Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'gemfile.5' command

$ man gemfile.5

GEMFILE(5) GEMFILE(5)

NAME

Gemfile - A format for describing gem dependencies for Ruby programs

SYNOPSIS

A Gemfile describes the gem dependencies required to execute associated Ruby code.

Place the Gemfile in the root of the directory containing the associ? ated code. For instance, in a Rails application, place the Gemfile in the same directory as the Rakefile.

SYNTAX

A Gemfile is evaluated as Ruby code, in a context which makes available a number of methods used to describe the gem requirements.

GLOBAL SOURCES

At the top of the Gemfile, add a line for the Rubygems source that con? tains the gems listed in the Gemfile.

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

It is possible, but not recommended as of Bundler 1.7, to add multiple global source lines. Each of these sources MUST be a valid Rubygems repository.

Sources are checked for gems following the heuristics described in SOURCE PRIORITY. If a gem is found in more than one global source, Bundler will print a warning after installing the gem indicating which source was used, and listing the other sources where the gem is avail?

able. A specific source can be selected for gems that need to use a non-standard repository, suppressing this warning, by using the :source option or a source block.

CREDENTIALS

Some gem sources require a username and password. Use bundle config(1) bundle-config.1.html to set the username and password for any of the sources that need it. The command must be run once on each computer that will install the Gemfile, but this keeps the credentials from be? ing stored in plain text in version control.

```
bundle config gems.example.com user:password
```

For some sources, like a company Gemfury account, it may be easier to include the credentials in the Gemfile as part of the source URL.

```
source "https://user:password@gems.example.com"
```

Credentials in the source URL will take precedence over credentials set using config.

RUBY

If your application requires a specific Ruby version or engine, specify your requirements using the ruby method, with the following arguments. All parameters are OPTIONAL unless otherwise specified.

VERSION (required)

The version of Ruby that your application requires. If your application requires an alternate Ruby engine, such as JRuby, Rubinius or Truf? fleRuby, this should be the Ruby version that the engine is compatible with.

```
ruby "1.9.3"
```

ENGINE

Each application may specify a Ruby engine. If an engine is specified, an engine version must also be specified.

What exactly is an Engine? - A Ruby engine is an implementation of the Ruby language.

? For background: the reference or original implementation of the Ruby programming language is called Matz?s Ruby Interpreter https://en.wikipedia.org/wiki/Ruby_MRI, or MRI for short. This is

named after Ruby creator Yukihiro Matsumoto, also known as Matz. MRI is also known as CRuby, because it is written in C. MRI is the most widely used Ruby engine.

? Other implementations https://www.ruby-lang.org/en/about/ of Ruby exist. Some of the more well-known implementations include Rubinius https://rubinius.com/, and JRuby http://jruby.org/. Rubinius is an alternative implementation of Ruby written in Ruby. JRuby is an im? plementation of Ruby on the JVM, short for Java Virtual Machine.

ENGINE VERSION

Each application may specify a Ruby engine version. If an engine ver? sion is specified, an engine must also be specified. If the engine is "ruby" the engine version specified must match the Ruby version.

```
ruby "1.8.7", :engine => "jruby", :engine_version => "1.6.7"
```

PATCHLEVEL

Each application may specify a Ruby patchlevel.

```
ruby "2.0.0", :patchlevel => "247"
```

GEMS

Specify gem requirements using the gem method, with the following argu? ments. All parameters are OPTIONAL unless otherwise specified.

NAME (required)

For each gem requirement, list a single gem line.

```
gem "nokogiri"
```

VERSION

Each gem MAY have one or more version specifiers.

```
gem "nokogiri", ">= 1.4.2"
gem "RedCloth", ">= 4.1.0", "< 4.2.0"
```

REQUIRE AS

Each gem MAY specify files that should be used when autorequiring via Bundler.require. You may pass an array with multiple files or true if the file you want required has the same name as gem or false to prevent any file from being autorequired.

```
gem "redis", :require => ["redis/connection/hiredis", "redis"]
gem "webmock", :require => false
```

gem "byebug", :require => true

The argument defaults to the name of the gem. For example, these are identical:

gem "nokogiri"

gem "nokogiri", :require => "nokogiri"

gem "nokogiri", :require => true

GROUPS

Each gem MAY specify membership in one or more groups. Any gem that does not specify membership in any group is placed in the default group.

