conda-build Documentation

Release 3.21.7+0.gb98d7ec0.dirty

Anaconda, Inc.

Aug 01, 2022

CONTENTS

1 Installing and updating conda-build 3

2 Concepts 5

3 User guide 17

4 Resources 49

5 Release notes 115

Index 247

Conda-build contains commands and tools to use conda to build your own packages. It also provides helpful tools to constrain or pin versions in recipes. Building a conda package requires *installing conda-build* and creating a conda *recipe*. You then use the `conda build` command to build the conda package from the conda recipe.

You can build conda packages from a variety of source code projects, most notably Python. For help packing a Python project, see the Setuptools documentation.

OPTIONAL: If you are planning to upload your packages to Anaconda Cloud, you will need an Anaconda Cloud account and client.

CHAPTER

ONE

INSTALLING AND UPDATING CONDA-BUILD

To enable building conda packages:

- install conda
- install conda-build
- update conda and conda-build

1.1 Installing conda-build

To install conda-build, in your terminal window or an Anaconda Prompt, run:

```
conda install conda-build
```

1.2 Updating conda and conda-build

Keep your versions of conda and conda-build up to date to take advantage of bug fixes and new features.

To update conda and conda-build, in your terminal window or an Anaconda Prompt, run:

```
conda update conda
conda update conda-build
```

For release notes, see the conda-build GitHub page.

CHAPTER

TWO

CONCEPTS

2.1 Conda channels

The `conda-build` options `-c CHANNEL` or `--channel CHANNEL` configure additional channels to search for packages.

These are URLs searched in the order they are given (including file:// for local directories).

Then, the defaults or channels from `.condarc` are searched (unless `--override-channels` is given).

You can use 'defaults' to get the default packages for conda, and 'system' to get the system packages, which also takes `.condarc` into account.

You can also use any name and the `.condarc` `channel_alias` value will be prepended. The default `channel_alias` is http://conda.anaconda.org/.

The option `--override-channels` tells to not search default or `.condarc` channels. Requires the `--channel` or `-c` option.

2.1.1 Identical channel and package name problem

If the channel and package name are identical, it's possible to encounter a build problem if the short channel name is used.

Let's say your Anaconda.org username or an organization name is `example`. And suppose you created a package `example`, whose files' layout is similar to:

```
setup.py
conda/meta.yaml
example/
```

If your build depends on some other packages inside your channel, you will need to add `-c example`, however, the following code:

```
conda-build ./conda/ -c example
```

will fail with the following error message (the path will be different):

```
requests.exceptions.HTTPError: 404 Client Error: None for url:
file:///path/to/your/local/example/noarch/repodata.json
[...]
The remote server could not find the noarch directory for the requested channel with
url: file:///path/to/your/local/example/noarch/repodata.json
[...]
```

(continues on next page)

(continued from previous page)

```
As of conda 4.3, a valid channel must contain a `noarch/repodata.json` and
associated `noarch/repodata.json.bz2` file, even if `noarch/repodata.json`
is empty. please request that the channel administrator create
`noarch/repodata.json` and associated `noarch/repodata.json.bz2` files.
```

This happens because `conda-build` will consider the directory `./example/` in your project as a channel. This is by design due to conda's CI servers, where the build path can be long, complicated, and not predictable prior to build.

There are several ways to resolve this issue.

1. Use the url of the desired channel:

   ```
   conda-build ./conda/ -c https://conda.anaconda.org/example/
   ```

2. Run the build from inside the conda recipe directory:

   ```
   cd conda
   conda-build . -c example
   ```

3. Use the label specification workaround:

   ```
   conda-build ./conda/ -c example/label/main
   ```

   which technically is the same as `-c example`, since `main` is the default label, but now it won't by mistake find a channel `example/label/main` on the local filesystem.

2.2 Channels and generating an index

2.2.1 Channel layout

```
.
├── channeldata.json
├── linux-32
|   ├── repodata.json
|   └── package-0.0.0.tar.bz2
├── linux-64
|   ├── repodata.json
|   └── package-0.0.0.tar.bz2
├── win-64
|   ├── repodata.json
|   └── package-0.0.0.tar.bz2
├── win-32
|   ├── repodata.json
|   └── package-0.0.0.tar.bz2
├── osx-64
|   ├── repodata.json
|   └── package-0.0.0.tar.bz2
...
```

2.2.2 Parts of a channel

- Channeldata.json contains metadata about the channel, including:
  - What subdirs the channel contains.
  - What packages exist in the channel and what subdirs they are in.
- Subdirs are associated with platforms. For example, the linux-64 subdir contains packages for linux-64 systems.
- Repodata.json contains an index of the packages in a subdir. Each subdir will have it's own repodata.
- Channels have packages as tarballs under corresponding subdirs.

2.2.3 channeldata.json

```
{
  "channeldata_version": 1,
  "packages": {
    "super-fun-package": {
      "activate.d": false,
      "binary_prefix": false,
      "deactivate.d": false,
      "home": "https://github.com/Home/super-fun-package",
      "license": "BSD",
      "post_link": false,
      "pre_link": false,
      "pre_unlink": false,
      "reference_package": "win-64/super-fun-package-0.1.0-py37_0.tar.bz2",
      "run_exports": {},
      "subdirs": [
        "win-64"
      ],
      "summary": "A fun package! Open me up for rainbows",
      "text_prefix": false,
      "version": "0.1.0"
    },
    "subdirs": [
      "win-64",
      ...
    ]
}
```

2.2.4 repodata.json

```
{
  "packages": {
    "super-fun-package-0.1.0-py37_0.tar.bz2": {
      "build": "py37_0",
      "build_number": 0,
      "depends": [
        "some-depends"
      ],
```

(continues on next page)

(continued from previous page)

```
    "license": "BSD",
    "md5": "a75683f8d9f5b58c19a8ec5d0b7f796e",
    "name": "super-fun-package",
    "sha256": "1fe3c3f4250e51886838e8e0287e39029d601b9f493ea05c37a2630a9fe5810f",
    "size": 3832,
    "subdir": "win-64",
    "timestamp": 1530731681870,
    "version": "0.1.0"
  },
  ...
}
```

2.2.5 How an index is generated

For each subdir:

- Look at all the packages that exist in the subdir.
- Generate a list of packages to add/update/remove.
- Remove all packages that need to be removed.

