NAME

bundle−update − Update your gems to the latest available versions

SYNOPSIS

bundle update **gems* [−−all] [−−group=NAME] [−−source=NAME] [−−local] [−−ruby] [−−bundler[=VERSION]] [−−full−index] [−−jobs=JOBS] [−−quiet] [−−patch|−−minor|−−major] [−−redownload] [−−strict] [−−conservative]

DESCRIPTION

Update the gems specified (all gems, if **−−all** flag is used), ignoring the previously installed gems specified in the **Gemfile.lock**. In general, you should use bundle install(1) *bundle−install.1.html* to install the same exact gems and versions across machines.

You would use **bundle update** to explicitly update the version of a gem.

OPTIONS

−−all Update all gems specified in Gemfile.

−−group=<name>, **−g=[<name>]**
Only update the gems in the specified group. For instance, you can update all gems in the development group with **bundle update −−group development**. You can also call **bundle update rails −−group test** to update the rails gem and all gems in the test group, for example.

−−source=<name>
The name of a **:git** or **:path** source used in the Gemfile(5). For instance, with a **:git** source of **http://github.com/rails/rails.git**, you would call **bundle update −−source rails**

−−local
Do not attempt to fetch gems remotely and use the gem cache instead.

−−ruby
Update the locked version of Ruby to the current version of Ruby.

−−bundler
Update the locked version of bundler to the invoked bundler version.

−−full−index
Fall back to using the single−file index of all gems.

−−jobs=[<number>], **−j[<number>]**
Specify the number of jobs to run in parallel. The default is **1**.

−−retry=[<number>]
Retry failed network or git requests for *number* times.

−−quiet
Only output warnings and errors.

−−redownload
Force downloading every gem.

−−patch
Prefer updating only to next patch version.

−−minor
Prefer updating only to next minor version.

−−major
Prefer updating to next major version (default).

−−strict
Do not allow any gem to be updated past latest **−−patch | −−minor | −−major**.

−−conservative
Use bundle install conservative update behavior and do not allow shared dependencies to be updated.

UPDATING ALL GEMS

If you run **bundle update −−all**, bundler will ignore any previously installed gems and resolve all dependencies again based on the latest versions of all gems available in the sources.

Consider the following Gemfile(5):

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

gem "rails", "3.0.0.rc"
gem "nokogiri"
```

When you run bundle install(1) *bundle−install.1.html* the first time, bundler will resolve all of the dependencies, all the way down, and install what you need:

```
Fetching gem metadata from https://rubygems.org/.........
Resolving dependencies...
Installing builder 2.1.2
Installing abstract 1.0.0
Installing rack 1.2.8
Using bundler 1.7.6
Installing rake 10.4.0
Installing polyglot 0.3.5
Installing mime−types 1.25.1
Installing i18n 0.4.2
Installing mini_portile 0.6.1
Installing tzinfo 0.3.42
Installing rack−mount 0.6.14
Installing rack−test 0.5.7
Installing treetop 1.4.15
Installing thor 0.14.6
Installing activesupport 3.0.0.rc
Installing erubis 2.6.6
Installing activemodel 3.0.0.rc
Installing arel 0.4.0
Installing mail 2.2.20
Installing activeresource 3.0.0.rc
Installing actionpack 3.0.0.rc
Installing activerecord 3.0.0.rc
Installing actionmailer 3.0.0.rc
Installing railties 3.0.0.rc
Installing rails 3.0.0.rc
Installing nokogiri 1.6.5

