第 1 章

ターミナルの使い方

Rails による Web アプリ開発では、ターミナルというソフトウェアが重要な役割を果たします。本章では、ターミナルの起動方法とシェルコマンドの実行方法について学習します。

1.1　ターミナルの起動方法

OS X にはターミナルというソフトウェアが標準で備わっています。Finder の「アプリケーション」→「ユーティリティ」からターミナルを起動できます。図 1.1 のような外観をしています。

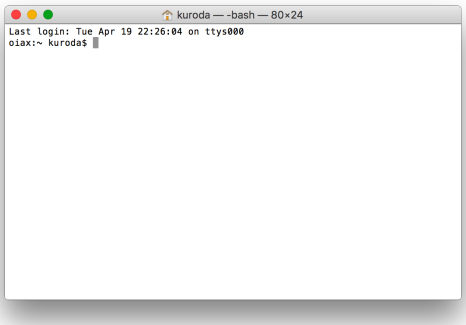

図 1.1　OS X のターミナル

Ubuntu の場合は同種のソフトウェアに「端末」という名前が付いていますが、本書では OS X に合わせて「ターミナル」と呼ぶことにします。

端末を起動するには、Dash[*1]に対して「term」というキーワードを入力し、検索結果の中から「端末」を探してクリックします[*2]。

その外観は図 1.2 のようになります。

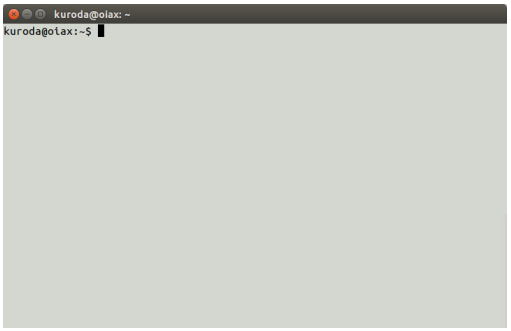

図 1.2　Ubuntu の標準ターミナル

1.2　シェルコマンドの実行

私たちはこのターミナルを用いて**コマンド**（command）を入力します。コマンドを入力した後で Enter キーを押すことを、「コマンドを実行する」と表現します。

ターミナルに入力されたコマンドを解釈し、OS を操作するソフトウェアを**シェル**（shell）と呼びます。シェルはターミナルの上で動いています。

シェルには *sh*、*tcsh*、*zsh* など多数の種類がありますが、OS X と Ubuntu の標準シェルは *bash* です。

シェルが受け付けるコマンドを特に**シェルコマンド**（shell command）と呼びます。本書では、シェルコマンドを次のような書式で表示します。

[*1] Ubuntu のデスクトップ左上隅にある Ubuntu ロゴマークをクリックするとアプリケーションやファイルを検索するウィンドウが現れます。これを Dash と呼びます。

[*2] Ubuntu の端末には Ctrl + Shift + T というキーボードショートカットが用意されています。キーボードで Ctrl、Shit、T の各キーを同時に押してください。この組み合わせを覚えれば、Dash で検索するよりも素早く端末を起動できます。

```
$ mkdir work
```

行頭の $ は、シェルがユーザーからのコマンド入力を待っている状態を表す記号で、**コマンドプロンプト**（ command prompt）と呼ばれます。実際にターミナルに表示されるコマンドプロンプトは bash-3.2$ や kuroda@oiax:~$ などの形をしていますが、この本の中では単に $ という記号で表します。

上記の例では、mkdir work がシェルコマンドです。読者の皆さんがターミナルでコマンドを入力する際には、行頭の $ 記号を入力しないように注意してください。

1.3 コマンドの本体と引数

コマンドは本体と**引数**（ ひきすう ）に分かれます。先ほどの例では mkdir がコマンド本体で、work が引数です。英語では引数を arguments と呼びます。「 数」という漢字を含みますが数値とは無関係で、work のようなアルファベットの列、~のような記号、100 のような数字の列などを引数として指定できます。

コマンドの本体と引数の間はスペースで区切ります。複数の引数を指定する場合は、次の例のように引数と引数の間をスペースで区切ります。

```
$ mv foo bar
```

このとき、foo を第一引数、bar を第二引数と呼びます。

1.4 文字列について

プログラミング用語で foo や bar のような文字の連なりを**文字列**と呼びます。英語では *string* といいます。

シェルコマンドの引数として指定したい文字列の中にスペースが含まれる場合は、次のようにダブルクオート（ "）で囲みます。

```
$ echo "Ruby on Rails"
```

ダブルクオートの代わりにシングルクオート（ ’）を用いることも可能です。

```
$ echo ’Ruby on Rails’
```

1.5 *sudo* コマンド

Unix 系 OS には、root という名前を持つ特別なユーザーアカウントが存在します。root はあらゆるコマンドを実行でき、あらゆるファイルを変更･削除できます。root で誤った操作を行うとコンピュータに深刻な影響を与えかねないため、文書作成やプログラミングなどの日常的な作業は root 以外の一般ユーザーで行います。