For all packages that need to be added/updated:

- Extract the package to access metadata including full package name, mtime, size, and index.json.
- Aggregate package metadata to repodata collection.
- Apply repodata hotfixes (patches).
- Compute and save the reduced *current_index.json* index.

2.2.6 Example: Building a channel

To build a local channel and put a package in it, follow the directions below.

1. Make the channel structure.

```
$ mkdir local-channel
$ cd local-channel
$ mkdir linux-64 osx-64
```

2. Put your favorite package in the channel.

```
$ wget https://anaconda.org/anaconda/scipy/1.1.0/download/linux-64/scipy-1.
↪1.0-py37hfa4b5c9_1.tar.bz2 -P linux-64
$ wget https://anaconda.org/anaconda/scipy/1.1.0/download/osx-64/scipy-1.1.
↪0-py37hf5b7bf4_0.tar.bz2 -P osx-64
```

3. Run a conda index. This will generate both channeldata.json for the channel and repodata.json for the linux-64 and osx-64 subdirs, along with some other files.

```
$ conda index .
```

4. Check your work by searching the channel.

```
$ conda search -c file:/<path to>/local-channel scipy | grep local-channel
```

2.2.7 More details behind the scenes

Caching package metadata

Caching utilizes the existing repodata.json file if it exists. Indexing checks which files to update based on which files are new, removed, or changed since the last repodata.json was created. When a package is new or changed, its metadata is extracted and cached in the subdir to which the package belongs. The subfolder is the *.cache* folder. This folder has one file of interest: *stat.json*, which contains results from the *stat* command for each file. This is used for understanding when a file has changed and needs to be updated. In each of the other subfolders, the extracted metadata file for each package is saved as the original package name, plus a *.json* extension. Having these already extracted can save a lot of time in fully re-creating the index, should that be necessary.

An aside: one design goal of the *.conda* package format was to make indexing as fast as possible. To achieve this, the .conda format separates metadata from the actual package contents. Where the old *.tar.bz2* container required extracting the entire package to obtain the metadata, the new package format allows extraction of metadata without touching the package contents. This allows indexing speed to be independent of the package size. Large *.tar.bz2* packages can take a very long time to extract and index.

It is generally never necessary to manually alter the cache. To force an update/rescan of all cached packages, you can delete the .cache folder, or you can delete just the *.cache/stat.json* file. Ideally, you could remove only one package of interest from the cache, but that functionality does not currently exist.

Repodata patching

Package repodata is bootstrapped from the index.json file within packages. Unfortunately, that metadata is not always correct. Sometimes a version bound needs to be added retroactively. The process of altering repodata from the values derived from package index.json files is called "hotfixing." Hotfixing is tricky, as it has the potential to break environments that have worked, but it is also sometimes necessary to fix environments that are known not to work.

Repodata patches generated from a python script

On your own server, you're probably fine to run arbitrary python code that you have written to apply your patches. The advantage here is that the patches are generated on the fly every time the index is generated. That means that any new packages that have been added since the patch python file was last committed will be picked up and will have hotfixes applied to them where appropriate.

Anaconda applies hotfixes by providing a python file to *conda index* that has logic on how to alter metadata. Anaconda's repository of hotfixes is at https://github.com/AnacondaRecipes/repodata-hotfixes

Repodata patches applied from a JSON file

Unfortunately, you can't always run your python code directly - other people who host your patches may not allow you to run code. What you can do instead is package the patches as .json files. These will clobber the entries in the repodata.json when they are applied.

This is the approach that conda-forge has to take, for example. Their patch creation code is here: https://github.com/conda-forge/conda-forge-repodata-patches-feedstock/tree/master/recipe

What that code does is to download the current repodata.json, then runs their python logic to generate the patch JSON file. Those patches are placed into a location where Anaconda's mirroring tools will find them and apply them to conda-forge's repodata.json at mirroring time.

The downside here is that this JSON file is only as new as the last time that the repodata-patches feedstock last generated a package. Any new packages that have been added to the index in the meantime will not have any hotfixes applied to them, because the hotfix JSON file does not know about those files.

Trimming to "current" repodata

The number of packages available is always growing. That means conda is always having to do more and more work. To slow down this growth, in conda 4.7, we added the ability to have alternate repodata.json files that may represent a subset of the normal repodata.json. One in particular is *current_repodata.json*, which represents:

1. the latest version of each package
2. any earlier versions of dependencies needed to make the latest versions satisfiable

current_repodata.json also keeps only one file type: *.conda* where it is available, and *.tar.bz2* where only *.tar.bz2* is available.

For Anaconda's defaults "main" channel, the current_repodata.json file is approximately 1/7 the size of repodata.json. This makes downloading the repodata faster, and it also makes loading the repodata into its python representation faster.

For those interested in how this is achieved, please refer to the code at https://github.com/conda/conda-build/blob/90a6de55d8b9e36fc4a8c471b566d356e07436c7/conda_build/index.py#L695-L737

2.3 Conda-build recipes

- *Conda-build process*
- *Deep dive*
  - *Templates*
  - *Environments*
  - *Building*
  - *Prefix replacement*
  - *Testing*
  - *Output metadata*
- *More information*

To enable building conda packages, *install and update conda and conda-build*.

Building a conda package requires a recipe. A conda-build recipe is a flat directory that contains the following files:

- `meta.yaml`---A file that contains all the metadata in the recipe. Only `package/name` and `package/version` are required.
- `build.sh`---The script that installs the files for the package on macOS and Linux. It is executed using the `bash` command.
- `bld.bat`---The build script that installs the files for the package on Windows. It is executed using `cmd`.

- `run_test.[py,pl,sh,bat]`---An optional Python test file, a test script that runs automatically if it is part of the recipe.
- Optional patches that are applied to the source.
- Other resources that are not included in the source and cannot be generated by the build scripts. Examples are icon files, readme files and build notes.

Tip: When you use the *conda skeleton* command, the first 3 files---`meta.yaml`, `build.sh`, and `bld.bat`---are automatically generated for you.

