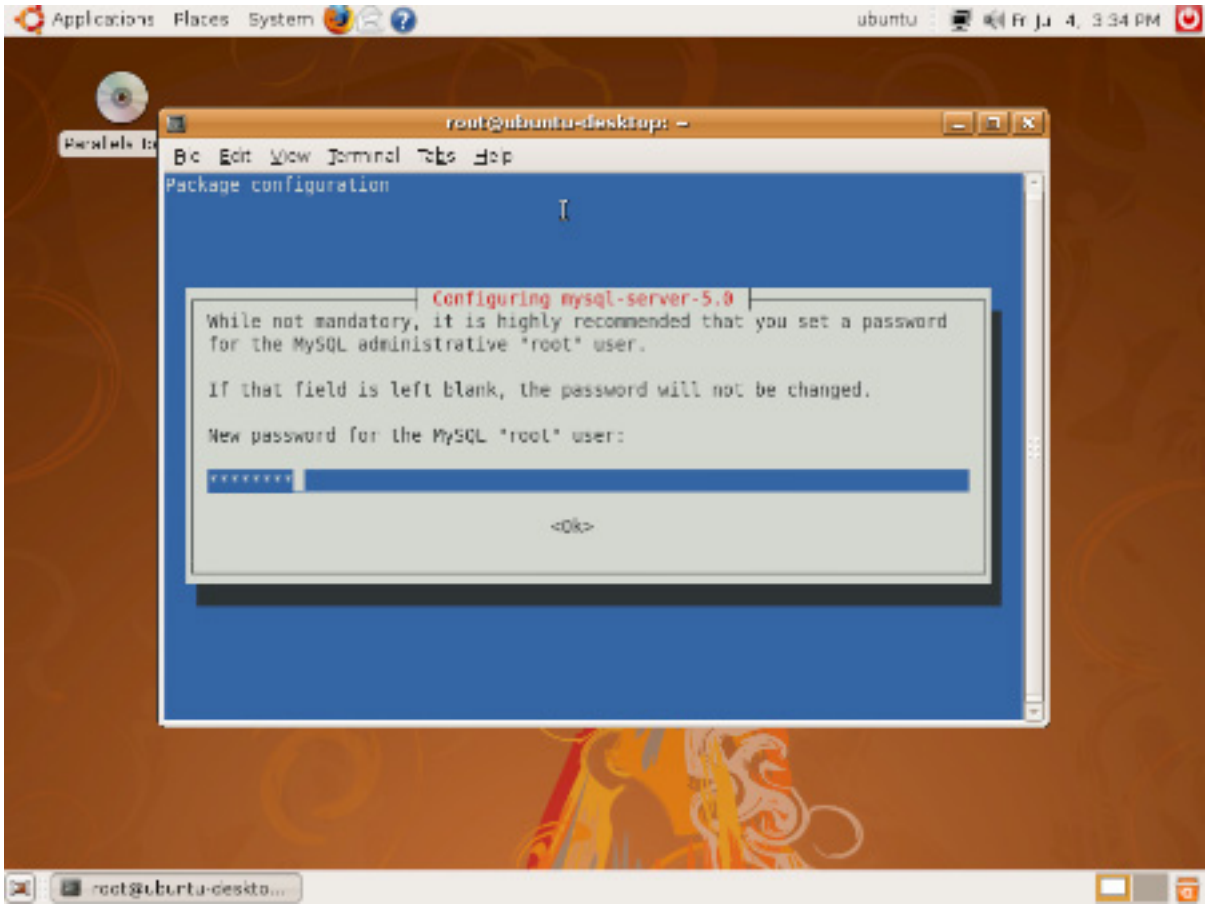
Install MySQL, the database

Next, install the database, MySQL 5. Again, all it takes is one apt command:

```
# apt-get install mysql-server-5.0
```

During the install, watch for an additional prompt to set the MySQL 5 superuser password. The prompt resembles Figure 3. Choose a password that is difficult to guess—it's best practice to use upper and lower case letters, numbers, and punctuation—and press **Enter**. Enter the password again to confirm and press **Enter**.

Figure 3. Remember to enter and confirm the MySQL superuser password



When the install ends, test your handiwork. Launch the MySQL utility and list the initial databases, as shown in Listing 1.

Listing 1. Launch the MySQL utility and list the initial databases

```
$ mysql -uroot -p
Enter your password: s0m3p@ssWoRD
mysql> show databases \G
Database: information_schema
Database: mysql
2 rows in set (0.01 sec)
mysql> quit
```

When prompted for your password, enter the password you chose previously during install. (`s0m3p@ssWoRD`, used above, is just an example of a suitably strong password.) At the MySQL prompt, type `show databases \G`. You should have two databases, one named information_schema and the other named mysql; both are required for proper operation of the software and should not be removed. Type `quit` to exit.

Install PHP, the programming language

Finally, install PHP 5. lighttpd uses the Fast Common Gateway Interface (FastCGI) to run Web applications, so be sure to install the appropriate variant of PHP 5, aptly named php5-cgi. To connect PHP to MySQL, you must also install the PHP 5 MySQL module, php5-mysql:

```
# apt-get install php5-cgi php5-mysql
```

Configure the Web server and programming language to interoperate

There's one final step before you can write a PHP application: You must configure lighttpd and PHP to interoperate. Each software package has its own configuration file, a common paradigm found on Linux systems.

Return to your terminal window and press **Control-D** to exit superuser mode. Next, type:

```
$ sudo gedit /etc/php5/cgi/php.ini
```

This command launches `gedit`—a simple text editor, not unlike Windows's Notepad or the Mac's Text Editor—as root and edits the /etc/php5/cgi/php.ini file. The file is a system file; hence superuser privileges are required to modify it.

On Linux systems, the /etc (pronounced "etsee") directory typically contains the configuration files for all of the packages on the system. It is also common for each package to be self-contained within its own directory, hence /etc/php5. The file to customize for CGI operation of PHP is cgi/php.ini. By convention, customization files end with ".ini" and often the more literal ".conf."

Using `gedit`, scroll down to find this line:

```
; cgi.fix_pathinfo = 0
```

Change it to read:

```
```

```
                cgi.fix_pathinfo = 1
```

Choose **File > Save** or press **Control-S** to save the file. Press **Control-Q** to quit. Next, type this command:

```
$ sudo gedit /etc/lighttpd/lighttpd.conf
```

Find the line that begins `server.modules=` and add `mod_fastcgi` to the comma-separated list found in parentheses. `mod_fastcgi` extends lighttpd to support the FastCGI version of PHP. Requests from a browser are received by lighttpd. It translates the request into a FastCGI request and hands off processing to PHP. The process is reversed when the PHP application returns a reply.

Your new list should resemble Listing 2, perhaps with more modules.

Listing 2. Updated server.modules = list

```
server.modules = (
  "mod_fastcgi",
  "mod_access",
  "mod_alias",
  "mod_accesslog",
  "mod_compress"
)
```

Now scroll to the end of the file and add the snippet from Listing 3 verbatim.

Listing 3. Code snippet that directs lighttpd to the FastCGI version of PHP and the correct socket

```
fastcgi.server = (
  ".php" => ((
        "bin-path" => "/usr/bin/php-cgi",
        "socket" => "/tmp/php.socket"
)))
```

This snippet tells lighttpd where to find the FastCGI version of PHP and what local network connection, or *socket*, to use to send PHP requests. Save the file and quit the editor.

Just one last step—restart lighttpd with the `sudo /etc/init.d/lighttpd restart` command. (Most servers must be restarted after a configuration change.)

```
$ sudo /etc/init.d/lighttpd restart
Stopping web server lighttpd [OK]
Starting web server lighttpd [OK]
```

Success! You are now ready to write your first (and second) PHP application.

Section 5. Your first (and second) PHP application

As I mentioned earlier, PHP is a popular Web development language because it is easy to learn and easy to mingle with common Hypertext Markup Language (HTML). Where HTML tags look like `<head>` and `</div>`, PHP code is embedded in `<? ?>` tags.

