Chapter 1

Getting Started

Before we start talking about the Ruby language, it would be useful if we helped you get Ruby running on your computer. That way, you can try sample code and experiment on your own as you read along. In fact, that's probably essential if you want to learn Ruby—get into the habit of writing code as you're reading. We will also show you some different ways to run Ruby.

The Command Prompt

(Feel free to skip to the next section if you're already comfortable at your system's command prompt.)

Although there's growing support for Ruby in IDEs, you'll probably still end up spending some time at your system's command prompt, also known as a *shell prompt* or just plain *prompt*. If you're a Linux user, you're probably already familiar with the prompt. If you don't already have a desktop icon for it, hunt around for an application called Terminal or xterm. (On Ubuntu, you can navigate to it using Applications > Accessories > Terminal.) On Windows, you'll want to run `cmd.exe`, accessible by typing `cmd` into the dialog box that appears when you select Start > Run. On OS X, run Applications > Utilities > Terminal.app.

In all three cases, a fairly empty window will pop up. It will contain a banner and a prompt. Try typing `echo hello` at the prompt and hitting Enter (or Return, depending on your keyboard). You should see `hello` echoed back, and another prompt should appear.

Directories, Folders, and Navigation

It is beyond the scope of this book to teach the commands available at the prompt, but we do need to cover the basics of finding your way around.

If you're used to a GUI tool such as Explorer on Windows, or Finder on OS X, for navigating to your files, then you'll be familiar with the idea of *folders*—locations on your hard drive that can hold files and other folders.

When you're at the command prompt, you have access to these same folders. But, somewhat confusingly, at the prompt they're called *directories* (because they contain lists of other directories and files). These directories are organized into a strict hierarchy. On Unix-based systems (including OS X), there's one top-level directory, called / (a single forward slash). On Windows, there is a top-level directory for each drive on your system, so you'll find the top level for your C: drive at C:\ (that's the drive letter, C, a colon, and a single backslash).

The path to a file or directory is the set of directories that you have to traverse to get to it from the top-level directory, followed by the name of the file or directory itself. Each component in this name is separated by a forward slash (on Unix) or a backslash (on Windows). So, if you organized your projects in a directory called projects under the top-level directory and if the projects directory had a subdirectory for your time_planner project, the full path to the README file would be /projects/time_planner/readme.txt on Unix and C:\projects\time_planner\readme.txt on Windows.

To navigate to a directory, use the cd command. (Because the Unix prompt varies from system to system, we'll just use a single dollar sign to represent it here.)

```
$ cd /projects/time_planner       (on Unix)
C:\> cd \projects\time_planner    (on Windows)
```

Now, on Unix boxes, you probably don't want to be creating top-level directories. Instead, Unix gives each user their own *home directory*. So, if your username is dave, your home directory might be located in /usr/dave, /home/dave, or /Users/dave. At the shell prompt, the special character ~ (a single tilde) stands for the path to your home directory. You can always change directories to your home directory using cd ~, which can also be abbreviated to just cd.

To find out the directory you're currently in, you can type pwd (on Unix) or cd on Windows. So, for Unix users, you could type this:

```
$ cd /projects/time_planner
$ pwd
/projects/time_planner
$ cd
$ pwd
/Users/dave
$
```

On Windows, there's no real concept of a user's home directory:

```
C:\> cd \projects\time_planner
C:\projects\time_planner> cd \projects
C:\projects>
```

You can create a new directory under the current directory using the mkdir command:

```
$ cd /projects
$ mkdir expense_tracker
$ cd expense_tracker
$ pwd
/projects/expense_tracker
```

Spaces in Directory Names and Filenames

Most operating systems now allow you to create folders with spaces in their names. This is great when you're working at the GUI level. However, from the command prompt, spaces can be a headache, because the shell that interprets what you type will treat the spaces in file and folder names as being parameter separators and not as part of the name. You can get around this, but it generally isn't worth the hassle. If you are creating new folders and files, it's easiest to avoid spaces in their names.

Notice that to change to the new directory, we could just give its name relative to the current directory—we don't have to enter the full path.

I suggest you create a directory called pickaxe to hold the code you write while reading this book:

```
$ mkdir ~/pickaxe       (on Unix)
C:\> mkdir \pickaxe     (on Windows)
```

Get into the habit of changing into that directory before you start work:

```
$ cd ~/pickaxe           (on Unix)
C:\> cd \pickaxe         (on Windows)
```

Installing Ruby

Quite often, you won't even need to download Ruby. It now comes preinstalled on many Linux distributions, and Mac OS X includes Ruby (although the version of Ruby preinstalled on OS X is normally several minor releases behind the current Ruby version). Try typing **ruby -v** at a command prompt—you may be pleasantly surprised.