2.3.1 Conda-build process

Conda-build performs the following steps:

1. Reads the metadata.
2. Downloads the source into a cache.
3. Extracts the source into the source directory.
4. Applies any patches.
5. Re-evaluates the metadata, if source is necessary to fill any metadata values.
6. Creates a build environment and then installs the build dependencies there.
7. Runs the build script. The current working directory is the source directory with environment variables set. The build script installs into the build environment.
8. Performs some necessary post-processing steps, such as shebang and rpath.
9. Creates a conda package containing all the files in the build environment that are new from step 5, along with the necessary conda package metadata.
10. Tests the new conda package if the recipe includes tests:
    1. Deletes the build environment and source directory to ensure that the new conda package does not inadvertantly depend on artifacts not included in the package.
    2. Creates a test environment with the package and its dependencies.
    3. Runs the test scripts.

The conda-recipes repo contains example recipes for many conda packages.

Caution: All recipe files, including `meta.yaml` and build scripts, are included in the final package archive that is distributed to users. Be careful not to put sensitive information such as passwords into recipes where it could be made public.

The `conda skeleton` command can help to make skeleton recipes for common repositories, such as PyPI.

2.3.2 Deep dive

Let's take a closer look at how conda-build uses a recipe to create a package.

Templates

When you build a conda package, conda-build renders the package by reading a template in the meta.yaml. See *Templating with Jinja*.

Templates are filled in using your conda-build config, which shows the matrix of things to build against. The `conda build config` determines how many builds it has to do. For example, defining a conda_build_config.yaml of the form and filling it defines a matrix of 4 packages to build:

```
foo:
  - 1.0
  - 2.0
bar:
  - 1.2.0
  - 1.4.0
```

After this, conda-build determines what the outputs will be. For example, if your `conda build config` indicates that you want 2 different versions of Python, conda-build will show you the rendering for each Python version.

Environments

To build the package, conda-build will make an environment for you and install all of the build and run dependencies in that environment. Conda-build will indicate where you can successfully build the package. The prefix will take the form:

```
<path to conda>/conda-bld/<package name and string>/h_env_placeholder...
```

Conda-forge downloads your package source and then builds the conda package in the context of the build environment. For example, you may direct it to download from a Git repo or pull down a tarball from another source. See the *Source section* for more information.

What conda-build puts into a package depends on what you put into the build, host, or run sections. See the *Requirements section* for more information. Conda-build will use this information to identify dependencies to link to and identify the run requirements for the package. This allows conda-build to understand what is needed to install the package.

Building

Once the content is downloaded, conda-build runs the build step. See the *Build section* for more information. The build step runs a script. It can be one that you provided. See the *Script* section for more information.

If you do not define the script section, then you can create a build.sh or a bld.bat file to be run.

Prefix replacement

When the build environment is created, it is in a placeholder prefix. When the package is all bundled up, the prefix is set to a dummy prefix. When conda is ready to install the package, it rewrites the dummy prefix with the correct one.

Testing

Once a package is built, conda-build will test it. To do this, it creates another environment and installs the conda package. The form of this prefix is:

```
<path to conda>/conda-bld/<package name + string>/_test_env_placeholder...
```

At this point, conda-build has all of the info from the meta.yaml about what its runtime dependencies are, so those dependencies are installed as well. This generates a test runner script with a reference to the testing meta.yaml that is created. See the *Test section* for more information. That file is run for testing.

Output metadata

After the package is built and tested, conda-build cleans up the environments created prior and outputs the metadata. The recipe for the package is also added in the output metadata. The metadata directory is on the top level of the tarball in the `info` directory. The metadata contains information about the dependencies of the package and a list of where all of the files in the package go when it is installed. Conda reads that metadata when it needs to install.

Running `conda install` causes conda to:

- reach out to the repo data containing the dependencies,
- guess the right dependencies,
- install a list of packages,
- unpack the tarball to look at the info,
- verify the file based on metadata in the package, and then
- go through each file in the package and put it in the right location.

2.3.3 More information

Review *Defining metadata (meta.yaml)* to see a breakdown of the components of a recipe, including:

- Package name.
- Package version.
- Descriptive metadata.
- Where to obtain source code.
- How to test the package.

2.4 Package naming conventions

To facilitate communication and documentation, conda observes the package naming conventions listed below.

2.4.1 Package name

The name of a package, without any reference to a particular version. Conda package names are normalized and they may contain only lowercase alpha characters, numeric digits, underscores, hyphens, or dots. In usage documentation, these are referred to by `package_name`.

2.4.2 Package version

A version number or string, often similar to `X.Y` or `X.Y.Z`, but it may take other forms as well.

2.4.3 Build string

An arbitrary string that identifies a particular build of a package for conda. It may contain suggestive mnemonics, but these are subject to change, and you should not rely on it or try to parse it for any specific information.

2.4.4 Canonical name

The package name, version, and build string joined together by hyphens---name-version-buildstring. In usage documentation, these are referred to by `canonical_name`.

2.4.5 Filename

Conda package filenames are canonical names, plus the suffix `.tar.bz2` or `.conda`.

The following figure compares a canonical name to a filename:

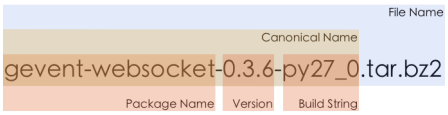

Fig. 1: Conda package naming

Conda supports both `.conda` and `.tar.bz2` package extensions. The `.conda` format is generally smaller and more efficient than `.tar.bz2` packages. Read our blog post about it to learn more.

The build string is created as the package is built. Things that contribute to it are the variants specified either by the command line or the configuration from the `conda_build_config.yaml`, and the build number in the recipe. If there are no variants, then the build string is the build number that is specified in the recipe.

2.4.6 Package specification

A package name together with a package version---which may be partial or absent---joined by an equal sign.

EXAMPLES:

- `python=2.7.3`
- `python=2.7`
- `python`

In usage documentation, these are referred to by `package_spec`.

2.5 What is a “package”?

- A package is anything you install using your package manager.
- A "conda package" is a compressed tarball that contains
  - the module to be installed
  - information on how to install the package
- You can use conda-build to build a conda package.

2.6 What about channels

- Channels contain packages.
- They conform to a standard structure and contain an index of available packages.
- An index of the available packages can be generated by running:

```
$ conda index <path to channel>
```

- conda is able to install from channels and uses the indexes in the channel to solve for requirements and dependencies.