First PHP application

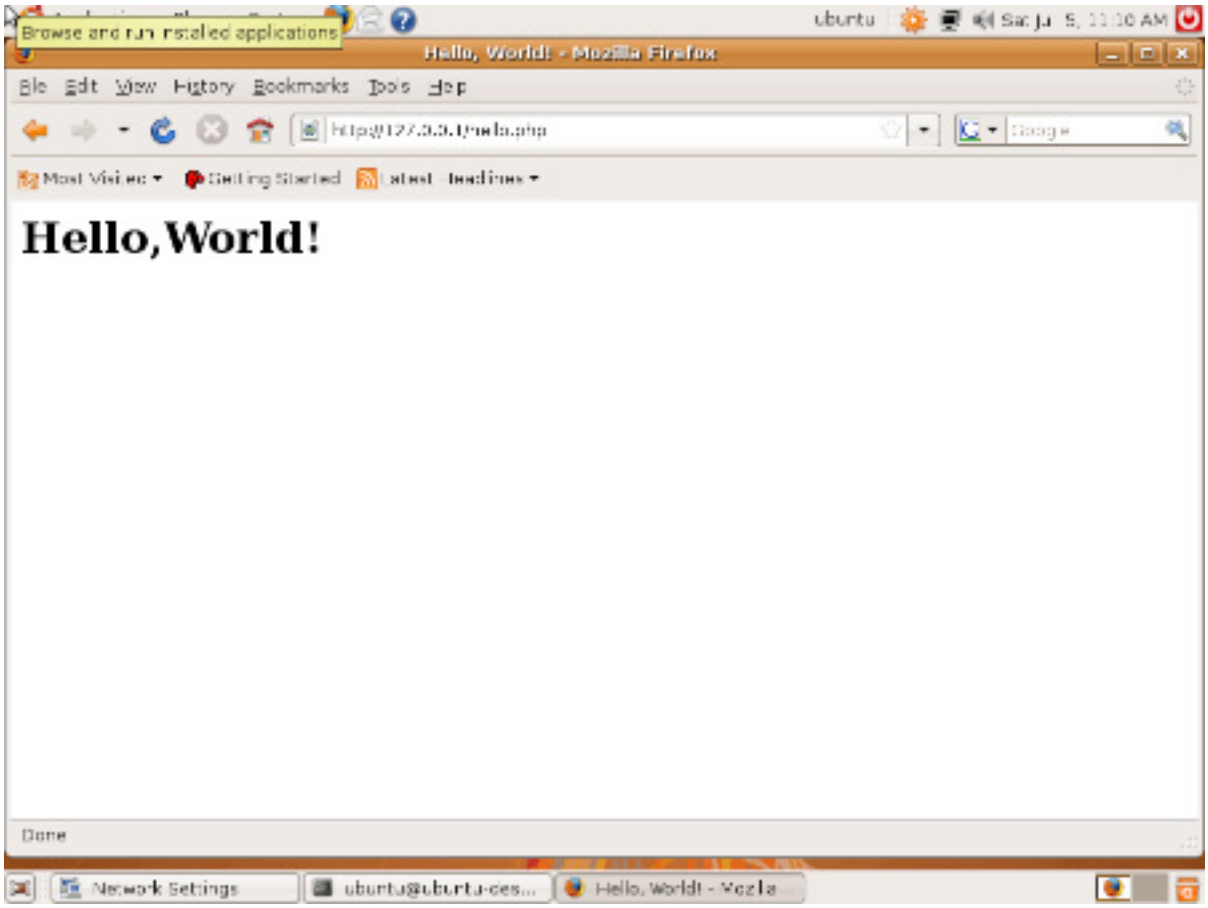
Hence, a simple PHP application might look like Listing 4.

Listing 4. Example of a simple PHP application

```
<html>
  <head> <title> Hello, World!</title> </head>
  <body>
        <h1>
          <? echo 'Hello, World! ?>
        </h1>
  </body>
</html>
```

Try the application. Run `gedit` with the command `sudo gedit /var/www/hello.php`, enter the snippet above, and save the file. Switch to Firefox and type `http://127.0.0.1/hello.php` in the address bar. You should see something like Figure 4. If so, lighttpd and PHP are working fine.

Figure 4. Hello, World! in PHP, running in lighttpd



Second PHP application

Now move on to create a more complex example. Use the MySQL database to persist some information and tie all the LAMP pieces together. In particular, write code to display the members of Superheroes, Inc., an outsourcing firm for superhero talent. To create the application, you'll need to do the following:

1. Design and create the database table to store the membership roster.
2. Populate the database with some information.
3. Write PHP code to connect to the database and display a Web page.

Create and populate the database

The snippet in Listing 5 is a SQL script that does the following:

1. Creates a database named superheroes.

2. Creates a table within the superheroes database named members.
3. Populates the members table with four heroes.
4. Creates a MySQL user named hero with permissions to access and alter the database.

Listing 5. SQL script that creates and populates the superheroes database

```
DROP DATABASE IF EXISTS `superheroes`;

CREATE DATABASE `superheroes`;

USE `superheroes`;

DROP TABLE IF EXISTS `members`;

CREATE TABLE `members` (
  `id` int(11) NOT NULL,
  `name` varchar(128) NOT NULL,
  `city` varchar(128) NOT NULL,
  `superpower` varchar(128) NOT NULL,
  `initiated` date NOT NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `id` (`id`),
  KEY `name` (`name`)
) ENGINE=MyISAM;

INSERT INTO `members`
  (`id`, `name`, `city`, `superpower`, `initiated`)
VALUES
  (1,'Batman','Gotham City','Detection','1939-05-01'),
  (2,'Spider-man','New York City','Webs','1962-08-01'),
  (3,'Daredevil','Hell\'s Kitchen (NYC)','Radar','1964-04-01'),
  (4,'Superman','Metropolis','Strength','1938-06-01');

GRANT ALL ON `superheroes`.*
  TO 'hero'@'localhost' IDENTIFIED BY 'shizzle';
```

In general, each Web application should have its own MySQL database and unique user to prevent miscreants and other applications from gaining access. Moreover, do not use the MySQL root user for any application. A root compromise can lead to the destruction of all of your data. In fact, it's a best practice to create a new MySQL superuser and delete root.

Using `gedit` again, save the snippet into a file named superheroes.sql and use the `mysql` command-line tool to execute the script. Because the user, hero, does not yet exist, you must run `mysql` as the MySQL root user:

```
$ mysql -uroot -p < superheroes.sql
Enter password:
```

Type your MySQL root password at the prompt and press **Enter**. To verify your

work, look at the database, this time as the user, hero. You should see the information shown in Listing 6.

Listing 6. Superheroes database

```
$ mysql -uhero -pshizzle superheroes
mysql> select * from members;
-----+------------+--------------------------------+------------+-------------
| id | name       | city                           | superpower | initiated  |
-----+------------+--------------------------------+------------+-------------
|  1 | Batman     | Gotham City                    | Detection  | 1939-05-01 |
|  2 | Spider-man | New York City                  | Webs       | 1962-08-01 |
|  3 | Daredevil  | Hell's Kitchen (New York City) | Radar      | 1964-04-01 |
|  4 | Superman   | Metropolis                     | Strength   | 1938-06-01 |
----+------------+--------------------------------+------------+--------------
4 rows in set (0.00 sec)
```

Great! The database is ready.

Create the Web page

Listing 7 is the PHP code to create the Web page.