Bundle complete! 2 Gemfile dependencies, 26 gems total.
Use ‘bundle show [gemname]‘ to see where a bundled gem is installed.
```

As you can see, even though you have two gems in the Gemfile(5), your application needs 26 different gems in order to run. Bundler remembers the exact versions it installed in **Gemfile.lock**. The next time you run bundle install(1) *bundle−install.1.html*, bundler skips the dependency resolution and installs the same gems as it installed last time.

After checking in the **Gemfile.lock** into version control and cloning it on another machine, running bundle install(1) *bundle−install.1.html* will *still* install the gems that you installed last time. You don´t need to worry that a new release of **erubis** or **mail** changes the gems you use.

However, from time to time, you might want to update the gems you are using to the newest versions that still match the gems in your Gemfile(5).

To do this, run **bundle update −−all**, which will ignore the **Gemfile.lock**, and resolve all the dependencies again. Keep in mind that this process can result in a significantly different set of the 25 gems, based on the requirements of new gems that the gem authors released since the last time you ran **bundle update −−all**.

UPDATING A LIST OF GEMS

Sometimes, you want to update a single gem in the Gemfile(5), and leave the rest of the gems that you specified locked to the versions in the **Gemfile.lock**.

For instance, in the scenario above, imagine that **nokogiri** releases version **1.4.4**, and you want to update it *without* updating Rails and all of its dependencies. To do this, run **bundle update nokogiri**.

Bundler will update **nokogiri** and any of its dependencies, but leave alone Rails and its dependencies.

OVERLAPPING DEPENDENCIES

Sometimes, multiple gems declared in your Gemfile(5) are satisfied by the same second−level dependency. For instance, consider the case of **thin** and **rack−perftools−profiler**.

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

gem "thin"
gem "rack−perftools−profiler"
```

The **thin** gem depends on **rack >= 1.0**, while **rack−perftools−profiler** depends on **rack ˜> 1.0**. If you run bundle install, you get:

```
Fetching source index for https://rubygems.org/
Installing daemons (1.1.0)
Installing eventmachine (0.12.10) with native extensions
Installing open4 (1.0.1)
Installing perftools.rb (0.4.7) with native extensions
Installing rack (1.2.1)
Installing rack−perftools_profiler (0.0.2)
Installing thin (1.2.7) with native extensions
Using bundler (1.0.0.rc.3)
```

In this case, the two gems have their own set of dependencies, but they share **rack** in common. If you run **bundle update thin**, bundler will update **daemons**, **eventmachine** and **rack**, which are dependencies of **thin**, but not **open4** or **perftools.rb**, which are dependencies of **rack−perftools_profiler**. Note that **bundle update thin** will update **rack** even though it´s *also* a dependency of **rack−perftools_profiler**.

In short, by default, when you update a gem using **bundle update**, bundler will update all dependencies of that gem, including those that are also dependencies of another gem.

To prevent updating shared dependencies, prior to version 1.14 the only option was the **CONSERVATIVE UPDATING** behavior in bundle install(1) *bundle−install.1.html*:

In this scenario, updating the **thin** version manually in the Gemfile(5), and then running bundle install(1) *bundle−install.1.html* will only update **daemons** and **eventmachine**, but not **rack**. For more information, see the **CONSERVATIVE UPDATING** section of bundle install(1) *bundle−install.1.html*.

Starting with 1.14, specifying the **−−conservative** option will also prevent shared dependencies from being updated.

PATCH LEVEL OPTIONS

Version 1.14 introduced 4 patch−level options that will influence how gem versions are resolved. One of the following options can be used: **−−patch**, **−−minor** or **−−major**. **−−strict** can be added to further influence resolution.

−−patch
Prefer updating only to next patch version.

−−minor
Prefer updating only to next minor version.

−−major
Prefer updating to next major version (default).

−−strict
Do not allow any gem to be updated past latest **−−patch | −−minor | −−major**.

When Bundler is resolving what versions to use to satisfy declared requirements in the Gemfile or in parent gems, it looks up all available versions, filters out any versions that don´t satisfy the requirement, and then, by default, sorts them from newest to oldest, considering them in that order.

Providing one of the patch level options (e.g. **−−patch**) changes the sort order of the satisfying versions, causing Bundler to consider the latest **−−patch** or **−−minor** version available before other versions. Note that versions outside the stated patch level could still be resolved to if necessary to find a suitable dependency graph.