2.7 Building Anaconda installers

- Anaconda(/Miniconda) installers are built with a modified version of constructor.
- The idea is to build an Anaconda metapackage and bundle it together with some other packages to build an Anaconda installer.

CHAPTER

THREE

USER GUIDE

Welcome to the conda-build user guide. Here you can find tutorials and recipes as well as information about environment variables and wheel files.

3.1 Getting started

Before starting the tutorials, consider reviewing *how to install and update conda-build* and *conda-build concepts*.

You may also find our *resources* collection helpful.

3.1.1 Prerequisites

Before starting the tutorials, you will need to install Miniconda or Anaconda, conda-build, and Git.

After you've installed Miniconda or Anaconda, you can use conda to install conda-build and Git.

3.1.2 Tutorial submissions

Have an idea for a tutorial? You can submit your suggestions to Anaconda by emailing us at documentation@anaconda.com.

To create your own tutorials, follow the *writing style guide* and *tutorial template*.

3.2 Tutorials

Before starting the tutorials, review the *Getting started* guide.

3.2.1 Building conda packages

- *Overview*
- *Who is this for?*
- *Before you start*
- *Toolkit*
- *Developing a build strategy*

- *Building with a Python version different from your Miniconda installation*
- *Automated testing*
- *Building a SEP package with conda and Python 2 or 3*
- *Building a GDAL package with conda and Python 2 or 3*

Overview

This tutorial describes how to use conda-build to create conda packages on Windows, macOS, and Linux using the examples of SEP and GDAL. Additional Windows-specific instructions are provided in the *Toolkit* section.

The final built packages from this tutorial are available on Anaconda Cloud:

- SEP.
- GDAL.

This tutorial also describes writing recipes. You can see the final SEP recipe and the GDAL recipe on GitHub in the conda-build documentation repository.

Who is this for?

This tutorial is for Windows, macOS, and Linux users who wish to build more complex conda packages. This tutorial will involve building scientific packages, which require compilers for several different Python versions.

Before you start

Before you start, make sure you have installed:

- Conda.
- *Conda-build*.
- Any compilers you want.

Toolkit

Microsoft Visual Studio

In the standard practices of the conda developers, conda packages for different versions of Python are each built with their own version of Visual Studio (VS):

- Python 2.7 packages with Visual Studio 2008
- Python 3.4 packages with VS 2010
- Python 3.5 packages with VS 2015, (default) 2017
- Python 3.6 packages with VS 2015, (default) 2017
- Python 3.7 packages with VS 2015, (default) 2017

Using these versions of VS to build packages for each of these versions of Python is also the practice used for the official python.org builds of Python. Currently VS 2008 and VS 2010 are available only through resellers, while VS 2015 and VS 2017 can be purchased online from Microsoft. Note there is also a community edition of VS 2015 and VS 2017 which may be used.

Alternatives to Microsoft Visual Studio

There are free alternatives available for each version of the VS compilers:

- Instead of VS 2008, it is often possible to substitute the free Microsoft Visual C++ Compiler for Python 2.7.
- Instead of VS 2010, it is often possible to substitute the free Microsoft Windows SDK for Windows 7 and .NET Framework 4.
- Make sure that you also install VS 2010 Service Pack 1 (SP1).
- Due to a bug in the VS 2010 SP1 installer, the compiler tools may be removed during installation of VS 2010 SP1. They can be restored as described in Microsoft Visual C++ 2010 Service Pack 1 Compiler Update for the Windows SDK 7.1.
- Visual Studio 2015 has a full-featured, free Community edition for academic research, open source projects, and certain other use cases.

The MS Visual C++ Compiler for Python 2.7 and the Microsoft Windows SDK for Windows 7 and .NET Framework 4 are both reasonably well tested. Conda-build is carefully tested to support these configurations, but there are known issues with the CMake build tool and these free VS 2008 and 2010 alternatives. In these cases, you should prefer the "NMake Makefile" generator, rather than a Visual Studio solution generator.

Windows versions

You can use any recent version of Windows. These examples were built on Windows 10.

Other tools

Some environments initially lack tools such as patch or Git that may be needed for some build workflows.

On Windows, these can be installed with conda:

```
conda install git m2-patch
```

On macOS and Linux, replace `m2-patch` with patch.

Developing a build strategy

Conda recipes are typically built with a trial-and-error method. Often the first attempt to build a package fails with compiler or linker errors, often caused by missing dependencies. The person writing the recipe then examines these errors and modifies the recipe to include the missing dependencies, usually as part of the `meta.yaml` file. Then the recipe writer attempts the build again and, after a few of these cycles of trial and error, the package builds successfully.

Building with a Python version different from your Miniconda installation

Miniconda2 and Miniconda3 can each build packages for either Python 2 or Python 3 simply by specifying the version you want. Miniconda2 includes only Python 2 and Miniconda3 includes only Python 3.

Installing only one makes it easier to keep track of the builds, but it is possible to have both installed on the same system at the same time. If you have both installed, use the `where` command on Windows, or `which` command on Linux to see which version comes first on PATH since this is the one you will be using:

```
where python
```

To build a package for a Python version other than the one in your Miniconda installation, use the `--python` option in the `conda-build` command.

EXAMPLE: To build a Python 3.5 package with Miniconda2:

```
conda-build recipeDirectory --python=3.5
```

Note: Replace `recipeDirectory` with the name and path of your recipe directory.

Automated testing

After the build, if the recipe directory contains a test file. This test file is named `run_test.bat` on Windows, `run_test.sh` on macOS or Linux, or `run_test.py` on any platform. The file runs to test the package and any errors are reported. After seeing "check the output," you can also test if this package was built by using the command:

```
$ conda build --test <path to package>.tar.bz2
```

Note: Use the *Test section of the meta.yaml file* to move data files from the recipe directory to the test directory when the test is run.

Building a SEP package with conda and Python 2 or 3

The SEP documentation states that SEP runs on Python 2 and 3, and it depends only on NumPy. Searching for SEP and PyPI shows that there is already a PyPI package for SEP.