If you don't already have Ruby on your system or if you'd like to upgrade to a newer version (remembering that this book describes Ruby 1.9), you can install it pretty simply. But first, you have a choice to make: go for a prepackaged distribution or build Ruby from source?

Prepackaged Distributions

A packaged distribution of Ruby simply works out of the box. You install it, and it runs. Binary distributions are prebuilt for a particular operating environment and are convenient if you don't want to mess around with building Ruby from source. The downside of a packaged distribution is that you may have to take it as given: it may be a minor release or two behind the leading edge, and it may not have the optional libraries that you might want (although you may be able to install additional libraries using RubyGems, described in a moment). If you can live with that, you'll need to find a packaged distribution for your operating system and machine architecture.

Windows Distributions

In the old days (where *old* means Ruby 1.8), things were good for Windows users. There was a great "batteries included" package that would install not just Ruby but also a vast array of libraries and gems. This was called the *One-Click Installer*, or OCI.

However, with the advent of Ruby 1.9, the situation has changed somewhat. Ruby 1.9 hasn't been around long, so some of the libraries that were included in the 1.8 installer have not yet been made compatible with 1.9. As I write this, the OCI project is in a state of flux. The maintainer, Luis Lavena, is planning on releasing a Ruby 1.9 version of the OCI in early 2009, but it may well not contain as many libraries as the 1.8 version. The situation will improve over time. (And, if you feel strongly about this, I know Luis would welcome your help porting stuff over.)

So, you have a couple of choices for installing Ruby 1.9 on Windows. You can visit http://rubyforge.org/pro and see whether a one-click installer is available. If not, you can download a prebuilt binary from ruby-lang.org.[1]

Linux Distributions

Most modern Linux distributions use the apt-get system (or the Synaptic GUI) to find and install Ruby. As of November 2008, the following command installs Ruby, irb, and ri:

```
$ sudo apt-get install ruby1.9 libruby1.9 libreadline-ruby1.9 irb1.9
$ sudo apt-get install rdoc1.9 ri1.9
```

This installs all the Ruby commands with a 1.9 suffix, so you'll need to do this:

```
$ ruby1.9 -v
ruby 1.9.0 (2007-12-25 revision 14709) [i486-linux]
```

Be aware that the version of Ruby we just installed is many months behind the current version.

Note that you need to have superuser access to install global packages on a Unix or Linux box, which is why we use the sudo command.

OS X Distributions

Leopard (OS X 10.5) comes with Ruby 1.8 preinstalled.[2] If you want to make use of the new Ruby 1.9 features, you'll want to install Ruby yourself. You can do this from source, or you can use a package management system. I personally use MacPorts.[3] Once you have

1. Visit http://www.ruby-lang.org/en/downloads/, and look for *Ruby on Windows*.

2. At some point, it seems likely that Apple will include MacRuby. This is its own port of Ruby 1.9, tightly integrated into the Objective-C runtime. In the meantime, you can download MacRuby from http://www.macruby.org.

3. http://www.macports.org/

the basic ports system installed, as described on its website, installing Ruby is as simple as doing this:

```
$ sudo port install ruby19
```

As with apt-get for Linux, MacPorts currently installs the Ruby executables with a 1.9 suffix (ruby1.9, irb1.9, and so on). If you don't already have /opt/local/bin in your path, you'll need to add it. As an alternative, you could investigate http://rubyosx.com/, which claims to offer a packaged OS X installation.

Building Ruby from Source

Because Ruby is an open source project, you can download the interpreter's source code and build it on your own system. Compared to using a binary distribution, this gives you a lot more control over where things go, and you can keep your installation totally up-to-date. The downside is that you're taking on the responsibility of managing the build and installation process. This isn't onerous, but it can be scary if you've never installed an open source application from source.

The first thing to do is to download the source. This comes in three flavors, all from http://www.ruby-lang.org/en/downloads:

- The stable release in *tarball* format. A tarball is an archive file, much like a .zip file.
- The *stable snapshot*. This is a tarball, created nightly, of the latest source code in Ruby's stable development branch. The stable branch is intended for production code and in general will be reliable. However, because the snapshot is taken daily, new features may not have received thorough testing yet—the stable tarball in the previous bullet will be generally more reliable.
- The *nightly snapshot*. This is again a tarball, created nightly. Unlike the stable code in the previous two tarballs, this code is leading edge, because it is taken from the head of the development branch. Expect things to be broken in here.

