In this chapter, you'll see:

- Installing Ruby, RubyGems, SQLite 3, and Rails
- Development environments and tools

CHAPTER 1

Installing Rails

In Part I of this book, we'll introduce you to both the Ruby language and the Rails framework. But we can't get anywhere until you've installed both and verified that they're operating correctly.

To get Rails running on your system, you need the following:

- A Ruby interpreter. Rails is written in Ruby, and you'll be writing your applications in Ruby too. Rails 7 will run on Ruby version 2.7, 3.0, and 3.1. It won't work on prior versions of Ruby.
- Ruby on Rails. This book was written using Rails version 7 (specifically, Rails 7.0.4).
- Some libraries, depending on the operating system.
- A database. We're using SQLite 3 in this book.

For a development machine, that's about all you'll need (apart from an editor, and we'll talk about editors separately). However, if you're going to deploy your application, you'll also need to install a production web server (as a minimum) along with some support code to let Rails run efficiently.

You can also install everything in a virtual machine. This can be good to isolate your Rails environment from your actual computer, though it will require a fair amount of disk space.

So how do you get all this installed? It depends on your choice of development environment. We'll go over three common choices: Windows, macOS, and Ubuntu Linux. For the Linux option, we'll show setup for using a virtual machine running Linux, so this is the version you want if you want complete isolation of your Rails development environment.

But before you dive in, recognize that for best results these instructions are meant for a fairly fresh, up-to-date, and clean machine. If this doesn't match you, consider doing your development in a Docker container[1] or with Vagrant.[2] If either of these options appeals to you, proceed with the Linux instructions that will follow.

A special note for Windows users: most Rails applications are developed on MacOS machines and deployed to Linux machines. Some tools you may want to use, like passenger and redis, don't work natively on Windows.

This puts Windows developers at a disadvantage, as much of the helpful advice you can find online won't be geared toward you. Fortunately Microsoft provides three tools that will provide you with an absolutely first-class developer environment:

- Windows Subsystem for Linux (WSL)[3]
- Windows Terminal[4]
- Visual Studio Code[5]

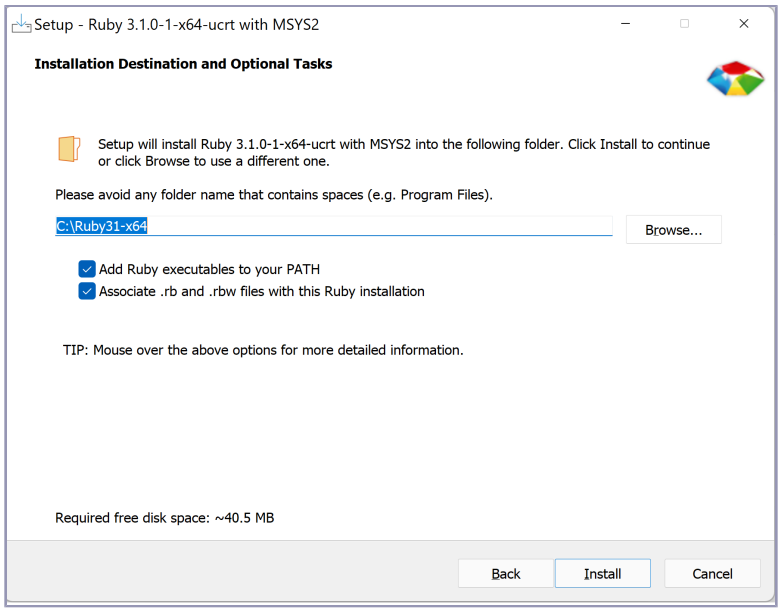
With WSL2, select the latest Ubuntu version and proceed with the Linux instructions that follow.