Because a PyPI package for SEP already exists, the `conda skeleton` command can make a skeleton or outline of a conda recipe based on the PyPI package. Then the recipe outline can be completed manually and conda can build a conda package from the completed recipe.

Install Visual Studio

If you have not already done so, install the appropriate version of Visual Studio:

- For Python 3---Visual Studio 2017:
  1. Choose Custom install.
  2. Under Programming Languages, choose to install Visual C++.
- For Python 2---Visual Studio 2008:
  1. Choose Custom install.
  2. Choose to install X64 Compilers and Tools. Install Service Pack 1.

Make a conda skeleton recipe

1. Run the skeleton command:

```
conda skeleton pypi sep
```

   The `skeleton` command installs into a newly created directory called `sep`.

2. Go to the `sep` directory to view the files:

```
cd sep
```

   One skeleton file has been created: `meta.yaml`

Edit the skeleton files

For this package, `bld.bat` and `build.sh` need no changes. You need to edit the `meta.yaml` file to add the dependency on NumPy and add an optional test for the built package by importing it. For more information about what can be specified in `meta.yaml`, see *Defining metadata (meta.yaml)*.

1. In the requirements section of the `meta.yaml` file, add a line that adds NumPy as a requirement to build the package.
2. Add a second line to list NumPy as a requirement to run the package.
3. Set the NumPy version to the letters `x.x`.
4. Make sure the new line is aligned with `- python` on the line above it, so as to ensure proper yaml format.

EXAMPLE:

```
requirements:
  host:
    - python
    - numpy     x.x

  run:
    - python
    - numpy     x.x
```

Notice that there are two types of requirements, host and run. Host represents packages that need to be specific to the target platform when the target platform is not necessarily the same as the native build platform. Run represents the dependencies that should be installed when the package is installed.

Note: Using the letters `x.x` instead of a specific version such as `1.11` pins NumPy dynamically, so that the actual version of NumPy is taken from the build command. Currently, NumPy is the only package that can be pinned dynamically. Pinning is important for SEP because this package uses NumPy's C API through Cython. That API changes between NumPy versions, so it is important to use the same NumPy version at runtime that was used at build time.

OPTIONAL: Add a test for the built package

Adding this optional test will test the package at the end of the build by making sure that the Python statement `import sep` runs successfully:

1. Add `- sep`, checking to be sure that the indentation is consistent with the rest of the file.

   EXAMPLE:

```
test:
  # Python imports
  imports:
    - sep
```

Build the package

Build the package using the recipe you just created:

```
conda build sep
```

Check the output

1. Check the output to make sure that the build completed successfully. The output contains the location of the final package file and a command to upload the package to Anaconda Cloud. The output will look something like:

```
# Automatic uploading is disabled
# If you want to upload package(s) to anaconda.org later, type:
anaconda upload /Users/builder/miniconda3/conda-bld/osx-64/sep-1.0.3-np111py36_0.
↪tar.bz2
# To have conda build upload to anaconda.org automatically, use
# $ conda config --set anaconda_upload yes
anaconda_upload is not set.  Not uploading wheels: []
##################################################################################
Resource usage summary:
Total time: 0:00:56.4
CPU usage: sys=0:00:00.7, user=0:00:07.0
Maximum memory usage observed: 220.1M
Total disk usage observed (not including envs): 3.9K
##################################################################################
Source and build intermediates have been left in /Users/builder/miniconda3/conda-
↪bld.
There are currently 437 accumulated.
To remove them, you can run the ```conda build purge``` command
```

2. If there are any linker or compiler errors, modify the recipe and build again.

Building a GDAL package with conda and Python 2 or 3

This procedure describes how to build a package with Python 2 or Python 3. Follow the instructions for your preferred version.

To begin, install Anaconda or Miniconda and conda-build. If you are using a Windows machine, also use conda to install Git and the m2-patch.

```
conda install git
conda install m2-patch
```

Because GDAL includes C and C++, building it on Windows requires Visual Studio. This procedure describes how to build a package with Python 2 or Python 3. Follow the instructions for the version with which you want to build.

To build a GDAL package:

1. Install Visual Studio:
   - For Python 3, install Visual Studio 2017. Choose Custom install. Under Programming Languages, select workloads that come from Visual Studio so you choose the Desktop Development with C++ and Universal Platform C.
   - For Python 2, install Visual Studio 2008. Choose Custom install. Choose to install X64 Compilers and Tools. Install Visual Studio 2008 Service Pack 1.
2. Install Git. Because the GDAL package sources are retrieved from GitHub for the build, you must install Git:

   ```
   conda install git m2-patch conda-build
   ```

3. Get gdal-feedstock. For the purpose of this tutorial, we will be using a recipe from Anaconda:

   ```
   git clone https://github.com/AnacondaRecipes/gdal-feedstock.git
   ```

4. Use conda-build to build the gdal-feedstock:

   ```
   conda build gdal-feedstock
   ```

5. Check the output to make sure the build completed successfully. The output also contains the location of the final package file and a command to upload the package to Cloud. For this package in particular, there should be two packages outputted: libgdal and GDAL.
6. In case of any linker or compiler errors, modify the recipe and run it again.

Let's take a better look at what's happening inside the gdal-feedstock. In particular, what is happening in the `meta.yaml`.

The first interesting bit happens under `source` in the patches section:

```
patches:
  # BUILT_AS_DYNAMIC_LIB.
  - 0001-windowshdf5.patch
  # Use multiple cores on Windows.
  - 0002-multiprocessor.patch
  # disable 12 bit jpeg on Windows as we aren't using internal jpeg
  - 0003-disable_jpeg12.patch
```

This section says that when this package is being built on a Windows platform, apply the following patch files. Notice that the patch files are in the *patches* directory of the recipe. These patches will only be applied to Windows since the `# [win]` selector is applied to each of the patch entries. For more about selectors, see *Preprocessing selectors*.

In the requirements section, notice how there are both a build and host set of requirements. For this recipe, all the compilers required to build the package are listed in the build requirements. Normally, this section will list out packages required to build the package. GDAL requires CMake on Windows, as well as C compilers. Notice that the C compilers are pulled into the recipe using the syntax `{{ compiler('c') }}`. Since conda-build 3, conda-build defines a jinja2 function `compiler()` to specify compiler packages dynamically. So, using the `compiler('c')` function in a conda recipe will pull in the correct compiler for any build platform. For more information about compilers with conda-build see *compiler-tools*.