gem "rspec", :group => :test

gem "wirble", :groups => [:development, :test]

The Bundler runtime allows its two main methods, Bundler.setup and Bundler.require, to limit their impact to particular groups.

```
# setup adds gems to Ruby?s load path
Bundler.setup                  # defaults to all groups
require "bundler/setup"        # same as Bundler.setup
Bundler.setup(:default)        # only set up the _default_ group
Bundler.setup(:test)           # only set up the _test_ group (but `not` _default_)
Bundler.setup(:default, :test) # set up the _default_ and _test_ groups, but no others
# require requires all of the gems in the specified groups
Bundler.require                # defaults to the _default_ group
Bundler.require(:default)      # identical
Bundler.require(:default, :test) # requires the _default_ and _test_ groups
Bundler.require(:test)         # requires the _test_ group
```

The Bundler CLI allows you to specify a list of groups whose gems bun? dle install should not install with the without configuration.

To specify multiple groups to ignore, specify a list of groups sepa? rated by spaces.

bundle config set --local without test

bundle config set --local without development test

Also, calling Bundler.setup with no parameters, or calling require "bundler/setup" will setup all groups except for the ones you excluded

via --without (since they are not available).

Note that on bundle install, bundler downloads and evaluates all gems, in order to create a single canonical list of all of the required gems and their dependencies. This means that you cannot list different ver? sions of the same gems in different groups. For more details, see Un? derstanding Bundler https://bundler.io/rationale.html.

PLATFORMS

If a gem should only be used in a particular platform or set of plat? forms, you can specify them. Platforms are essentially identical to groups, except that you do not need to use the --without install-time flag to exclude groups of gems for other platforms.

There are a number of Gemfile platforms:

ruby C Ruby (MRI), Rubinius or TruffleRuby, but NOT Windows

mri Same as ruby, but only C Ruby (MRI)

mingw Windows 32 bit ?mingw32? platform (aka RubyInstaller)

x64_mingw

Windows 64 bit ?mingw32? platform (aka RubyInstaller x64)

rbx Rubinius

jruby JRuby

truffleruby

TruffleRuby

mswin Windows

You can restrict further by platform and version for all platforms ex? cept for rbx, jruby, truffleruby and mswin.

To specify a version in addition to a platform, append the version num? ber without the delimiter to the platform. For example, to specify that a gem should only be used on platforms with Ruby 2.3, use:

ruby_23

The full list of platforms and supported versions includes:

ruby 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6

mri 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6

mingw 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6

x64_mingw

2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6

As with groups, you can specify one or more platforms:

```
gem "weakling",   :platforms => :jruby
gem "ruby-debug", :platforms => :mri_18
gem "nokogiri",   :platforms => [:mri_18, :jruby]
```

All operations involving groups (bundle install bundle-install.1.html, Bundler.setup, Bundler.require) behave exactly the same as if any groups not matching the current platform were explicitly excluded.

SOURCE

You can select an alternate Rubygems repository for a gem using the ?:source? option.

```
gem "some_internal_gem", :source => "https://gems.example.com"
```

This forces the gem to be loaded from this source and ignores any global sources declared at the top level of the file. If the gem does not exist in this source, it will not be installed.

Bundler will search for child dependencies of this gem by first looking in the source selected for the parent, but if they are not found there, it will fall back on global sources using the ordering described in SOURCE PRIORITY.

Selecting a specific source repository this way also suppresses the am? biguous gem warning described above in GLOBAL SOURCES (#source).

Using the :source option for an individual gem will also make that source available as a possible global source for any other gems which do not specify explicit sources. Thus, when adding gems with explicit sources, it is recommended that you also ensure all other gems in the Gemfile are using explicit sources.

GIT

If necessary, you can specify that a gem is located at a particular git repository using the :git parameter. The repository can be accessed via several protocols:

HTTP(S)

```
gem "rails", :git => "https://github.com/rails/rails.git"
```

SSH `gem "rails", :git => "git@github.com:rails/rails.git"`

git gem "rails", :git => "git://github.com/rails/rails.git"

If using SSH, the user that you use to run bundle install MUST have the appropriate keys available in their $HOME/.ssh.