If you plan on downloading either of the nightly snapshots regularly, it may be easier to subscribe to the source repository directly. The sidebar on page 31 gives more details.

Once you've loaded a tarball, you'll have to expand the archive into its constituent files. Use the tar command for this (if you don't have tar installed, you can try using another archiving utility, because many now support tar-format files).

```
$ tar xzf snapshot.tar.gz
ruby/
ruby/bcc32/
ruby/bcc32/Makefile.sub
ruby/bcc32/README.bcc32
  :     :     :
```

This installs the Ruby source tree in the subdirectory ruby/. In that directory, you'll find a file named README, which explains the installation procedure in detail. To summarize, you build Ruby on Unix-based systems using the same four commands you use for most other open source applications: ./configure, make, make test, and make install. You can build Ruby under other environments (including Windows)—see README.win32 in the distribution's win32 subdirectory as a starting point.

Source Code from This Book

We have made the source code from this book available for download from our website at http://pragprog.com/titles/ruby3/code. Sometimes, the listings of code in the book correspond to a complete source file. Other times, the book shows just part of the source in a file—the program file may contain additional scaffolding to make the code run.

Running Ruby

Now that Ruby is installed, you'd probably like to run some programs. Unlike compiled languages, you have two ways to run Ruby—you can type in code interactively, or you can create program files and run them. Typing in code interactively is a great way to experiment with the language, but for code that's more complex or that you will want to run more than once, you'll need to create program files and run them. But, before we go any further, let's test to see whether Ruby is installed. Bring up a fresh command prompt, and type this:[4]

```
$ ruby -v
ruby 1.9.1p0 (2009-01-30 revision 21907) [i386-darwin9.6.0]
```

If you believe that you should have Ruby installed and yet you get an error saying something like "ruby: command not found," then it is likely that the Ruby program is not in your path—the list of places that the shell searches for programs to run. If you used the Windows One-Click Installer, make sure you rebooted before trying this command. If you're on OS X and installed Ruby from source, you'll probably have to add a line like this to the file .profile in your home directory:

```
PATH=/usr/local/bin:$PATH
```

Interactive Ruby

One way to run Ruby interactively is simply to type ruby at the shell prompt. Here we typed in the single puts expression and an end-of-file character (which is Ctrl+D on our system). This process works, but it's painful if you make a typo, and you can't really see what's going on as you type.

```
% ruby
puts "Hello, world!"
^D
Hello, world!
```

4. Remember you may need to use ruby1.9 as the command name if you installed using a package management system.

The Very Latest Ruby

For those who just have to be on the very latest, hot-off-the-press, and *untested* cutting edge (as we were while writing this book), you can get development versions straight from the developers' working repository.

The Ruby developers use Subversion (often abbreviated as SVN) as their revision control system. Subversion clients can be downloaded from http://subversion.tigris.org/. You can check files out as an anonymous user from their archive by executing the following SVN command:

```
$ svn co http://svn.ruby-lang.org/repos/ruby/trunk ruby
```

The complete source code tree, just as the developers last left it, will now be copied to a `ruby` subdirectory on your machine.

This command will check out the head of the development tree. If you want the Ruby 1.8 branch, change `trunk` to `branches/ruby_1_8` in the checkout command.

For most folks, *irb*—Interactive Ruby—is the tool of choice for executing Ruby interactively. irb is a Ruby shell, complete with command-line history, line-editing capabilities, and job control. (In fact, it has its own chapter beginning on page 278.) You run irb from the command line. Once it starts, just type in Ruby code. It will show you the value of each expression as it evaluates it. Exit an irb session by typing exit or by using the end-of-file character on your operating system (normally Ctrl+D or Ctrl+Z).

```
% irb
irb(main):001:0> def sum(n1, n2)
irb(main):002:1>   n1 + n2
irb(main):003:1> end
=> nil
irb(main):004:0> sum(3, 4)
=> 7
irb(main):005:0> sum("cat", "dog")
=> "catdog"
irb(main):006:0> exit
```

We recommend that you get familiar with irb so you can try our examples interactively.

Ruby Programs

The normal way to write Ruby programs is to put them in one or more files. You'll use a text editor (Emacs, vim, TextMate, and so on) or an IDE (such as NetBeans) to create and maintain these files. You'll then run the files either from within the editor or IDE or from the command line. I personally use both techniques, typically running from within the editor for single-file programs and from the command line for more complex ones.