Also note that the compilers used by conda-build can be specified using a `conda_build_config.yaml`. For more information about how to do that, see *Using your customized compiler package with conda-build 3*.

Notice that this package has an `outputs` section. This section is a list of packages to output as a result of building this package. In this case, the packages libgdal and GDAL will be built. Similar to a normal recipe, the outputs can have build scripts, tests scripts and requirements specified. For more information on how outputs work, see the *Outputs section*.

Now, let's try to build GDAL against some build matrix. We will specify building against Python 3.7 and 3.5 using a conda-build config. Add the following to your `conda_build_config.yaml`:

```
python:
  - 3.7
  - 3.5
```

Now you can build GDAL using conda-build with the command:

```
conda build gdal-feedstock
```

Or explicitly set the location of the conda-build variant matrix:

```
conda build gdal-feedstock --variant-config-file conda_build_config.yaml
```

If you want to know more about build variants and `conda_build_config.yaml`, including how to specify a config file and what can go into it, take a look at *Creating conda-build variant config files*.

3.2.2 Building conda packages with conda skeleton

- *Overview*
- *Who is this for?*
- *Before you start*
- *Building a simple package with conda skeleton pypi*
- *Optional---Building for a different Python version*
- *Optional---Converting conda package for other platforms*
- *Optional---Uploading packages to Anaconda.org*
- *Troubleshooting a sample issue*
- *More information*

Overview

This tutorial describes how to quickly build a conda package for a Python module that is already available on PyPI.

In the first procedure, building a simple package, you build a package that can be installed in any conda environment of the same Python version as your root environment. The remaining optional procedures describe how to build packages for other Python versions and other architectures, as well as how to upload packages to your Anaconda.org account.

Note: You may consider using Docker to run the tutorial.

Who is this for?

This tutorial is for Windows, macOS, and Linux users who wish to build a conda package from a PyPI package. No prior knowledge of conda-build or conda recipes is required.

Before you start

Before you start, check the *Prerequisites*.

Building a simple package with conda skeleton pypi

The `conda skeleton` command picks up the PyPI package metadata and prepares the conda-build recipe. The final step is to build the package itself and install it into your conda environment.

It is easy to build a skeleton recipe for any Python package that is hosted on PyPI, the official third-party software repository for the Python programming language.

In this section you are going to use conda skeleton to generate a conda recipe, which informs conda-build about where the source files are located and how to build and install the package.

You'll be working with a package named Click that is hosted on PyPI. Click is a tool for exposing Python functions to create command line interfaces.

First, in your user home directory, run the `conda skeleton` command:

```
conda skeleton pypi click
```

The two arguments to `conda skeleton` are the hosting location, in this case `pypi`, and the name of the package.

This creates a directory named `click` and creates one skeleton file in that directory: `meta.yaml`. Many other files can be added there as necessary, such as `build.sh` and `bld.bat`, test scripts, or anything else you need to build your software. For simple, pure-Python recipes, these extra files are unnecessary and the build/script section in the `meta.yaml` is sufficient. Use the `ls` command on macOS or Linux or the `dir` command on Windows to verify that this file has been created. The `meta.yaml` file has been populated with information from the PyPI metadata and in many cases will not need to be edited.

Files in the folder with `meta.yaml` are collectively referred to as the "conda-build recipe":

- `meta.yaml`---Contains all the metadata in the recipe. Only the package name and package version sections are required---everything else is optional.
- `bld.bat`---Windows commands to build the package.
- `build.sh`---macOS and Linux commands to build the package.

Now that you have the conda-build recipe ready, you can use conda-build to create the package:

```
conda-build click
```

When conda-build is finished, it displays the exact path and filename of the conda package. See *Troubleshooting a sample issue* if the `conda-build` command fails.

Windows example file path:

```
C:\Users\jsmith\miniconda\conda-bld\win-64\click-7.0-py37_0.tar.bz2
```

macOS example file path:

```
/Users/jsmith/miniconda/conda-bld/osx-64/click-7.0-py37_0.tar.bz2
```

Linux example file path:

```
/home/jsmith/miniconda/conda-bld/linux-64/click-7.0-py37_0.tar.bz2
```

Note: Your path and filename will vary depending on your installation and operating system. Save the path and filename information for the next step.

Now you can install your newly built package in your conda environment by using the use-local flag:

```
conda install --use-local click
```

Notice that Click is coming from the local conda-build channel.

```
(click) 0561:~ jsmith$ conda list
# packages in environment at /Users/Jsmith/miniconda/envs/click:
# Name                    Version                   Build  Channel
ca-certificates           2019.1.23                     0
certifi                   2019.3.9                 py37_0
click                     7.0                      py37_0    local
```

Now verify that Click installed successfully:

```
conda list
```

Scroll through the list until you find Click.

At this point you now have a conda package for Click that can be installed in any conda environment of the same Python version as your root environment. The remaining optional sections show you how to make packages for other Python versions and other architectures and how to upload them to your Anaconda.org account.

Optional---Building for a different Python version

By default, conda-build creates packages for the version of Python installed in the root environment. To build packages for other versions of Python, you use the `--python` flag followed by a version. For example, to explicitly build a version of the Click package for Python 2.7, use:

```
conda-build --python 2.7 click
```

Notice that the file printed at the end of the `conda-build` output has changed to reflect the requested version of Python. `conda install` will look in the package directory for the file that matches your current Python version.

Windows example file path:

```
C:\Users\jsmith\Miniconda\conda-bld\win-64\click-7.0-py27_0.tar.bz2
```

macOS example file path:

```
/Users/jsmith/miniconda/conda-bld/osx-64/click-7.0-py27_0.tar.bz2
```

Linux example file path:

```
/home/jsmith/miniconda/conda-bld/linux-64/click-7.0-py27_0.tar.bz2
```

Note: Your path and filename will vary depending on your installation and operating system. Save the path and filename information for the next task.

Optional---Converting conda package for other platforms

Now that you have built a package for your current platform with conda-build, you can convert it for use on other platforms with the `conda convert` command. This works only for pure Python packages where there is no compiled code. Conda convert does nothing to change compiled code, it only adapts file paths to take advantage of the fact that Python scripts are mostly platform independent. Conda convert accepts a platform specifier from this and a platform specifier from this list:

- osx-64.
- linux-32.
- linux-64.
- win-32.
- win-64.
- all.