NOTE: http:// and git:// URLs should be avoided if at all possible. These protocols are unauthenticated, so a man-in-the-middle attacker can deliver malicious code and compromise your system. HTTPS and SSH are strongly preferred.

The group, platforms, and require options are available and behave exactly the same as they would for a normal gem.

A git repository SHOULD have at least one file, at the root of the directory containing the gem, with the extension .gemspec. This file MUST contain a valid gem specification, as expected by the gem build command.

If a git repository does not have a .gemspec, bundler will attempt to create one, but it will not contain any dependencies, executables, or C extension compilation instructions. As a result, it may fail to properly integrate into your application.

If a git repository does have a .gemspec for the gem you attached it to, a version specifier, if provided, means that the git repository is only valid if the .gemspec specifies a version matching the version specifier. If not, bundler will print a warning.

```
gem "rails", "2.3.8", :git => "https://github.com/rails/rails.git"
# bundle install will fail, because the .gemspec in the rails
# repository's master branch specifies version 3.0.0
```

If a git repository does not have a .gemspec for the gem you attached it to, a version specifier MUST be provided. Bundler will use this version in the simple .gemspec it creates.

Git repositories support a number of additional options.

branch, tag, and ref

You MUST only specify at most one of these options. The default is :branch => "master". For example:

```
gem "rails", :git => "https://github.com/rails/rails.git",
:branch => "5-0-stable"
```

gem "rails", :git => "https://github.com/rails/rails.git", :tag => "v5.0.0"

gem "rails", :git => "https://github.com/rails/rails.git", :ref => "4aded"

submodules

For reference, a git submodule https://git-scm.com/book/en/v2/Git-Tools-Submodules lets you have another git repository within a subfolder of your reposi? tory. Specify :submodules => true to cause bundler to expand any submodules included in the git repository

If a git repository contains multiple .gemspecs, each .gemspec repre? sents a gem located at the same place in the file system as the .gem? spec.

```
|~rails                 [git root]
| |-rails.gemspec       [rails gem located here]
|~actionpack
| |-actionpack.gemspec  [actionpack gem located here]
|~activesupport
| |-activesupport.gemspec [activesupport gem located here]
|...
```

To install a gem located in a git repository, bundler changes to the directory containing the gemspec, runs gem build name.gemspec and then installs the resulting gem. The gem build command, which comes standard with Rubygems, evaluates the .gemspec in the context of the directory in which it is located.

GIT SOURCE

A custom git source can be defined via the git_source method. Provide the source?s name as an argument, and a block which receives a single argument and interpolates it into a string to return the full repo ad? dress:

```
git_source(:stash){ |repo_name| "https://stash.corp.acme.pl/#{repo_name}.git" }
gem ?rails?, :stash => ?forks/rails?
```

In addition, if you wish to choose a specific branch:

```
    gem "rails", :stash => "forks/rails", :branch => "branch_name"
```

GITHUB

```
  NOTE: This shorthand should be avoided until Bundler 2.0, since it cur?
  rently expands to an insecure git:// URL. This allows a man-in-the-mid?
  dle attacker to compromise your system.
  If the git repository you want to use is hosted on GitHub and  is  pub?
  lic,  you  can use the :github shorthand to specify the github username
  and repository name (without  the  trailing  ".git"),  separated  by  a
  slash.  If  both the username and repository name are the same, you can
  omit one.
    gem "rails", :github => "rails/rails"
    gem "rails", :github => "rails"
  Are both equivalent to
    gem "rails", :git => "git://github.com/rails/rails.git"
  Since the github method is a specialization of git_source, it accepts a
  :branch named argument.
  You can also directly pass a pull request URL:
    gem "rails", :github => "https://github.com/rails/rails/pull/43753"
  Which is equivalent to:
    gem "rails", :github => "rails/rails", branch: "refs/pull/43753/head"
```

GIST

```
  If the git repository you want to use is hosted as a Github Gist and is
  public, you can use the :gist shorthand to specify the gist  identifier
  (without the trailing ".git").
    gem "the_hatch", :gist => "4815162342"
  Is equivalent to:
    gem "the_hatch", :git => "https://gist.github.com/4815162342.git"
  Since  the  gist method is a specialization of git_source, it accepts a
  :branch named argument.
```

BITBUCKET

```
  If the git repository you want to use is hosted  on  Bitbucket  and  is
  public,  you  can use the :bitbucket shorthand to specify the bitbucket
  username and repository name (without the trailing  ".git"),  separated
```

by a slash. If both the username and repository name are the same, you can omit one.

```
gem "rails", :bitbucket => "rails/rails"
gem "rails", :bitbucket => "rails"
```

Are both equivalent to

```
gem "rails", :git => "https://rails@bitbucket.org/rails/rails.git"
```

Since the bitbucket method is a specialization of git_source, it ac? cepts a :branch named argument.