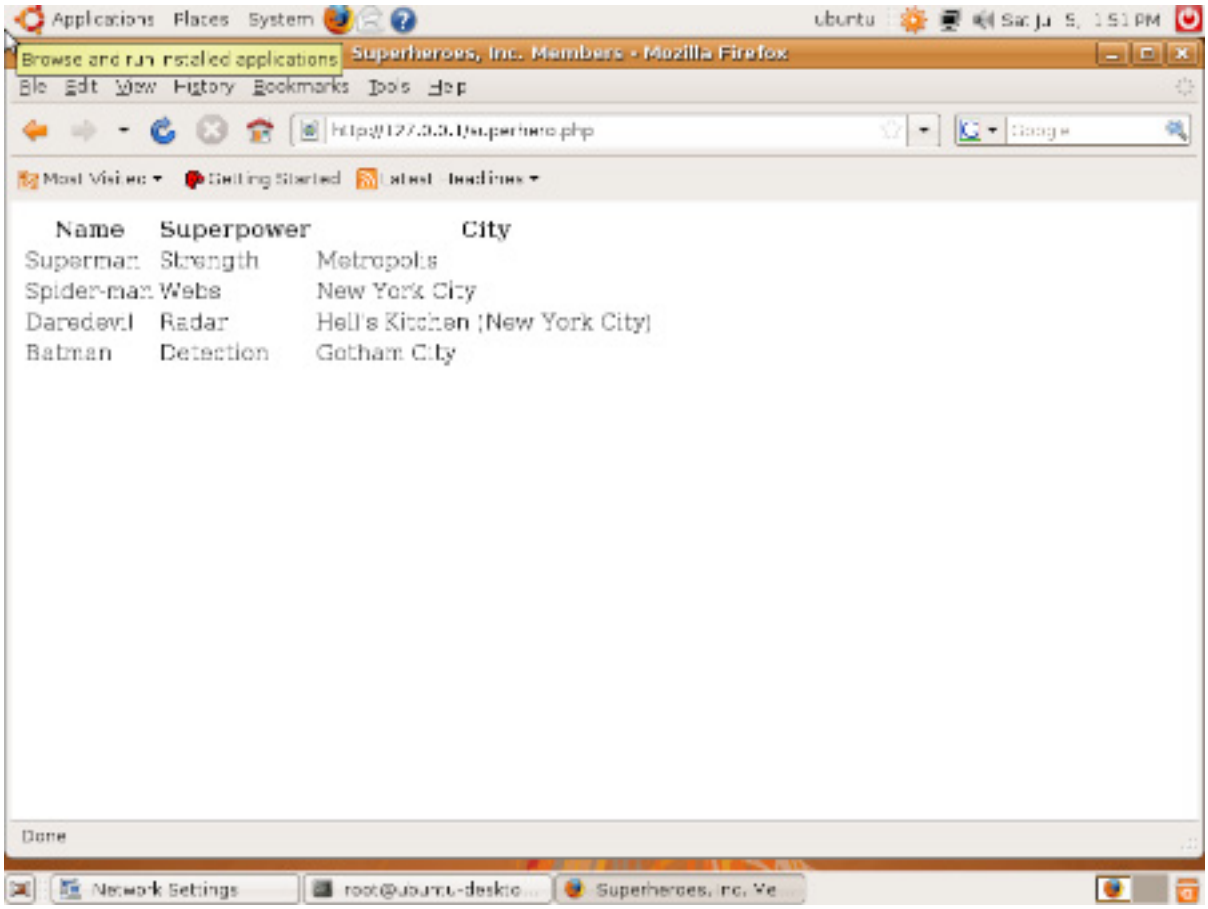
Listing 7. PHP code to create the Superheroes, Inc. Web page

```
<html>
<head>
<title>
  Superheroes, Inc. Members
</title>
</head>
<body>
<?
  if ( ! ( $connection = mysql_connect("localhost", "hero", "shizzle" ) ) ) {
        die( 'An error occurred. Check your connections.' );
  }

  if ( !mysql_select_db( "superheroes", $connection ) ) {
        die( 'An error occurred. Check your database and permissions.' );
  }

  $query = "select * from members order by name";
  if ( ! ( $result = mysql_query( $query, $connection ) ) ) {
        die( 'An error occurred. ' . mysql_error() );
  }
?>

  <table>
        <tr>
          <th>Name</th>
          <th>Superpower</th>
          <th>City</th>
        </tr>
<?
        while ($row = mysql_fetch_assoc($result)) {
?>
          <tr>
                <td><? echo $row{'name'}; ?></td>
                <td><? echo $row{'superpower'}; ?></td>
                <td><? echo $row{'city'}; ?></td>
          </tr>
```

```
<?
        }
?>
  </table>

<?
  mysql_close( $connection );
?>
</body>
</html>
```

Recreate this snippet with (or copy-and-paste it from the browser to) `gedit` and save the code to the /var/www/superhero.php file. Open Firefox again and point the browser to `http://127.0.0.1/superhero.php`. In an instant —nay, faster than a speeding bullet—you should see the page shown in Figure 5.

Figure 5. The list of available superhero contractors



Review of the PHP code

Let's walk through the PHP code. The container for the code is a traditional HTML page. PHP code appears within <? ?> tags, and you can intermix HTML tags and

code, as is done to create the table of heroes. The application connects to the local MySQL server, chooses a database, runs a query, and displays the results. More specifically:

1. `mysql_connect()` establishes a connection to the MySQL server on the local machine (aptly nicknamed *localhost*) using hero's credentials.
2. `mysql_select_db()` selects the superheroes database. All subsequent database operations—reads, writes, updates, deletes—now apply to it (until, say, another database is selected using `mysql_select_db()` again).
3. `mysql_query()` asks the database for information. This query asks for all columns of all rows in the database, returned in alphabetical order. The value returned by `mysql_query()` is a *result set*, a kind of pointer to all matching rows.
4. The loop in the remainder of the code retrieves each row in the result set as an *associative array*. This is an array where the indices of the array are names, and in this case, the names of the selected columns. (Recall that traditional arrays use integers as the indices.) Hence `$row{'name'}` is the name column of the current row.
5. After all results have been rendered as rows in the HTML table, the database connection is closed with `mysql_close()`. The application ends when there's nothing left to render.

And you're done! Of course, this example is no Twitter or Mint.com (see the Resources section for links), but it nonetheless reflects how easy (and cheap) it is to launch an application. On Ubuntu, oodles of software is just one Aptitude command away.

Refinements and best practices

Certainly, there are many refinements and best practices you should consider as any application expands in complexity, including:

- Keep logic and presentation separate
- Object features
- Application frameworks

Keep logic and presentation separate

Keep your code (logic) and HTML (presentation) separate. Almost all the Web programming languages offer a template system to realize this dichotomy. After your data structures have been initialized, control passes to the template system to interpret and render the data structures. Templates make your code and HTML easier to read and allow the developer and designer clear separation of roles. Smarty is one such template system used widely in PHP applications. Template Toolkit is a popular template system for Perl.

Object features

Virtually all modern programming languages offer object features. If you've used C++ or Smalltalk, for instance, you can find a Web programming language to fit your notions of object-oriented programming. You can find flexible dialects, such as Perl, where the motto is, "There's more than one way to do it." There are also rigorous dialects, such as Python, where white space is a critical part of the language syntax.

Application frameworks

Consider adopting an application framework. In most cases, there's no need to reinvent the wheel. Whether the framework is a set of classes or a complete set of conventions, tools, and techniques, build on the work of others and hasten your time to market. Shop around—you may even choose your programming language after choosing a framework.

Actually, one particular framework, Ruby on Rails, has converted thousands of developers to Ruby, a scripting language with the power of Perl, the object features of Java, and the minimal syntax of Python. If you're considering Web development on open source, Ruby on Rails demands a look.

Section 6. Exploring Ruby on Rails

The masthead of the Ruby on Rails home page (shown in Figure 6) is a great introduction to the package.

Figure 6. The Rails motto

"Web development that doesn't hurt" is one motto. Another Rails' motto is, "Don't

repeat yourself." Rather than reinvent the wheel with each new Web application, Ruby on Rails enforces a number of conventions, leaving you, the developer, to focus on features that matter.

For example, every Rails application is organized according to strict convention—configuration in one well-defined place, style sheets in another, and JavaScript™ code in yet another. Convention leads to a certain vernacular, a shorthand for design, development, deployment, and reuse. Rails developers speak the same language (pardon the pun).