Installing on Windows

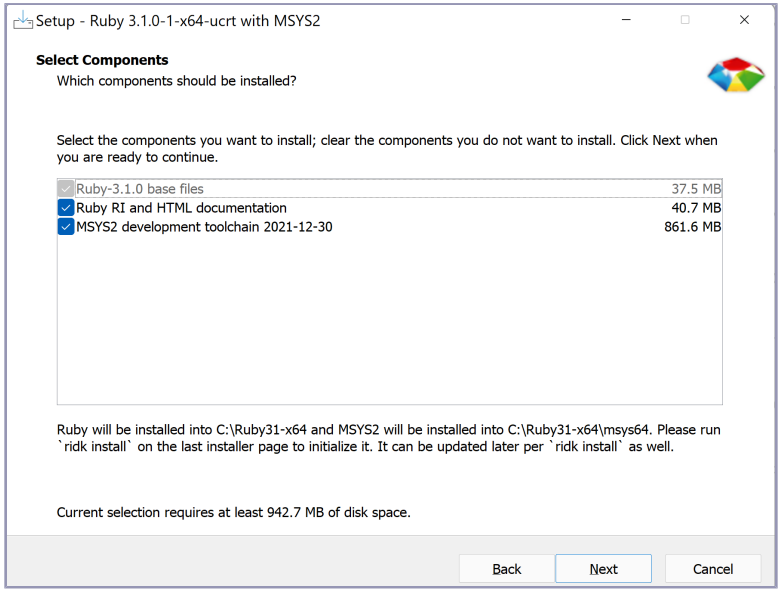
First, you need to install Ruby. We recommend using the RubyInstaller for Windows package.[6] At the time of this writing, the latest version of Ruby available via RubyInstaller is Ruby 3.1.3. If you use RubyInstaller, be sure to pick a version that includes Devkit. If you use a different installer, make sure you install `MSYS2` along with Ruby.

Installation is a snap. After you click Save/Download, click Run and then click OK. Select "I accept the License" (after reading it carefully, of course) and then click Next. Ensure "Add Ruby executables to your PATH" is selected, and click Install. See the screenshot on page 5.

1. https://docs.docker.com/samples/rails/
2. https://www.vagrantup.com/
3. https://docs.microsoft.com/en-us/windows/wsl/install
4. https://www.microsoft.com/en-us/p/windows-terminal/9n0dx20hk701
5. https://code.visualstudio.com/
6. http://rubyinstaller.org/downloads

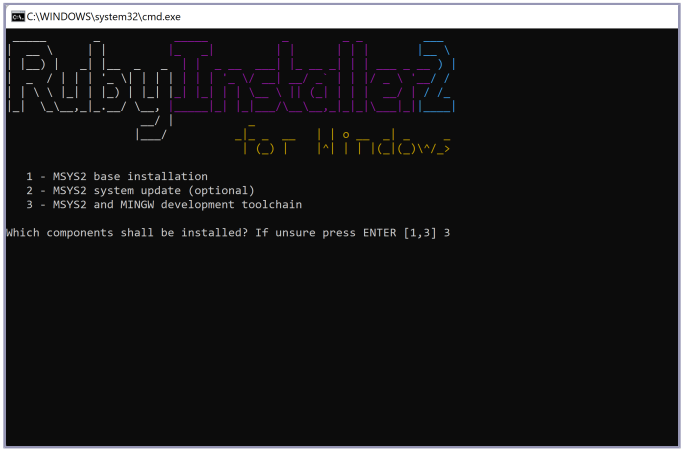


Next you'll need to select the components to be installed. Ensure that the MSYS2 development toolchain is selected. Click Next. See the following screenshot:

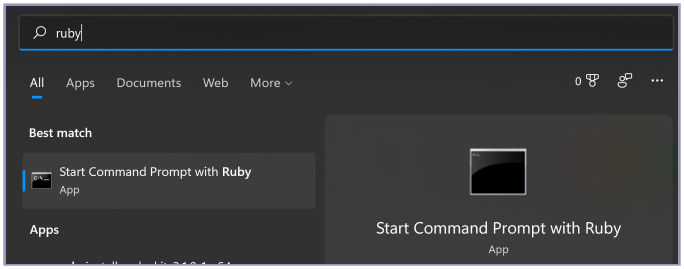


When you see the following screen, you'll be done with the first part of the installation. Click Finish to proceed to the next and final part.