ここまで私たちがターミナルに入力してきたコマンドはすべて一般ユーザーの権限で実行されています。root でないと実行できない（ root 権限の必要な）コマンドをターミナルで実行したい場合は、*sudo* コマンドに続けてそのコマンドを入力します。

試しに、ターミナルで次のコマンドを実行してみてください。

```
$ cat /etc/sudoers
```

すると「 cat: /etc/sudoers: 許可がありません」というエラーメッセージが表示されます。このファイルは root ユーザーの持ち物で、一般ユーザー

には閲覧権限がありません。しかし、

```
$ sudo cat /etc/sudoers
```

というコマンドを入力すると「 [sudo] password for kuroda:」とパスワードの入力が求められ、あなた自身のパスワードを入力すれば `/etc/sudoers` ファイルの中身をターミナルに出力することができます。

なお、この *sudo* コマンドは誰でも自由に使えるわけではありません。*sudo* コマンドを使う権限を与えられていないユーザーがこのコマンドを実行すると、次のようなエラーメッセージがターミナルに表示されます[*3]。

```
kuroda is not in the sodoers file.  This incident will be reported.
```

[*3] OS X や Ubuntu の初期設定時に作成したユーザーアカウントには *sudo* コマンドを使う権限が与えらています。

第 12 章

bundle コマンドの使い方

本章では、*bundle* コマンドを用いて Rails アプリケーションが依存する Gem パッケージ群を一括してインストールまたは更新する方法について解説します。

12.1 Bundler とは

Bundler（バンドラー）は、特定の Rails アプリケーションが依存する（必要とする）Gem パッケージ群を一括してインストールまたは更新するツールです。

> 正確に言えば、Bundler は Rails アプリケーション専用のツールではなく、Ruby 言語で書かれたあらゆるアプリケーションで使用できます。

Rails アプリケーションを配布したり、本番環境にデプロイ（設置）したりするとき、Bundler が重要な役割を果たします。また、複数人からなるチームで Rails 開発を行うときには、Bundler のおかげで全員の開発環境を一致させることができます。

12.2 *bundle* コマンド

Bundler が提供するシェルコマンドが *bundle* です。語末の“r”が抜けている点に注意してください。「包む」あるいは「束ねる」という意味を持つ英語の動詞です。

12.2.1 `install` サブコマンド

bundle の `install` サブコマンドは、Rails アプリケーションが依存する Gem パッケージ群を一括してインストールします。

> サブコマンドを省略して単に *bundle* とだけターミナルに入力すると、`install` サブコマンドが実行されます。

インストールの流れはソースツリーのルートディレクトリに `Gemfile.lock` というファイルがあるかどうかで変化します。Rails アプリケーションの骨格を作った直後はこのファイルが存在しません。

`Gemfile.lock` がない場合、`Gemfile` というファイルの内容を解析して、Gem パッケージ同士の依存関係を調べ、できるかぎり新しいバージョンの Gem ファイルの組み合わせをインストールします。そして、その情報を `Gemfile.lock` に書き込みます。

`Gemfile.lock` がある場合は、`Gemfile` の解析をスキップし、`Gemfile.lock` の情報に基づいて Gem パッケージ群をインストールします。

あなたが作った Rails アプリケーションのソースコードを別の環境にコピーする際は、この `Gemfile.lock` を含めてください。そうすることで、開発の時に利用したバージョンの Gem パッケージ群がコピー先の環境でもインストールされることになります。

Railsアプリケーションのソースコードを第6章で紹介した *git* コマンドで管理する場合、必ずリポジトリに `Gemfile.lock` を登録してください。

12.2.2 `update` サブコマンド

bundle の `update` サブコマンドは、`Gemfile` に記述されたGemパッケージ群の一部または全部を更新します。

引数にGemパッケージのリストを指定すれば、それらが更新されます。例えば、`rails` と `rake` とそれらが依存するGemパッケージだけを更新するには、次のコマンドを実行してください。

```
$ bundle update rails rake
```

引数を省略すると、`Gemfile` にある全Gemパッケージが更新の対象となります。

```
$ bundle update
```

12.3 Bundler の仕組み

この節には、少し高度な内容が含まれています。Rails の学習を始めるにあたっては必ずしも必要ではありませんので、初心者の方はいったん読み飛ばしても構いません。あなたの Rails アプリケーションに新しい Gem パッケージを導入する際に読み返してください。

12.3.1 Gemfile

ある Rails アプリケーションが依存する Gem パッケージのリストを Bundler が得るための情報源が、`Gemfile` という名前のテキストファイルです。前章で作成した Rails アプリケーション AirBoy のソースコードにも含まれています。テキストエディタで開いてみましょう。