Further, all Ruby on Rails applications are based on the same paradigm: the venerable Model-View-Controller pattern.

Model-View-Controller pattern

As mentioned earlier, it's desirable to separate logic from presentation. Further, it's also advantageous to separate logic from the complexities of a user interface. Those divisions are the principle tenets of the Model-View-Controller pattern.

Model

The model encapsulates data. For example, a model for a member of Superheroes, Inc. would include a name, a city, a superpower, and other information, such as when the superhero first appeared. Additionally, the model might include code to derive other attributes, much like an object in an object-oriented language might combine private data and public methods to surface computed attributes.

View

The view presents information. Like a template system, the View teases apart data structures and renders an apt display, including the user interface elements required to manipulate the data. As an example, a view for the superhero application might display the existing data and provide the controls to add, edit, or delete members.

Controller

Finally, the controller is responsible for input and output. Web forms and user interface controls generate the input. The controller interprets the input, affects the model according to business rules, and prepares data for the view, the output.

Division of roles

The division of roles between model, view, and controller reduces and narrows complexity. Developers have taken the pattern even further. Conceptually, each:

- Model represents a single entity
- View enables one task
- Controller realizes a single purpose

If you own an iPhone, you're already familiar with this "extreme" Model-View-Controller archetype—the iPhone interface is designed using this pattern.

Re-implement the Superheroes application in Ruby on Rails

Now you'll re-implement the superheroes membership roster application in Ruby on Rails. For convenience and quick prototyping, Rails includes its own Web server named WEBrick. (When you're ready to deploy your application for public use, you can combine lighttpd and Rails for scale.)

Install Ruby and Rails

You already have a database installed, so now install Ruby and Rails. The most recent release of Ruby suitable for Rails is 1.8.6; the most recent version of Rails is 2.1.0. Again, turn to Aptitude to install the packages:

```
$ sudo apt-get install ruby rubygems rails
$ sudo gem install rails --include-dependencies
```

The first command installs the Ruby programming language, the Ruby package manager (RubyGems), and the Rails suite of tools. Gem is like Aptitude, only for Ruby libraries and modules. The second command uses gem to install the Rails programming libraries and all prerequisites.

Initialize your application

When Rails is installed, you're ready to start coding. Be sure you're not root, change to your home directory, and initialize your application with the `rails` utility. Just specify the application's name—in this case, superheroes—and Rails does the rest, as shown in Listing 8.

Listing 8. Initializing your applications with rails

```
$ cd
$ rails superheroes
          create
          create  app/controllers
          create  app/helpers
          create  app/models
          create  app/views/layouts
```

```
          create  config/environments
          create  config/initializers
          create  db
          create  doc
          create  lib
          create  lib/tasks
          create  log
          create  public/images
          create  public/javascripts
          create  public/stylesheets
          create  script/performance
          create  script/process
          create  test/fixtures
          create  test/functional
          create  test/integration
          create  test/mocks/development
          create  test/mocks/test
          create  test/unit
          create  vendor
          create  vendor/plugins
          create  tmp/sessions
          create  tmp/sockets
          create  tmp/cache
          create  tmp/pids
          create  Rakefile
          create  README
          create  app/controllers/application.rb
          create  app/helpers/application_helper.rb
          create  test/test_helper.rb
          create  config/database.yml
          create  config/routes.rb
          create  public/.htaccess
          create  config/initializers/inflections.rb
          create  config/initializers/mime_types.rb
          create  config/boot.rb
          create  config/environment.rb
          create  config/environments/production.rb
          create  config/environments/development.rb
          create  config/environments/test.rb
          create  script/about
          create  script/console
          create  script/destroy
          create  script/generate
          create  script/performance/benchmarker
          create  script/performance/profiler
          create  script/performance/request
          create  script/process/reaper
          create  script/process/spawner
          create  script/process/inspector
          create  script/runner
          create  script/server
          create  script/plugin
          create  public/dispatch.rb
          create  public/dispatch.cgi
          create  public/dispatch.fcgi
          create  public/404.html
          create  public/422.html
          create  public/500.html
          create  public/index.html
          create  public/favicon.ico
          create  public/robots.txt
          create  public/images/rails.png
          create  public/javascripts/prototype.js
          create  public/javascripts/effects.js
          create  public/javascripts/dragdrop.js
          create  public/javascripts/controls.js
          create  public/javascripts/application.js
          create  doc/README_FOR_APP
          create  log/server.log
```

```
          create  log/production.log
          create  log/development.log
          create  log/test.log
```

The `rails` command creates all the directories and required files you need to house and build your application—again, following strict convention.

Create the database

Next, you need a database. Because the name of the application is superheroes, call the database superheroes_development. Again, this follows convention: Take the application name and suffix it with "_development." (If you forget what the convention is, open the db/database.yml file and look at the database names.) You can use the `mysqladmin` command to create the database immediately from the command line:

```
$ sudo mysqladmin -uroot -p create superheroes_development
```

Configure the application to connect to the database

With the database in place, you can now configure your Rails application to connect to it. Change to the directory named superheroes and edit the config/databases.yaml file to resemble the snippet in Listing 9.

Listing 9. Edited config/databases.yaml file

```
development:
  adapter: mysql
  encoding: utf8
  database: superheroes_development
  username: hero
  password: shizzle
  socket: /var/run/mysqld/mysqld.sock
```

You should only have to change the `username` and `password` entries in the `development` section because all other fields were pre-filled by convention when you generated the superheroes project. Save the file and exit the editor.

Build the tables

You should now have access to the new database, albeit an empty one, without any tables or data. The next step is to build the tables.

Unlike the PHP application, which used a SQL script to initialize the database, you can use Rails itself to create the members table. Indeed, you can use Rails to create the members table (the model) and a simple interface (the view and controller), to

create, edit, and delete the member data. Rails calls this feature a *scaffold*. A scaffold stands in place while you erect a specific part of your application. Simply remove each scaffold as you flesh out your own code.

To create a scaffold, you simply name it and provide a list of data field and type tuples. As with PHP, the members model requires a name, a superpower, a city name, and a start date, as shown in Listing 10.

Listing 10. Script to create the members table

```
$ ruby ./script/generate scaffold member name:string city:string initiated:date
   exists  app/models/
          exists  app/controllers/
          exists  app/helpers/
          create  app/views/members
          exists  app/views/layouts/
          exists  test/functional/
          exists  test/unit/
          create  app/views/members/index.html.erb
          create  app/views/members/show.html.erb
          create  app/views/members/new.html.erb
          create  app/views/members/edit.html.erb
          create  app/views/layouts/members.html.erb
          create  public/stylesheets/scaffold.css
  dependency  model
          exists    app/models/
          exists    test/unit/
          exists    test/fixtures/
          create    app/models/member.rb
          create    test/unit/member_test.rb
          create    test/fixtures/members.yml
          create    db/migrate
          create    db/migrate/001_create_members.rb
          create  app/controllers/members_controller.rb
          create  test/functional/members_controller_test.rb
          create  app/helpers/members_helper.rb
           route  map.resources :members
```

How Rails organizes the Model-View-Controller pattern

If you read through the list of files the scaffold created, you can see how Rails organizes the Model-View-Controller pattern.

Rails created a model in member.rb. The model's name is Member and, by convention, is associated with the table named members. If you look down four lines, you see a special file generated expressly for the purpose of creating that table: 001_create_members.rb, shown in Listing 11. The `create_table` block contains the fields you specified.

Listing 11. 001_create_members.rb

```
class CreateMembers < ActiveRecord::Migration
  def self.up
        create_table :members do |t|
          t.string :name
          t.string :city
```

```
          t.date :initiated

          t.timestamps
        end
  end

  def self.down
        drop_table :members
  end
end
```

The scaffold created six files for the view, including a nascent, simple style sheet to customize the look of a page. The index.html.erb file lists all the members. The show.html.erb file lists the specifics of an individual member. new.html.erb is a form to create a new member, while edit.html.erb is another form to edit the details of an existing record. members.html.erb is a wrapper that embeds the four former pages. It might eventually contain your header and footer and other common elements.