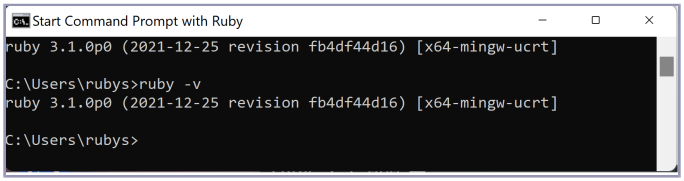
The next screen installs the development toolchains. Select option 3 and press `Enter`. This will take a while and ultimately prompt you again for which tools to install. The second time you're prompted, press `Enter` and the window will be dismissed.



From the Windows start screen you'll be able to find an app named Start Command Prompt with Ruby. Following is how this will look with Windows 11:



From this window you can verify that Ruby was installed correctly by entering the command `ruby -v` as follows:



Next, configure Git, adjusting the `user.name` and `user.email` as appropriate:

```
> git config --global user.name "John Doe"
> git config --global user.email johndoe@example.com
```

Before you proceed to install Rails itself, you'll need to upgrade the version of RubyGems that's provided by RubyInstaller to avoid a problem with missing tzinfo data.

```
> gem update --system
```

With this in place, proceed to installing Rails itself with the following command:

```
> gem install rails -v 7.0.4 --no-document
```

This will take a while. Once it completes, skip to Choosing a Rails Version, on page 13, to ensure that the version of Rails you've installed matches the version described in this edition. See you there.

Installing on macOS

Since macOS Monterey ships with Ruby 2.6.8, you'll need to download a newer version of Ruby that works with Rails 7. The easiest way to do this is to use Homebrew.[7]

7. https://brew.sh/

Before you start, go to your Utilities folder and drag the Terminal application onto your dock. You'll be using this during the installation and then frequently as a Rails developer. Open the terminal and run the following command:

```
$ /bin/bash -c "$(curl -fsSL \
  https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

You'll be asked you for your password and then to press Enter. Once the installation completes, it will output some next steps for you to take. At the present time, those steps are as follows:

```
$ echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> ~/.zprofile
$ eval "$(/opt/homebrew/bin/brew shellenv)"
```

Next, you have a choice. You can let Homebrew install the latest version of Ruby (currently Ruby 3.0.3p157). Or you can install rbenv[8] and install the version of Ruby of your choice.

Installing Ruby with Homebrew starts with a single command:

```
$ brew install ruby
```

Next you'll need to follow the post-installation instructions provided, which involve adding lines to your ~/.zshrc file:

```
export PATH="/opt/homebrew/lib/ruby/gems/3.0.0/bin:$PATH"
export PATH="/opt/homebrew/opt/ruby/bin:$PATH"
export LDFLAGS="-L/opt/homebrew/opt/ruby/lib"
export CPPFLAGS="-I/opt/homebrew/opt/ruby/include"
```

Alternatively, you can install rbenv and use it to install Ruby 3.1.3. Just be sure that you do not have RVM installed, as those two applications don't work well together.

Note that starting with macOS Catalina, the default shell is zsh, not bash as it had been historically. Assuming you're using either Catalina or Monterey and zsh, the Homebrew setup is as follows:

```
$ brew install rbenv ruby-build
$ rbenv init
$ echo 'eval "$(rbenv init -)"' >> ~/.zshrc
$ rehash
```

On older versions of macOS (or if you are using bash on Catalina or Monterey), the instructions are similar:

```
$ brew install rbenv ruby-build
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bash_profile
```

8. https://github.com/rbenv/rbenv#readme

```
$ ~/.rbenv/bin/rbenv init
$ hash -r
```

Once you've done that, restart your terminal by typing exit and hitting Return and then opening a new terminal window. After that, you can install Ruby like so:

```
$ rbenv install 3.1.3
$ rbenv global 3.1.3
```

If you had previously installed ruby-build and it can't find the definition for Ruby 3.1.3, you might need to reinstall ruby-build and try again:

```
$ brew reinstall --HEAD ruby-build
$ rbenv install 3.1.3
$ rbenv global 3.1.3
```

These are the two most popular routes for Mac developers. RVM and chruby are two other alternatives.[9,10]

Whichever path you take, run the following command to see which version of Ruby you're working with:

```
$ ruby -v
```

You should see the following type of result:

```
ruby 3.1.3p185 (2022-11-24 revision 1a6b16756e) [arm64-darwin22]
```

Next, run this command to update Rails to the version used by this book:

```
$ gem install rails -v 7.0.4 --no-document
```

While technically not required for development, installing redis now will prepare you for production later:

```
$ brew install redis
$ brew services start redis
```

OK, you OS X users are done. You can skip forward to join the Windows users in Choosing a Rails Version, on page 13. See you there.

Installing on Linux

Linux has many different distributions, each having its own method of installing software, along with various idiosyncracies around how it behaves. It would be too difficult to cover them all, so in this section, we'll outline how to get a Rails

9. https://rvm.io/rvm/install
10. https://github.com/postmodern/chruby#readme

environment running in a virtual machine running Ubuntu Linux. Most of the software we'll install would be needed on any Linux distribution, so if you aren't using Ubuntu, hopefully this will help you know what you need to set up.

Also note that if you're using Ubuntu but *not* a virtual machine, some of the behavior might be different, especially if your machine already has some packages installed on it.

With that disclaimer out of the way, our setup will require a few different steps. First, we'll set up a virtual machine using Vagrant (which you can skip if you already have Linux running on your computer). Then we'll install some system software that the Ruby and Rails development tools require, before finally installing Ruby and Rails.

Setting Up a Virtual Machine

Vagrant[11] is a system that can manage a virtual computer. Virtual Box[12] provides that virtual computer, and together they can mimic a real computer but do so to provide a totally clean, predictable, and isolated environment from your actual computer. We'll set up the virtual machine with Ubuntu Linux.

Of course, that means you need to install both Vagrant and Virtual Box on your computer, and how you do *that* depends on your operating system! Vagrant provides installation instructions,[13] and the best thing to do is to follow the instructions there. Once you've done this, head over to Virtual Box's installation page[14] and install that on your computer.

With those installed, we'll create a file to describe how Vagrant should set up your virtual machine. This file is called `Vagrantfile`. You'll need to create it somewhere on your computer where you prefer to work, such as your home directory or a subdirectory of it. Locate wherever that is and create `Vagrantfile` to have the following contents (comments in the code explain what each bit does, if you're curious):

```
# We want to use version '2' of Vagrant's configuration language
Vagrant.configure("2") do |config|

  # This is the operating system to use, in this case
  # Ubuntu Linux
  config.vm.box = "ubuntu/jammy64"
```

11. https://www.vagrantup.com
12. https://www.virtualbox.org
13. https://www.vagrantup.com/intro/getting-started/install.html
14. https://www.virtualbox.org/wiki/Downloads

```
  # This is configuration specifically for the virtual
  # machine, and this gives it 4G of memory
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "4096"
  end

  # When Rails runs on port 3000 inside your virtual machine, this
  # allows you to access it from a browser on your machine by
  # going to port 3000 on your machine.
  config.vm.network "forwarded_port", guest: 3000, host: 3000

  # This will mount your current directory on your computer
  # to the directory /files_on_your_computer inside the virtual machine
  config.vm.synced_folder ".", "/files_on_your_computer"
end
```

If you aren't in a command-line window at this point, open one up and change to the directory where you created this file. Then bring up the machine with the following command:

```
$ vagrant up
```

The very first time, it'll take a while for this command to complete. It has to download and install an entire operating system, so be patient. When it's done, you can log in to the virtual machine, like so:

```
$ vagrant ssh
```

You're now logged in to your virtual machine, so we can now start installing the software you'll need to learn Rails.

Installing System Software

If you're using a virtual machine, the commands that follow assume you've logged in to it with `vagrant ssh`. Otherwise, we'll assume you're logged in as a user who can execute `sudo`. Also note that in this case, you may have some software installed already. If you experience problems, you might want to update that software to the latest versions.

Many Ruby libraries are actually wrappers for C libraries, and when you install them, your system will try to build those libraries or build native connectors. This is the main reason we need certain software installed before we get to Ruby. First, refresh the list of packages available for your operating system:

```
$ sudo apt-get update
```

That will produce a large amount of output. Once that's done, we'll install several different libraries and tools. What this will look like on Ubuntu and most Debian-based Linuxes is as follows:

```
$ sudo apt-get install -y \
    build-essential \
    git \
    libsqlite3-dev \
    redis \
    ruby-dev \
    tzdata
```

For Centos Stream and most RedHat-based Linuxes, the following will get you started:

```
$ sudo yum install -y \
    gcc \
    git \
    redis \
    ruby-devel \
    sqlite-devel \
    which
$ sudo yum reinstall -y tzdata
```

Now we can install Ruby and Rails!

Installing Ruby and Rails

At this point, your system will have Ruby installed, though it may not be the version you need to run Rails. You need Ruby 2.7.3 or higher. The 2.7.0p0 version that comes with Ubuntu 20.04 is *not* sufficient; it will produce segfaults when you attempt to run tests.

If you're comfortable upgrading your system's Ruby from a third-party respository, you can add the BrightBox repository:[15]

```
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:brightbox/ruby-ng
$ sudo apt-get update
$ sudo apt-get upgrade
```

Note that BrightBox only contains up to to Ruby 2.7. If you want a later version of Ruby, or are concerned about there being unintended consequences to upgrading your system's Ruby, you can use rbenv[16] to install Ruby in parallel to your system Ruby. This also allows you to use many different versions of Ruby on the same computer but without disrupting the version of Ruby your

15. https://www.brightbox.com/docs/ruby/ubuntu/#adding-the-repository
16. https://github.com/rbenv/rbenv

system may depend on. rbenv is widely used for exactly this purpose. First, install it like so:

```
$ sudo apt install rbenv
$ rbenv init
```

Close your Terminal window and open a new one so your changes take effect. Verify that rbenv is properly set up using this rbenv-doctor script:

```
$ curl -fsSL \
    https://github.com/rbenv/rbenv-installer/raw/main/bin/rbenv-doctor \
    | bash -
```

If you're using another shell, consult the rbenv website for instructions if you aren't sure. Next, we'll install Ruby 3.1.3, the lastest version of Ruby 3.1 at the time of this writing:

```
$ rbenv install 3.1.3
```

This will take a long time, as it's downloading and compiling Ruby locally. Once that's done, you won't yet be using Ruby 3.1.3. To do that, you either need to tell rbenv to use Ruby 3.1.3 in the current directory or globally. To avoid confusion, we'll do it globally, meaning that rbenv should use Ruby 3.1.3 if it doesn't know what other version to use. We do that like so:

```
$ rbenv global 3.1.3
```

With this done, you can try running `ruby -v` on the command line. You should see 3.1.3 in the output.

Next, we'll install Rails itself. Rails is a RubyGem, and Ruby comes with the command `gem` which installs RubyGems. We'll use that to install Rails, like so:

```
$ gem install rails -v 7.0.4 --no-document
```

When that completes, you can verify it worked by running `rails --version`. You should see 7.0.4 in the output.

This completes the setup of Ruby and Rails. The rest of this chapter will outline other software you might need to do development.

Choosing a Rails Version

The previous instructions helped you install the version of Rails used by the examples in this book. But occasionally you might not want to run that version. For example, a newer version with some fixes or new features might become available. Or perhaps you're developing on one machine but intending

to deploy on another machine that contains a version of Rails that you don't have any control over.

If either of these situations applies to you, you need to be aware of a few things. For starters, you can use the gem command to find out all the versions of Rails you have installed:

```
$ gem list --local rails
```

You can also verify which version of Rails you're running as the default by using the rails --version command. It should return 7.0.4.

If it doesn't, insert the version of Rails surrounded by underscores before the first parameter of any rails command. Here's an example:

```
$ rails _7.0.4_ --version
```

This is particularly handy when you create a new application, because once you create an application with a specific version of Rails, it'll continue to use that version of Rails—even if newer versions are installed on the system—until *you* decide it's time to upgrade. To upgrade, simply update the version number in the Gemfile that's in the root directory of your application and run bundle install.

Setting Up Your Development Environment

The day-to-day business of writing Rails programs is pretty straightforward. Everyone works differently; here's how we work.

The Command Line

We do a lot of work at the command line. Although an increasing number of GUI tools help generate and manage a Rails application, we find the command line is still the most powerful place to be. It's worth spending a little while getting familiar with the command line on your operating system. Find out how to use it to edit commands that you're typing, how to search for and edit previous commands, and how to complete the names of files and commands as you type.

So-called tab completion is standard on Unix shells such as bash and zsh. It allows you to type the first few characters of a filename, hit Tab, and have the shell look for and complete the name based on matching files.

Version Control

We keep all our work in a version control system (currently Git). We make a point of checking a new Rails project into Git when we create it and committing

changes once we've passed the tests. We normally commit to the repository many times an hour.

If you're not familiar with Git, don't worry, because this book will introduce you to the few commands that you'll need to follow along with the application being developed. If you ever need it, extensive documentation is available online.[17]

If you're working on a Rails project with other people, consider setting up a continuous integration (CI) system. When anyone checks in changes, the CI system will check out a fresh copy of the application and run all the tests. It's a common way to ensure that accidental breakages get immediate attention. You can also set up your CI system so that your customers can use it to play with the bleeding-edge version of your application. This kind of transparency is a great way to ensure that your project isn't going off the tracks.

Editors

We write our Rails programs using a programmer's editor. We've found over the years that different editors work best with different languages and environments. For example, Dave originally wrote this chapter using Emacs because he thinks that its Filladapt mode is unsurpassed when it comes to neatly formatting XML as he types. Sam updated the chapter using Vim. But many think that neither Emacs nor Vim is ideal for Rails development. Although the choice of editor is a personal one, here are some suggestions for features to look for in a Rails editor:

- Support for syntax highlighting of Ruby and HTML—ideally, support for `.erb` files (a Rails file format that embeds Ruby snippets within HTML).
- Support for automatic indentation and reindentation of Ruby source. This is more than an aesthetic feature: having an editor indent your program as you type is the best way to spot bad nesting in your code. Being able to reindent is important when you refactor your code and move stuff. (TextMate's ability to reindent when it pastes code from the clipboard is convenient.)
- Support for insertion of common Ruby and Rails constructs. You'll be writing lots of short methods, and if the IDE creates method skeletons with a keystroke or two, you can concentrate on the interesting stuff inside.
- Good file navigation. As you'll see, Rails applications are spread across many files; for example, a newly created Rails application enters the world containing seventy-seven files spread across forty-five directories. That's before you've written a thing.

17. https://git-scm.com/book/en/v2

You need an environment that helps you navigate quickly among these. You'll add a line to a controller to load a value, switch to the view to add a line to display it, and then switch to the test to verify you did it all right. Something like Notepad, where you traverse a File Open dialog box to select each file to edit, won't cut it. We prefer a combination of a tree view of files in a sidebar, a small set of keystrokes that help us find a file (or files) in a directory tree by name, and some built-in smarts that know how to navigate (say) between a controller action and the corresponding view.

- Name completion. Names in Rails tend to be long. A nice editor will let you type the first few characters and then suggest possible completions to you at the touch of a key.

We hesitate to recommend specific editors because we've used only a few in earnest and we'll undoubtedly leave someone's favorite editor off the list. Nevertheless, to help you get started with something other than Notepad, here are some suggestions:

- Visual Studio Code is a free editor built on open source that runs everywhere.[18]
- TextMate is the favorite of many programmers who prefer to do their development on macOS, including David Heinemeier Hansson.[19]
- Sublime Text is a cross-platform alternative that some see as the de facto successor to TextMate.[20]
- rails.vim is a Vim/NeoVim plugin for editing Ruby on Rails applications.[21]
- RubyMine is a commercial IDE for Ruby and is available for free to qualified educational and open source projects.[22] It runs on Windows, macOS, and Linux.