In the output directory, 1 folder will be created for each of the 1 or more platforms you chose and each folder will contain a .tar.bz2 package file for that platform.

Windows:

```
conda convert -f --platform all C:\Users\jsmith\miniconda\conda-bld\win-64\click-7.0-
↪py37_0.tar.bz2
-o outputdir\
```

macOS and Linux:

```
conda convert --platform all /home/jsmith/miniconda/conda-bld/linux-64/click-7.0-py37_0.
↪tar.bz2
-o outputdir/
```

Note: Change your path and filename to the exact path and filename you saved in *Optional---Building for a different Python version*.

To use these packages, you need to transfer them to other computers and place them in the correct `conda-bld/$ARCH` directory for the platform, where `$ARCH` can be `osx-64`, `linux-32`, `linux-64`, `win-32`, or `win-64`.

A simpler way is to upload all of the bz2 files to Anaconda.org as described in the next task.

If you find yourself needing to use `conda convert`, you might instead prefer to change your recipe to make your package a "noarch" package. Noarch packages run anywhere and do not require conda convert. Some of the ecosystem tools don't yet support noarch packages but, for the most part, noarch packages are a better way to go.

Optional---Uploading packages to Anaconda.org

Anaconda.org is a repository for public or private packages. Uploading to Anaconda.org allows you to easily install your package in any environment with just the `conda install` command, rather than manually copying or moving the tarball file from one location to another. You can choose to make your files public or private. For more information about Anaconda.org, see the Anaconda.org documentation.

1. Create a free Anaconda.org account and record your new Anaconda.org username and password.
2. Run `conda install anaconda-client` and enter your Anaconda.org username and password.
3. Log into your Anaconda.org account from your terminal with the command `anaconda login`.

Now you can upload the new local packages to Anaconda.org.

Windows:

```
anaconda upload C:\Users\jsmith\miniconda\conda-bld\win-64\click-7.0-py37_0.tar.bz2
```

macOS and Linux:

```
anaconda upload /home/jsmith/miniconda/conda-bld/linux-64/click-7.0-py37_0.tar.bz2
```

Note: Change your path and filename to the exact path and filename you saved in *Optional---Building for a different Python version*. Your path and filename will vary depending on your installation and operating system.

If you created packages for multiple versions of Python or used `conda convert` to make packages for each supported architecture, you must use the `anaconda upload` command to upload each one. It is considered best practice to create packages for Python versions 2.7, 3.4, and 3.5 along with all of the architectures.

Tip: If you want to always automatically upload a successful build to Anaconda.org, run: `conda config --set anaconda_upload yes`

You can log out of your Anaconda.org account with the command:

```
anaconda logout
```

Troubleshooting a sample issue

Conda-build may produce the error message "Build Package missing."

To explore this error:

1. Create a conda skeleton package for skyfield. The `conda skeleton` command is:

   ```
   conda skeleton pypi skyfield
   ```

   This command creates the skyfield conda-build recipe.

2. Run `conda-build skyfield` and observe that it fails with the following output:

```
Removing old build environment
Removing old work directory
BUILD START: skyfield-0.8-py35_0
Using Anaconda Cloud api site https://api.anaconda.org
Fetching package metadata: ......
Solving package specifications: .
Error:  Package missing in current osx-64 channels:
  - sgp4 >=1.4
```

In this example, the conda recipe requires `sgp4` for the skyfield package. The skyfield recipe was created by `conda skeleton`. This error means that conda could not find the sgp4 package and install it.

Since many PyPI packages depend on other PyPI packages to build or run, the solution is sometimes as simple as using `conda skeleton` to create a conda recipe for the missing package and then building it:

```
conda skeleton sgp4
conda build sgp4
```

You may also try using the `--recursive` flag with `conda skeleton`, but this makes conda recipes for all required packages, even those that are already available to conda install.

More information

For more options, see the full *conda skeleton command documentation*.

3.2.3 Building conda packages from scratch

- *Overview*
- *Who is this for?*
- *Before you start*
- *Editing the meta.yaml file*
- *Writing the build script files build.sh and bld.bat*
- *Building and installing*
- *Converting a package for use on all platforms*
- *Optional---Using PyPI as the source instead of GitHub*
- *Optional---Uploading new packages to Anaconda.org*
- *More information*

Overview

This tutorial describes how to build a conda package for Click by writing the required files in the conda-build recipe.

Who is this for?

This tutorial is for Windows, macOS, and Linux users who wish to generate a conda package by writing the necessary files. Prior knowledge of conda-build and conda recipes is helpful.

Before you start

- Check the *prerequisites*.
- You should have already completed *Building conda packages with conda skeleton*.

Editing the meta.yaml file

1. Make a new directory for this tutorial named `click`, and then change to the new directory:

```
mkdir click
cd click
```

2. To create a new `meta.yaml` file, open your favorite editor. Create a new text file and insert the information shown below. A blank sample `meta.yaml` follows the table to make it easier to match up the information.

Note: To allow correct sorting and comparison, specify `version` as a string.

name	click
version	"7.0" (or latest from https://github.com/pallets/click/releases)
git_rev	6.7 (or latest from https://github.com/pallets/click/releases)
git_url	https://github.com/pallets/click.git
imports	click
home	https://github.com/pallets/click
license	BSD

```
package:
  name:
  version:

source:
  git_rev:
  git_url:

requirements:
  build:
    - python
    - setuptools

  run:
```

(continues on next page)

(continued from previous page)

```
    - python

test:
  imports:
    -

about:
  home:
```

3. Save the file in the same `click` directory as `meta.yaml`. It should match `this meta.yaml file`.

Writing the build script files build.sh and bld.bat

Besides `meta.yaml`, 2 files are required for a build:

- `build.sh`---Shell script for macOS and Linux.
- `bld.bat`---Batch file for Windows.

These 2 build files contain all the variables, such as for 32-bit or 64-bit architecture---the ARCH variable---and the build environment prefix---PREFIX. The 2 files `build.sh` and `bld.bat` must be in the same directory as your `meta.yaml` file.