The scaffold generator also created a controller in members_controller.rb. This provides everything needed to process input from forms, affect the model, and generate the views.

Create the members table

You'll run the application in a moment. First, you must create the members table. `rake`, a kind of `make` utility specific to Rails, includes a command to manage the database. Specifically, Rails calls a change to the structure or content of the database a *migration*, and `rake` is able to migrate forward or backward to yield a specific database configuration. (Databases change constantly during development. Migrations allow you to recreate a known database state.) If you want to migrate to the latest version of the database, simply type `rake db:migrate`, as shown in Listing 12.

Listing 12. Use rake db:migrate to migrate to the latest version of the database

```
$ rake db:migrate
== 1 CreateMembers: migrating ================================================
-- create_table(:members)
   -> 0.1660s
== 1 CreateMembers: migrated (0.1668s) =======================================
```

And, believe it or not, the application is ready to run.

Run the application

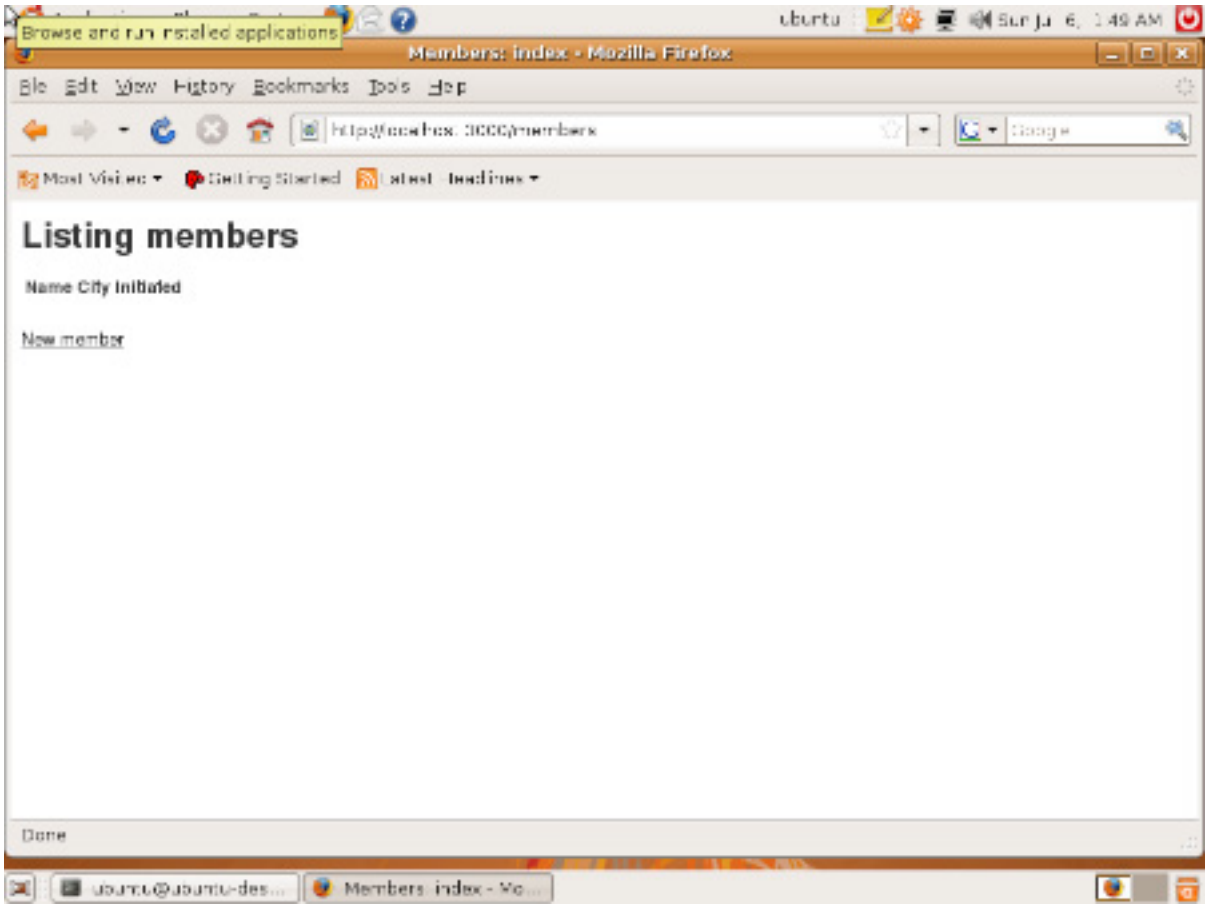
From the superheroes directory, type `ruby script/server`, as shown in Listing 13.

Listing 13. Type ruby script/server to run the application

```
$ ruby script/server
=> Booting WEBrick...
=> Rails application started on http://127.0.0.1:3000
=> Ctrl-C to shutdown server; call with --help for options
[2008-07-06 01:47:11] INFO  WEBrick 1.3.1
[2008-07-06 01:47:11] INFO  ruby 1.8.6 (2007-09-24) [i486-linux]
[2008-07-06 01:47:11] INFO  WEBrick::HTTPServer#start: pid=10298 port=3000
```

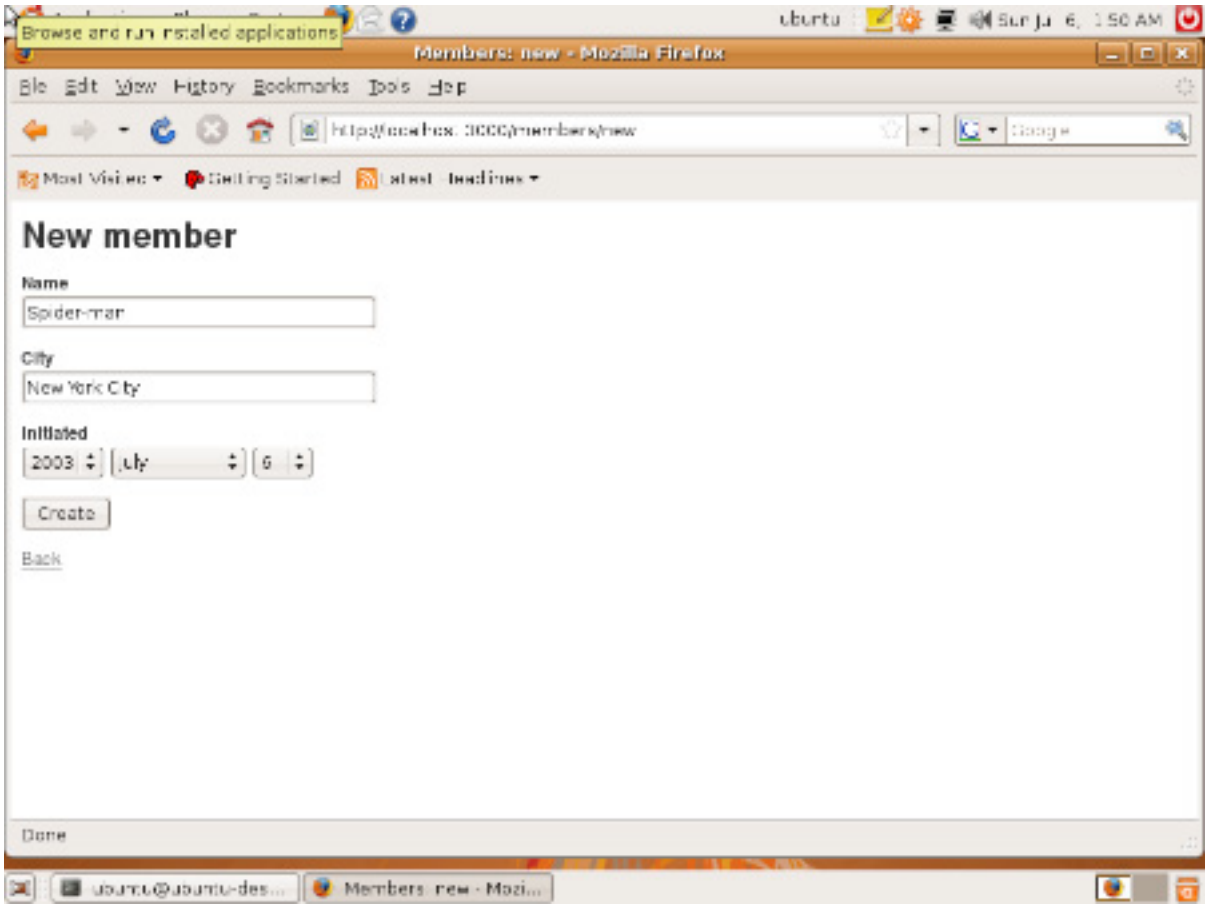
Point your browser to http://localhost:3000/members, and you should see the page shown in Figure 7.