PATH

You can specify that a gem is located in a particular location on the file system. Relative paths are resolved relative to the directory con? taining the Gemfile.

Similar to the semantics of the :git option, the :path option requires that the directory in question either contains a .gemspec for the gem, or that you specify an explicit version that bundler should use.

Unlike :git, bundler does not compile C extensions for gems specified as paths.

```
gem "rails", :path => "vendor/rails"
```

If you would like to use multiple local gems directly from the filesys? tem, you can set a global path option to the path containing the gem?s files. This will automatically load gemspec files from subdirectories.

```
path ?components? do
  gem ?admin_ui?
  gem ?public_ui?
end
```

BLOCK FORM OF SOURCE, GIT, PATH, GROUP and PLATFORMS

The :source, :git, :path, :group, and :platforms options may be applied to a group of gems by using block form.

```
source "https://gems.example.com" do
  gem "some_internal_gem"
  gem "another_internal_gem"
end
git "https://github.com/rails/rails.git" do
```

```
  gem "activesupport"
  gem "actionpack"
end
platforms :ruby do
  gem "ruby-debug"
  gem "sqlite3"
end
group :development, :optional => true do
  gem "wirble"
  gem "faker"
end
```

In the case of the group block form the :optional option can be given to prevent a group from being installed unless listed in the --with option given to the bundle install command.

In the case of the git block form, the :ref, :branch, :tag, and :submodules options may be passed to the git method, and all gems in the block will inherit those options.

The presence of a source block in a Gemfile also makes that source available as a possible global source for any other gems which do not specify explicit sources. Thus, when defining source blocks, it is recommended that you also ensure all other gems in the Gemfile are using explicit sources, either via source blocks or :source directives on individual gems.

INSTALL_IF

The install_if method allows gems to be installed based on a proc or lambda. This is especially useful for optional gems that can only be used if certain software is installed or some other conditions are met.

```
install_if -> { RUBY_PLATFORM =~ /darwin/ } do
  gem "pasteboard"
end
```

GEMSPEC

The .gemspec http://guides.rubygems.org/specification-reference/ file is where you provide metadata about your gem to Rubygems. Some required

Gemspec attributes include the name, description, and homepage of your gem. This is also where you specify the dependencies your gem needs to run.

If you wish to use Bundler to help install dependencies for a gem while it is being developed, use the gemspec method to pull in the dependen? cies listed in the .gemspec file.

The gemspec method adds any runtime dependencies as gem requirements in the default group. It also adds development dependencies as gem re? quirements in the development group. Finally, it adds a gem requirement on your project (:path => ?.?). In conjunction with Bundler.setup, this allows you to require project files in your test code as you would if the project were installed as a gem; you need not manipulate the load path manually or require project files via relative paths.

The gemspec method supports optional :path, :glob, :name, and :develop? ment_group options, which control where bundler looks for the .gemspec, the glob it uses to look for the gemspec (defaults to: "{,,/*}.gem? spec"), what named .gemspec it uses (if more than one is present), and which group development dependencies are included in.

When a gemspec dependency encounters version conflicts during resolu? tion, the local version under development will always be selected -- even if there are remote versions that better match other requirements for the gemspec gem.

SOURCE PRIORITY

When attempting to locate a gem to satisfy a gem requirement, bundler uses the following priority order:

1. The source explicitly attached to the gem (using :source, :path, or :git)
2. For implicit gems (dependencies of explicit gems), any source, git, or path repository declared on the parent. This results in bundler prioritizing the ActiveSupport gem from the Rails git repository over ones from rubygems.org
3. The sources specified via global source lines, searching each source in your Gemfile from last added to first added.