# bundle cache

`bundle-cache` - Package your needed `.gem` files into your application

```
bundle cache [OPTIONS]
```

alias: `package`, `pack`

## 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)`, use the gems in the cache in preference to the ones on `rubygems.org`.

## Options

**`--all`**
Include all sources (including path and git).

**`--all-platforms`**
Include gems for all platforms present in the lockfile, not only the current one.

**`--cache-path=CACHE-PATH`**
Specify a different cache path than the default (vendor/cache).

**`--gemfile=GEMFILE`**
Use the specified gemfile instead of Gemfile.

**`--no-install`**
Don't install the gems, only update the cache.

**`--no-prune`**
Don't remove stale gems from the cache.

**`--path=PATH`**
Specify a different path than the system default ($BUNDLE_PATH or $GEM_HOME).

**`--quiet`**
Only output warnings and errors.

**`--frozen`**
Do not allow the Gemfile.lock to be updated after this bundle cache operation's install.

## Git And Path Gems

The `bundle cache` 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 supports 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)` after running bundle cache(1), 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 cache` 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 cache` on an identical machine and check in the gems. For instance, you can run `bundle cache` on an identical staging box during your staging process, and check in the `vendor/cache` before deploying to production.

By default, bundle cache(1) 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 cache --no-install`.

## History

In Bundler 2.1, `cache` took in the functionalities of `package` and now `package` and `pack` are aliases of `cache`.

**Edit this document on GitHub** if you caught an error or noticed something was missing.