Figure 7. The superhero members index page



Yes, the page lacks entries because the members table wasn't seeded with the four records. For now, though, click **New Member**. Fill out the form for Spider-man (as shown in Figure 8) and click **Create**. Navigate around the user interface and create more heroes.

Figure 8. A form to create a new member



Scaffold limitations

While a scaffold jump starts development, it has limits. You may have noticed that the date menu offers a ten-year range when you really need a span of eighty years. You could argue that the intermediate step to view the hero record after it has been created is unnecessary. Both can be easily corrected—just edit the Member view and controller. In any case, scaffolds, conventions, and the Model-View-Controller pattern combine to make Rails powerful and, as its motto promises, Web development painless.

Section 7. The Eclipse integrated development environment

As you've seen, the LAMP stack takes many forms and provides many choices at each layer of the stack. But what about programming aids, such as source control, a syntax-directed editor, a debugger, and an integrated development environment

(IDE) that pulls tools together seamlessly into a kind of programming portal?

Open source programming aids

Not surprisingly, open source programming aids are plentiful.

Source control

The leading source control systems on Linux are Subversion and Git. Both provide basic revision control, branching, and tagging, and both are well suited to distributed, parallel development. Many open source projects use these tools to manage code. For example, the Linux kernel development team uses Git, written by kernel guru and Linux founder Linus Torvalds, to coordinate their work.

Editor

If you want to shop for a code editor on Linux, try "text editor Linux" at Google. The longstanding incumbents Vim and Emacs have dedicated followings, as does Kate. Search the Web a little to find the one that suits your tastes.

Debugger

In general, each Web scripting programming language comes complete with its own debugger. A debugger and console application, for instance, is bundled into Rails and WEBrick. Put the statement `debugger` anywhere in your Rails application, and WEBrick will halt with a prompt so you can list the values of variables, peruse the call stack, and step through methods. The granddaddy of all Linux debuggers is GDB, used to debug any of the compiled languages on Linux.

Integrated development environment

As you might expect, there are many open source IDEs available for Linux. However, Eclipse is one of the most popular, with an enormous number of plug-ins that extend the core features of the IDE to support new language, source control, interactive user interface design tools, and a great deal more. Eclipse Plugin Central (EPIC) houses over 1,000 modules alone. Components are also packaged together into platforms, such as the Web Tools Platform and PDT. See the Resources section for links to these tools.

The latest release of Eclipse is called Ganymede, or Version 3.4. It is available for Windows, Linux, and Mac OS X, and you can find several flavors of Ganymede, such as a bundle ideal for C developers and another customized for Java developers. Links to the bundles are available from the Eclipse home page.

Install Eclipse

Ganymede works best on Java 1.6. You can again use Aptitude to install the Java Software Development Kit (SDK) on your Linux machine.

```
$ sudo apt-get install sun-java6-jdk
$ sudo update-alternatives --config java
```

The second command, `sudo update-alternatives --config java` may prompt you to make a choice. If it does, choose the option that points to the sun-java6-jdk. For instance, from the options in Listing 14, you would choose 3.

Listing 14. Example Java options

```
There are 3 alternatives which provide `java'.

  Selection    Alternative
-----------------------------------------------
          1    /usr/bin/gij-4.2
*+        2    /usr/lib/jvm/java-gcj/jre/bin/java
          3    /usr/lib/jvm/java-6-sun/jre/bin/java

Press enter to keep the default[*], or type selection number: 3
Using '/usr/lib/jvm/java-6-sun/jre/bin/java' to provide 'java'.
```

To continue, go to the Ganymede home page and download Eclipse Classic. When the download is complete, unpack the release with `tar xvzf eclipse-SDK-3.4-linux-gtk.tar.gz`, as shown in Listing 15.

Listing 15. Unpack the release with tar xvzf eclipse-SDK-3.4-linux-gtk.tar.gz

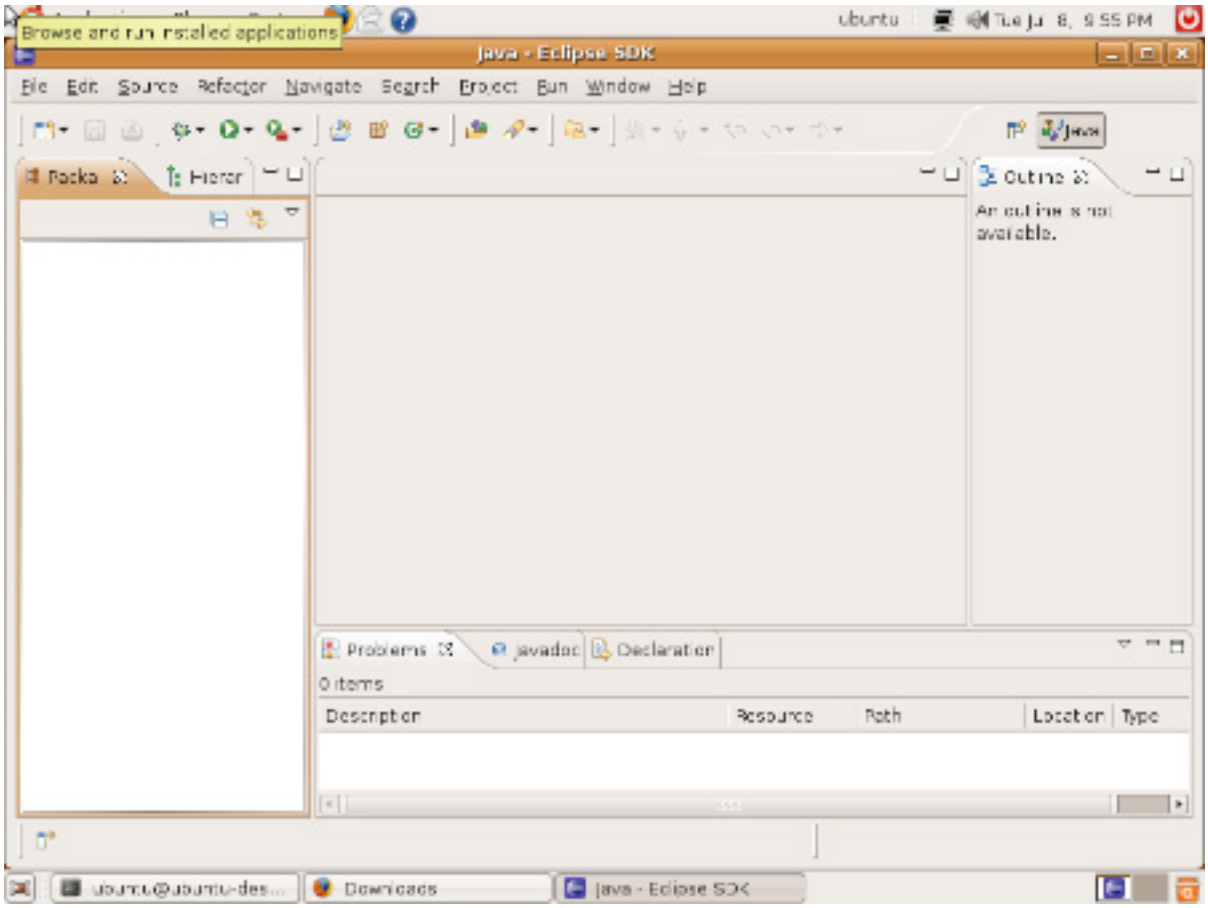
```
$ tar xvzf eclipse-SDK-3.4-linux-gtk.tar.gz
eclipse/
eclipse/features/
...
eclipse/plugins/org.eclipse.jface.databinding_1.2.0.I20080515-2000a.jar
```

Change to the newly-created directory named eclipse and run ./eclipse:

```
$ cd eclipse
$ ./eclipse
```

When prompted to create a workspace, click **OK**. In a moment, you should see the Eclipse workspace, which resembles Figure 9.