Ask experienced developers who use your kind of operating system which editor they use. Spend a week or so trying alternatives before settling in.

The Desktop

We're not going to tell you how to organize your desktop while working with Rails, but we'll describe what we do.

18. https://code.visualstudio.com/
19. http://macromates.com/
20. http://www.sublimetext.com/
21. https://github.com/tpope/vim-rails
22. http://www.jetbrains.com/ruby/features/index.html

Where's My IDE?

If you're coming to Ruby and Rails from languages such as C# and Java, you may be wondering about IDEs. After all, we all know that it's impossible to code modern applications without at least 100 MB of IDE supporting our every keystroke. For you enlightened ones, here's the point in the book where we recommend you sit down—ideally propped up on each side by a pile of framework references and 1,000-page Made Easy books.

It may surprise you to know that most Rails developers don't use fully fledged IDEs for Ruby or Rails (although some of the environments come close). Indeed, many Rails developers use plain old editors. And it turns out that this isn't as much of a problem as you might think. With other less expressive languages, programmers rely on IDEs to do much of the grunt work for them because IDEs do code generation, assist with navigation, and compile incrementally to give early warning of errors.

With Ruby, however, much of this support isn't necessary. Editors such as TextMate and BBEdit give you 90 percent of what you'd get from an IDE but are far lighter weight. About the only useful IDE facility that's missing is refactoring support.

Most of the time, we're writing code, running tests, and poking at an application in a browser. So, our main development desktop has an editor window and a browser window permanently open. We also want to keep an eye on the logging that's generated by the application, so we keep a terminal window open. In it, we use `tail -f` to scroll the contents of the log file as it's updated. We normally run this window with a small font so it takes up less space. If we see something interesting flash by, we increase the font size to investigate.

Alternately, you can use `less +F` to scroll through messages. This has the advantage of being able to exit the follow mode by pressing Ctrl-C, at which point you can do searches by typing / followed by the string you want to search for.

Windows developers should take a look at Windows Terminal.[23]

We also need access to the Rails API documentation, which we view in a browser. In the Introduction, we talked about using the `gem server` command to run a local web server containing the Rails documentation. This is convenient, but it unfortunately splits the Rails documentation across a number of separate documentation trees. If you're online, you can see a consolidated view of all the Rails documentation in one place.[24]

23. https://www.microsoft.com/en-us/p/windows-terminal/9n0dx20hk701
24. http://api.rubyonrails.org/

Rails and Databases

The examples in this book were written using SQLite 3 (version 3.36.0 or thereabouts). If you want to follow along with our code, it's probably simplest if you use SQLite 3 as well. If you decide to use something else, it won't be a major problem. You may have to make minor adjustments to any explicit SQL in our code, but Rails pretty much eliminates database-specific SQL from applications.

If you want to connect to a database other than SQLite 3, Rails also works with DB2, MySQL, Oracle Database, Postgres, Firebird, and SQL Server. For all but SQLite 3, you'll need to install a database driver—a library that Rails can use to connect to and use with your database engine. This section contains links to instructions to get that done.

The database drivers are all written in C and are primarily distributed in source form. If you don't want to bother building a driver from source, take a careful look at the driver's website. Many times you'll find that the author also distributes binary versions.

If you can't find a binary version or if you'd rather build from source anyway, you need a development environment on your machine to build the library. For Windows, you need a copy of Visual C++. For Linux, you need `gcc` and friends (but these will likely already be installed).

On OS X, you need to install the developer tools (they come with the operating system but aren't installed by default). You also need to install your database driver into the correct version of Ruby. If you installed your own copy of Ruby, bypassing the built-in one, it's important to have this version of Ruby first in your path when building and installing the database driver. You can use the `which ruby` command to make sure you're *not* running Ruby from `/usr/bin`.

The following are the available database adapters and the links to their respective home pages:

DB2	https://rubygems.org/gems/ibm_db/
Firebird	https://rubygems.org/gems/fireruby
MySQL	https://rubygems.org/gems/mysql2
Oracle Database	https://rubygems.org/gems/activerecord-oracle_enhanced-adapter
Postgres	https://rubygems.org/gems/pg
SQL Server	https://github.com/rails-sqlserver
SQLite	https://github.com/luislavena/sqlite3-ruby

MySQL and SQLite adapters are also available for download as RubyGems (mysql2 and sqlite3, respectively).

What We Just Did

- We installed (or upgraded) the Ruby language.
- We installed (or upgraded) the Rails framework.
- We selected an editor.
- We installed (or upgraded) the SQLite 3 database.

Now that we have Rails installed, let's use it. It's time to move on to the next chapter, where you'll create your first application.