Let's start by creating a simple Ruby program and running it. Open a command window, and navigate to the pickaxe directory you created earlier:

```
$ cd ~/pickaxe            (unix)
C:\> cd \pickaxe          (windows)
```

Then, using your editor of choice, create the file myprog.rb, containing the following:

Download **samples/gettingstarted_2.rb**

```
puts "Hello, Ruby Programmer"
puts "It is now #{Time.now}"
```

(Note that the second string contains the text Time.now between curly braces, not parentheses.)

You can run a Ruby program from a file as you would any other shell script, Perl program, or Python program. Simply run the Ruby interpreter, giving it the script name as an argument:

```
$ ruby myprog.rb
Hello, Ruby Programmer
It is now 2009-04-13 13:25:51 -0500
```

On Unix systems, you can use the "shebang" notation as the first line of the program file:[5]

Download **samples/gettingstarted_4.rb**

```
#!/usr/local/bin/ruby -w
puts "Hello, Ruby Programmer"
puts "It is now #{Time.now}"
```

If you make this source file executable (using, for instance, chmod +x myprog.rb), Unix lets you run the file as a program:

```
$ ./myprog.rb
Hello, Ruby Programmer
It is now 2009-04-13 13:25:51 -0500
```

You can do something similar under Microsoft Windows using file associations, and you can run Ruby GUI applications by double-clicking their names in Explorer.

Ruby Documentation: RDoc and ri

As the volume of the Ruby libraries has grown, it has become impossible to document them all in one book; the standard library that comes with Ruby now contains more than 9,000

5. If your system supports it, you can avoid hard-coding the path to Ruby in the "shebang" line by using #!/usr/bin/env ruby, which will search your path for ruby and then execute it.

methods. Fortunately, an alternative to paper documentation exists for these methods (and classes and modules). Many are now documented internally using a system called *RDoc*.

If a source file is documented using RDoc, its documentation can be extracted and converted into HTML and ri formats.

Several websites contain a complete set of the RDoc documentation for Ruby, but http://www.ruby-doc.org is probably the best known. Browse on over, and you should be able to find at least some form of documentation for any Ruby library. The site is adding new documentation all the time.

The ri tool is a local, command-line viewer for this same documentation. Most Ruby distributions now also install the resources used by the ri program.

To find the documentation for a class, type ri *ClassName*. For example, the following lists the summary information for the GC class. (For a list of classes with ri documentation, type ri.)

```
$ ri GC
--------------------------------------------------------------- Class: GC
     The GC module provides an interface to Ruby's mark and sweep
     garbage collection mechanism. Some of the underlying methods are
     also available via the ObjectSpace module.
-------------------------------------------------------------------------

Class methods:
     count, disable, enable, malloc_allocated_size, malloc_allocations,
     start, stress, stress=

Instance methods:
     garbage_collect
```

For information on a particular method, give its name as a parameter:

```
% ri GC::enable
------------------------------------------------------- GC::enable
     GC.enable    => true or false
------------------------------------------------------------------
     Enables garbage collection, returning true if garbage
     collection was previously disabled.

        GC.disable   #=> false
        GC.enable    #=> true
        GC.enable    #=> false
```

If the method you pass to ri occurs in more than one class or module, ri will list all of the alternatives.

Reissue the command, prefixing the method name with the name of the class and a dot:

```
$ ri assoc
     More than one method matched your request.  You can refine your
     search by asking for information on one of:

Array#assoc [Ruby 1.9.1]
Array#rassoc [Ruby 1.9.1]
Hash#assoc [Ruby 1.9.1]
Hash#rassoc [Ruby 1.9.1]

$ ri Array.assoc
---------------------------------------------------------- Array#assoc
     array.assoc(obj)   ->  an_array  or  nil
----------------------------------------------------------------------
     Searches through an array whose elements are also arrays
     comparing obj with the first element of each contained array
     using obj.==. Returns the first contained array that matches
     (that is, the first associated array), or nil if no match is
     found. See also Array#rassoc.
       :   :   :
```

For general help on using ri, type ri --help. In particular, you might want to experiment with the --format option, which tells ri how to render decorated text (such as section headings). If your terminal program supports ANSI escape sequences, using --format ansi will generate a nice, colorful display. Once you find a set of options you like, you can set them into the RI environment variable. Using my shell (zsh), this would be done using the following:

```
% export RI="--format ansi --width 70"
```

If a class or module isn't yet documented in RDoc format, ask the friendly folks over at suggestions@ruby-doc.org to consider adding it.

All this command-line hacking may seem a tad off-putting if you're not a regular visitor to the shell prompt. But, in reality, it isn't that difficult, and the power you get from being able to string together commands this way is often surprising. Stick with it, and you'll be well on your way to mastering both Ruby and your computer.