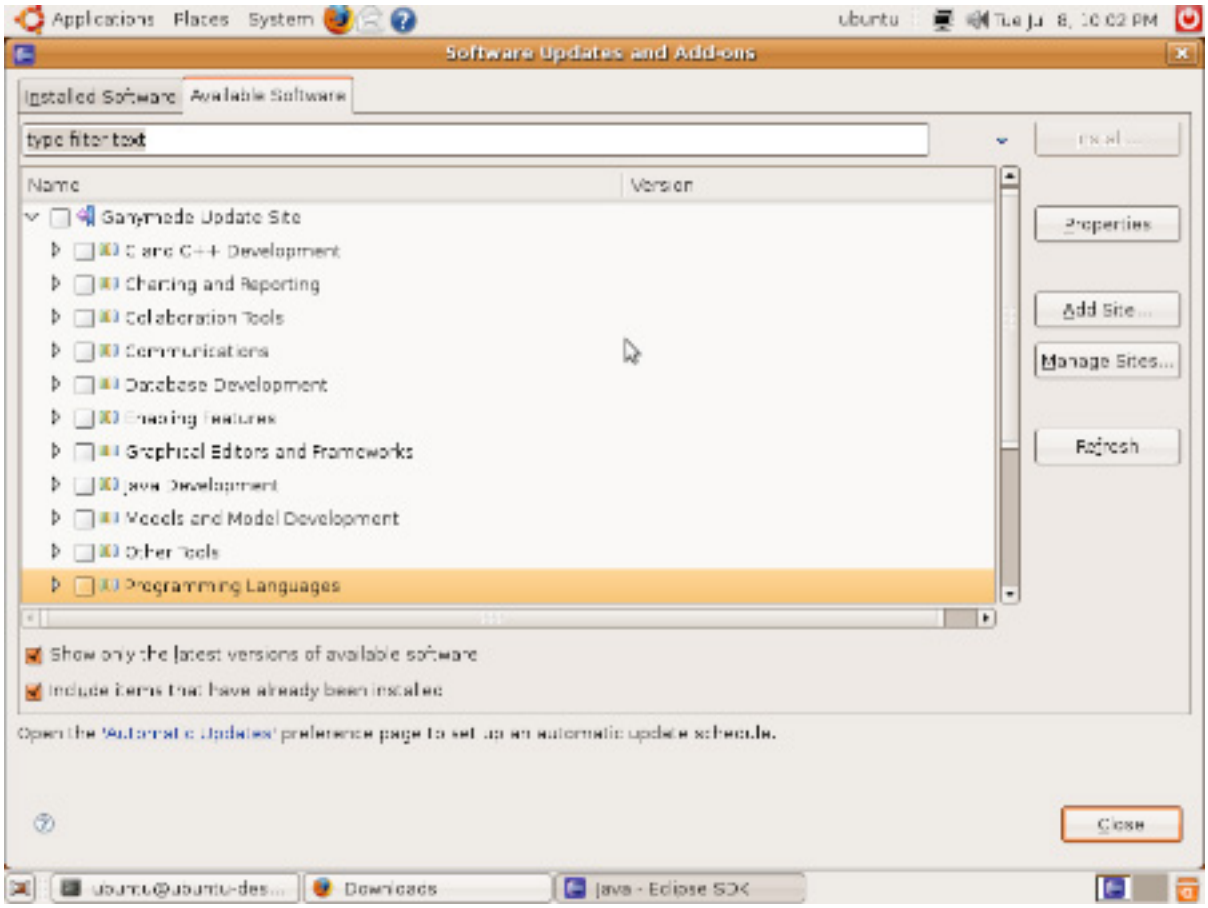
Figure 9. The Eclipse workspace



Install the Ruby development environment for Eclipse

Continue and install the Ruby development environment for Eclipse. In the process, you will also see how to update and expand Eclipse's features. Click **Help > Software Updates**. When the dialog box displays, choose the tab labeled **Available Software**. When the tab displays, expand the arrow next to **Ganymede Update Site**. In a moment, you should see a long list of options you can install, as shown in Figure 10.

Figure 10. Some choices to expand your Eclipse environment

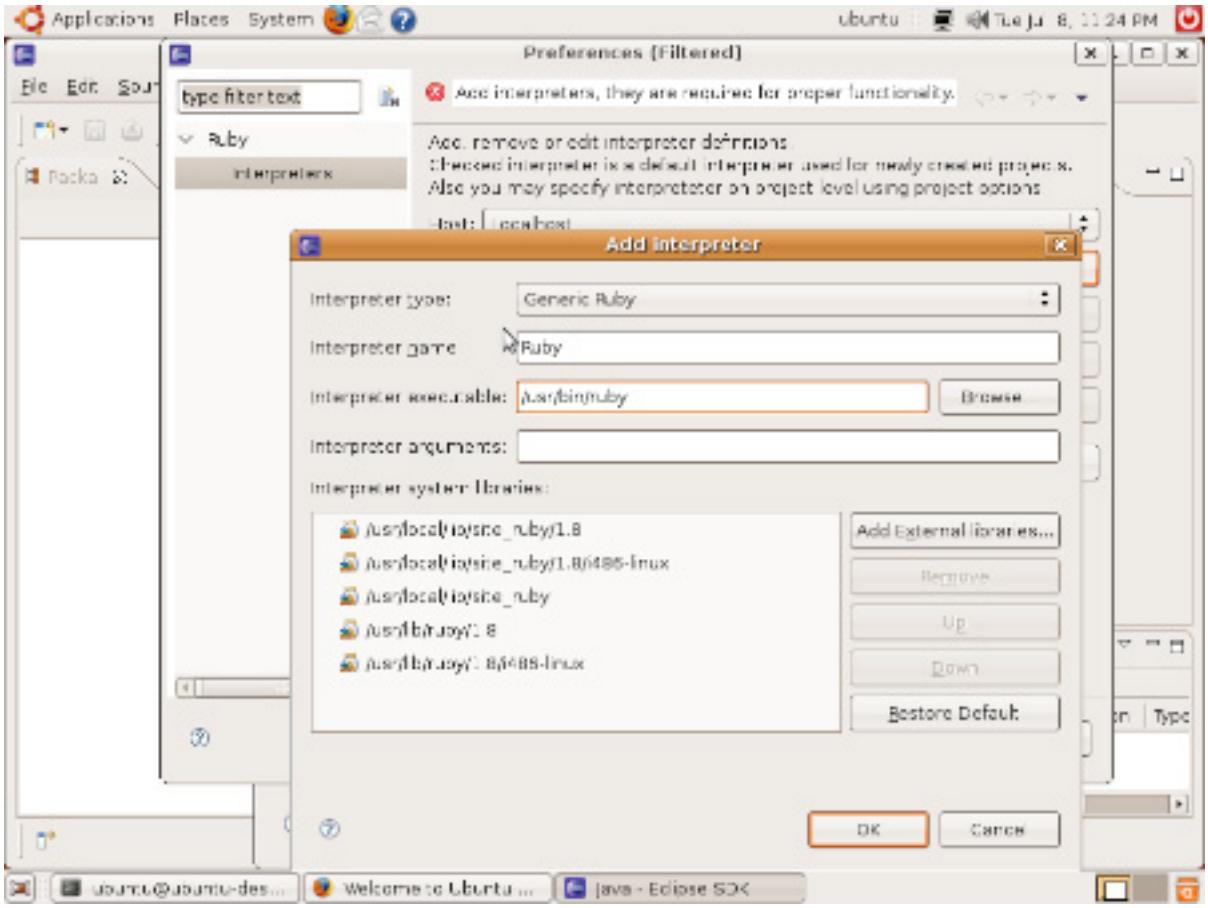


Expand the arrow next to **Programming Languages** and check the **Ruby** option. Then click **Install**. The Update Manager computes all of the software needed to run the Ruby environment. On the next screen, click **Finish** to start the install. After a few moments, the install should complete, and you are ready to code Ruby in a context-sensitive editor built expressly for the language.