次に示すのは、その `Gemfile` からの抜粋です。ただし、シャープ記号(`#`)で始まるコメント行は省略しています。

```
source 'https://rubygems.org'

gem 'rails', '>= 5.0.0.rc1', '< 5.1'
gem 'sqlite3'
gem 'puma', '~> 3.0'
gem 'sass-rails', '~> 5.0'
gem 'uglifier', '>= 1.3.0'
gem 'coffee-rails', '~> 4.1.0'
```

`Gemfile` に書かれているのは Ruby プログラムで、各行の冒頭にある `source` や `gem` といった文字列は Ruby のメソッドです。

`source` メソッドの引数には、Bundler が Gem パッケージをダウンロードする元の URL を指定します。この値は基本的に

'https://rubygems.org' 固定で、変更する必要はありません。

gem メソッドの第 1 引数には、Gem パッケージの名前を指定します。必要に応じて、第 2 引数以降に次節で説明する**バージョン指定子**（ version specifier）を指定します。

Bundler は Gemfile から得られた情報を解析して、できるかぎり新しいバージョンの Gem パッケージの組み合わせを見つけ出します。ただし、すべての Gem パッケージが最新版になるとは限りません。Gem パッケージ同士の間にも依存関係があり、古いバージョンの Gem パッケージが必要とされることがあるからです。

■コラム: 依存関係の解決に失敗する場合

bundle コマンドを実行した時、次のようなエラーメッセージが出ることがあります。ただし、X の部分には任意の Gem パッケージ名が入ります。

```
Bundler could not find compatible versions for gem "X":
```

これは Bundler が依存関係の解決に失敗したことを意味します。例えば、A および B という二つの Gem パッケージがともに X に依存しているとします。もしここで、A がバージョン 1.0 以上 2.0 未満の X を要求していて、B がバージョン 2.0 以上の X を要求しているならば、Bundler は X のバージョンを決定できません。

このような場合、私たち自身が A をアップグレードしたり、B をダウングレードしたりして、調整する必要があります。

12.3.2 バージョン指定子

バージョン指定子は、>= のような演算子と 5.0.0.rc1 のようなバージョン番号により構成されます。演算子は =、!=、>、<、>=、<=、~> の 7 種類です。最後の ~> 以外の演算子の意味は明白です。

演算子 ~> は**悲観的バージョン演算子**と呼ばれ、ある Ruby アプリケーションが、未来にリリースされるバージョンの Gem パッケージでも動作することを保証するために使われます。

例えば、あなたのアプリケーションが Gem パッケージ nokogiri のバージョン 1.6.0 を利用しているとします。バージョン 2.0 をリリースするまで nokogiri の作者が Gem パッケージの後方互換性[*1]を維持するだろうとあなたが考えるのであれば、Gemfile で次のように書くことができます。

```
gem 'nokogiri', '~> 1.6'
```

悲観的バージョン演算子は、バージョン番号の最後の桁のアップグレードだけを許容します。このように書いておけば、Bundler は nokogiri のバージョン 1.7 や 1.8 がリリースされたらアップグレードするけれど、2.0 がリリースされてもアップグレードしません。

しかし、nokogiri のバージョンを 1.7.0 未満にとどめておきたいのであれば、Gemfile には次のように記述します。

```
gem 'nokogiri', '~> 1.6.0'
```

この場合、Bundler は nokogiri のバージョン 1.6.1 がリリースされたらアップグレードするけれど、1.7.0 がリリースされてもアップグレードしま

[*1] 新しいバージョンの Gem パッケージで古いバージョンと同じ機能やデータ形式が使える状態を、後方互換性（ backward compatibility）が維持されると表現します。

せん。

12.3.3 Gemfile.lock

bundle コマンドで Rails アプリケーションが依存する Gem パッケージ群を一括してインストールすると、ソースコードツリーのルートディレクトリに `Gemfile.lock` という名前のファイルが作られます。

Bundler は、`Gemfile` を解析して得られた Gem ファイル同士の依存関係に関する情報をこのファイルに記録します。基本的に私たち Rails プログラマーが `Gemfile.lock` を書き換えることはないので、その中身がどうなっているかを気にする必要はありません。

ただし、ひとつだけ知っておいた方がいい点があります。それは、`Gemfile.lock` の末尾につぎのような形式で、Bundler のバージョン番号が記録されていることです。

```
BUNDLED WITH
   1.12.4
```

私たちが `bundle install` または `bundle update` コマンドを実行すると、その時点での Bundler のバージョン番号がここに書き込まれます。ただし、元々書かれていたバージョン番号の方が新しい場合は、次のような警告メッセージが表示されます。

> Warning: the running version of Bundler is older than the version that created the lockfile. We suggest you upgrade to the latest version of Bundler by running ‘gem install bundler‘.

この場合、`Gemfile.lock` 末尾のバージョン番号は書き換わりません。なお、`gem install bundler` コマンドを実行して Bundler をアップグレードすればこの警告は出なくなります。複数人からなるチームで Rails アプリケーション開発を行っている場合、時々この警告を見ることになります。あわてずに対応してください。