



Linux Ubuntu 22.4.5 Manual Pages on command 'bundle-package2.7.1'

\$ man bundle-package2.7.1

BUNDLE-PACKAGE(1)

BUNDLE-PACKAGE(1)

NAME

bundle-package - Package your needed .gem files into your application

SYNOPSIS

bundle package

DESCRIPTION

Copy all of the .gem files needed to run the application into the vendor/cache directory. In the future, when running [bundle install(1)][bundle-install], use the gems in the cache in preference to the ones on rubygems.org.

GIT AND PATH GEMS

Since Bundler 1.2, the bundle package command can also package :git and :path dependencies besides .gem files. This needs to be explicitly enabled via the --all option. Once used, the --all option will be remembered.

SUPPORT FOR MULTIPLE PLATFORMS

When using gems that have different packages for different platforms, Bundler 1.8 and newer support caching of gems for other platforms where the Gemfile has been resolved (i.e. present in the lockfile) in vendor/cache. This needs to be enabled via the --all-platforms option. This setting will be remembered in your local bundler configuration.

REMOTE FETCHING

By default, if you run bundle install(1)(bundle-install.1.html) after running bundle package(1) bundle-package.1.html, bundler will still connect to rubygems.org to

check whether a platform-specific gem exists for any of the gems in vendor/cache.

For instance, consider this Gemfile(5):

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

If you run bundle package under C Ruby, bundler will retrieve the version of nokogiri for the "ruby" platform. If you deploy to JRuby and run bundle install, bundler is forced to check to see whether a "java" platformed nokogiri exists.

Even though the nokogiri gem for the Ruby platform is technically acceptable on JRuby, it has a C extension that does not run on JRuby. As a result, bundler will, by default, still connect to rubygems.org to check whether it has a version of one of your gems more specific to your platform.

This problem is also not limited to the "java" platform. A similar (common) problem can happen when developing on Windows and deploying to Linux, or even when developing on OSX and deploying to Linux.

If you know for sure that the gems packaged in vendor/cache are appropriate for the platform you are on, you can run bundle install --local to skip checking for more appropriate gems, and use the ones in vendor/cache.

One way to be sure that you have the right platformed versions of all your gems is to run bundle package on an identical machine and check in the gems. For instance, you can run bundle package on an identical staging box during your staging process, and check in the vendor/cache before deploying to production.

By default, bundle package(1) bundle-package.1.html fetches and also installs the gems to the default location. To package the dependencies to vendor/cache without installing them to the local install location, you can run bundle package --no-install.