Create a Ruby project

Choose **File > New > Project**, scroll down in the dialog box to expand the arrow next to **Ruby**, and choose **Ruby Project**. Click **Next** and assign the project a name, such as simple demo. Next, click the **Configure interpreters** link. You must choose the Ruby interpreter you want to use—unless you've installed JRuby, the only choice is /usr/bin/ruby. Click **Add** and fill out the following pane to resemble Figure 11. Click **OK**, then **OK** again, and, finally, **Finish**.

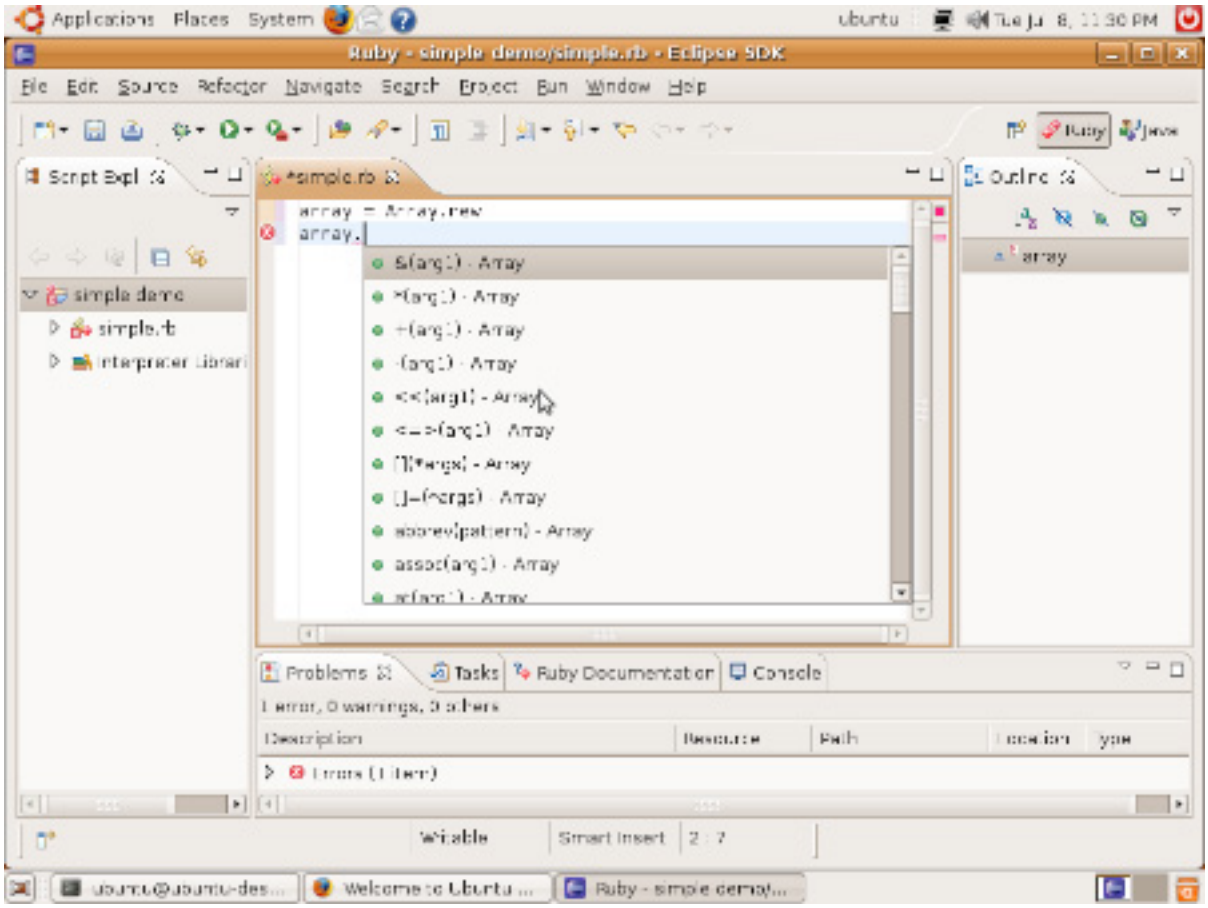
Figure 11. Choose your Ruby interpreter



Because you just created a Ruby project, Eclipse switches your view to the Ruby Perspective, where you have the tools you need to code in Ruby. Choose **File > New > Empty Ruby Script**, and name the new file simple.rb. In the middle pane, start typing some Ruby code.

Figure 12 is a snapshot of the pane as a line of Ruby is written. Because `array` is an `Array`, the editor prompts you with the list of methods available to the class. You can also see the line has been flagged as erroneous, at least temporarily. Variables display on the right. Warnings, such as questionable syntax like a space between a method name and a parenthesis, display on the bottom.

Figure 12. Auto-completion for the Ruby programming language



No matter what language you program in, chances are that Eclipse supports it. Browse the Eclipse site and the plug-in repository for neat features.

Section 8. It's all about choice

Wrap up

Given the features of modern Web browsers and cheap desktop computing power, programmers are pushing the limits of what can be done on the Web. Luckily, those pioneers are blazing trails for others to follow and sharing the tools of their trade, largely as open source. Take a common PC, a copy of Linux, and download all the software you need to create, deploy, manage, and scale your Web site, free of charge. Truly pioneering.

Resources

Learn

- "Ajax: A New Approach to Web Applications" (Adaptive Path, February 2005) provides a good overview of Ajax.
- To learn more about CSS, go to the Cascading Style Sheets Home Page of the World Wide Web Consortium (W3C).
- Basecamp, Google Mail, and Apple's MobileMe are all examples of "thick clients" that eclipse software packages available on the desktop.
- Twitter and Mint.com are just two highly-interactive Web applications built with a modern framework such as Ruby on Rails.
- Learn about the Model-View-Controller pattern on Wikipedia.
- Learn how to program in PHP 5.
- Explore the power of Ruby on Rails.
- Browse the technology bookstore for books on these and other technical topics.
- Read more about the Subversion and Git source control systems.

Get products and technologies

- This tutorial uses Ubuntu Linux.
- Apache is an open source Web server that is part of LAMP.
- lighttpd uses FastCGI to run Web applications.
- Discover all of the features of MySQL 5.0. Other open source databases include PostgreSQL, preferred for its extensive support for transactions, and SQLite, preferred for its tiny size.
- Perl is a longstanding open source programming language that was used in early LAMP configurations. It was followed by PHP and Python.
- Some of the other Web programming languages available for Linux include Java, Mono, and Ruby.
- Download the lightweight, but powerful lighttpd Web server.
- Download the latest version of the open source IDE, Eclipse.
- Eclipse Plugin Central (EPIC) houses over 1,000 modules for Eclipse.
- Components for Eclipse are also packaged together into platforms, such as the Web Tools Platform and the PDT.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Martin Streicher
Martin Streicher is the Chief Technology Officer of McClatchy Interactive and the former Editor-in-Chief of *Linux Magazine.* Martin holds a Masters of Science degree in computer science from Purdue University and has programmed UNIX-like systems since 1986. You can reach Martin at martin.streicher@gmail.com.

Trademarks