For example, if gem ´foo´ is locked at 1.0.2, with no gem requirement defined in the Gemfile, and versions 1.0.3, 1.0.4, 1.1.0, 1.1.1, 2.0.0 all exist, the default order of preference by default (**−−major**) will be "2.0.0, 1.1.1, 1.1.0, 1.0.4, 1.0.3, 1.0.2".

If the **−−patch** option is used, the order of preference will change to "1.0.4, 1.0.3, 1.0.2, 1.1.1, 1.1.0, 2.0.0".

If the **−−minor** option is used, the order of preference will change to "1.1.1, 1.1.0, 1.0.4, 1.0.3, 1.0.2, 2.0.0".

Combining the **−−strict** option with any of the patch level options will remove any versions beyond the scope of the patch level option, to ensure that no gem is updated that far.

To continue the previous example, if both **−−patch** and **−−strict** options are used, the available versions for resolution would be "1.0.4, 1.0.3, 1.0.2". If **−−minor** and **−−strict** are used, it would be "1.1.1, 1.1.0, 1.0.4, 1.0.3, 1.0.2".

Gem requirements as defined in the Gemfile will still be the first determining factor for what versions are available. If the gem requirement for **foo** in the Gemfile is ´˜> 1.0´, that will accomplish the same thing as providing the **−−minor** and **−−strict** options.

PATCH LEVEL EXAMPLES

Given the following gem specifications:

```
foo 1.4.3, requires: ˜> bar 2.0
foo 1.4.4, requires: ˜> bar 2.0
foo 1.4.5, requires: ˜> bar 2.1
foo 1.5.0, requires: ˜> bar 2.1
foo 1.5.1, requires: ˜> bar 3.0
bar with versions 2.0.3, 2.0.4, 2.1.0, 2.1.1, 3.0.0
```

Gemfile:

```
gem ´foo´
```

Gemfile.lock:

```
foo (1.4.3)
  bar (~> 2.0)
bar (2.0.3)
```

Cases:

```
#  Command Line                    Result
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
1  bundle update −−patch           ´foo 1.4.5´, ´bar 2.1.1´
2  bundle update −−patch foo       ´foo 1.4.5´, ´bar 2.1.1´
3  bundle update −−minor           ´foo 1.5.1´, ´bar 3.0.0´
4  bundle update −−minor −−strict  ´foo 1.5.0´, ´bar 2.1.1´
5  bundle update −−patch −−strict  ´foo 1.4.4´, ´bar 2.0.4´
```

In case 1, bar is upgraded to 2.1.1, a minor version increase, because the dependency from foo 1.4.5 required it.

In case 2, only foo is requested to be unlocked, but bar is also allowed to move because it´s not a declared dependency in the Gemfile.

In case 3, bar goes up a whole major release, because a minor increase is preferred now for foo, and when it goes to 1.5.1, it requires 3.0.0 of bar.

In case 4, foo is preferred up to a minor version, but 1.5.1 won´t work because the −−strict flag removes bar 3.0.0 from consideration since it´s a major increment.

In case 5, both foo and bar have any minor or major increments removed from consideration because of the −−strict flag, so the most they can move is up to 1.4.4 and 2.0.4.

RECOMMENDED WORKFLOW

In general, when working with an application managed with bundler, you should use the following workflow:

- After you create your Gemfile(5) for the first time, run

  ```
  $ bundle install
  ```

- Check the resulting **Gemfile.lock** into version control

  ```
  $ git add Gemfile.lock
  ```

- When checking out this repository on another development machine, run

  ```
  $ bundle install
  ```

- When checking out this repository on a deployment machine, run

```
$ bundle install −−deployment
```

- After changing the Gemfile(5) to reflect a new or update dependency, run

  ```
  $ bundle install
  ```

- Make sure to check the updated **Gemfile.lock** into version control

  ```
  $ git add Gemfile.lock
  ```

- If bundle install(1) *bundle−install.1.html* reports a conflict, manually update the specific gems that you changed in the Gemfile(5)

  ```
  $ bundle update rails thin
  ```

- If you want to update all the gems to the latest possible versions that still match the gems listed in the Gemfile(5), run

  ```
  $ bundle update −−all
  ```