This tutorial describes how to make both `build.sh` and `bld.bat` so that other users can build the appropriate package for their architecture.

1. Open a text editor and create a new file named `bld.bat`. Type the text exactly as shown:

```
"%PYTHON%" setup.py install
if errorlevel 1 exit 1
```

> **Note:** In `bld.bat`, the best practice is to to add `if errorlevel 1 exit 1` after every command so that if the command fails, the build fails.

2. Save this new file `bld.bat` to the same directory where you put your `meta.yaml` file.
3. Open a text editor and create a new file named `build.sh`. Enter the text exactly as shown:

```
$PYTHON setup.py install     # Python command to install the script.
```

4. Save your new `build.sh` file to the same directory where you put the `meta.yaml` file.

You can run `build.sh` with `bash -x -e`. The `-x` makes it echo each command that is run, and the `-e` makes it exit whenever a command in the script returns nonzero exit status. If you need to revert this in the script, use the `set` command in `build.sh`.

Building and installing

Now that you have your 3 new build files ready, you are ready to create your new package with conda-build and install the package on your local computer.

1. Run conda-build:

```
conda-build click
```

   If you are already in the click folder, you can type `conda build .`.

   When conda-build is finished, it displays the package filename and location. In this case the file is saved to:

```
~/anaconda/conda-bld/linux-64/click-7.0-py37_0.tar.bz2
```

   Note: Save this path and file information for the next task. The exact path and filename vary depending on your operating system and whether you are using Anaconda or Miniconda. The `conda-build` command tells you the exact path and filename.

2. Install your newly built program on your local computer by using the `use-local` flag:

```
conda install --use-local click
```

   If there are no error messages, Click installed successfully.

   Note: Explicitly installing a local package bypasses the dependency resolver, as such the package's `run` dependencies will not be evaluated. See *conda install --help* or the install command reference page for more info.

Converting a package for use on all platforms

Now that you have built a package for your current platform with conda-build, you can convert it for use on other platforms by using the 2 build files, `build.sh` and `bld.bat`.

Use the `conda convert` command with a platform specifier from the list:

- `osx-64`.
- `linux-32`.
- `linux-64`.
- `win-32`.
- `win-64`.
- `all`.

EXAMPLE: Using the platform specifier `all`:

```
conda convert --platform all ~/anaconda/conda-bld/linux-64/click-7.0-py37_0.tar.bz2 -o␣
↪outputdir/
```

Note: Change your path and filename to the path and filename you saved in *Building and installing*.

Optional---Using PyPI as the source instead of GitHub

You can use PyPI or another repository instead of GitHub. There is little difference to conda-build between building from Git versus building from a tarball on a repository like PyPI. Because the same source is hosted on PyPI and GitHub, you can easily find a script on PyPI instead of GitHub.

Replace this `source` section:

```
git_rev: v0.6.7
git_url: https://github.com/pallets/click.git
```

With the following:

```
url: https://files.pythonhosted.org/packages/f8/5c/
↪f60e9d8a1e77005f664b76ff8aeaee5bc05d0a91798afd7f53fc998dbc47/Click-7.0.tar.gz
sha256: 5b94b49521f6456670fdb30cd82a4eca9412788a93fa6dd6df72c94d5a8ff2d7
```

Note: The `url` and `sha256` are found on the PyPI Click page.

Optional---Uploading new packages to Anaconda.org

After converting your files for use on other platforms, you may choose to upload your files to Anaconda.org, formerly known as binstar.org. It only takes a minute to do if you have a free Anaconda.org account.

1. If you have not done so already, open a free Anaconda.org account and record your new user name and password.
2. Run the command `conda install anaconda-client`, and then enter your Anaconda.org username and password.
3. Log into your Anaconda.org account with the command:

   ```
   anaconda login
   ```

4. Upload your package to Anaconda.org:

   ```
   anaconda upload ~/miniconda/conda-bld/linux-64/click-7.0-py37_0.tar.bz2
   ```

   Note: Change your path and filename to the path and filename you saved in *Building and installing*.

   Tip: To save time, you can set conda to always upload a successful build to Anaconda.org with the command: `conda config --set anaconda_upload yes`.

More information

- For more information about all the possible values that can go into the `meta.yaml` file, see *Defining metadata (meta.yaml)*.

3.2.4 Building R packages with skeleton CRAN

- *Overview*
- *Who is this for?*
- *Before you start*
- *Building a simple package with conda skeleton CRAN*
- *Building an R package with dependencies*
- *Optional---Building for a different R version*
- *Optional---Uploading packages to Anaconda.org*
- *More information*

Overview

This tutorial describes how to quickly build an R-language package on macOS for an R module that is already available on CRAN.

You will build a simple package that can be installed in any conda environment of the same R version as your root environment. The tutorial will also tell you how to build the dependencies that may arise while building the package.

Who is this for?

This tutorial is for macOS users who wish to build an R-language package from CRAN. No prior knowledge of conda-build or conda recipes is required.

Before you start

Before you start, check the *prerequisites*.

Building a simple package with conda skeleton CRAN

The conda skeleton command picks up the CRAN package metadata and prepares the conda-build recipe. The final step is to build the package itself and install it into your conda environment.

It is easy to build a skeleton recipe for any R package that is hosted on CRAN. In this section you are going to use conda skeleton to generate a conda recipe, which informs conda-build about where the source files are located and how to build and install the package.

You'll be working with a package that is hosted on CRAN named fansi, a tool that accounts for the effects of ANSI text formatting control sequences.

First, in your user home directory, run the conda skeleton command:

```
conda skeleton cran fansi
```

The two arguments to `conda skeleton` are the type of hosting location, in this case `cran`, and the name of the package.

This creates a directory named `r-fansi` and creates 1 skeleton file in that directory: `meta.yaml`. Many other files can be added there as necessary, such as `build.sh` and `bld.bat`, test scripts, or anything else you need to build your software. Use the `ls` command to verify that this file has been created. The `meta.yaml` file has been populated with information from the CRAN metadata and in many cases will not need to be edited.

Files in the folder with `meta.yaml` are collectively referred to as the "conda-build recipe":

- `meta.yaml`---Contains all the metadata in the recipe. The package name and package version sections are required---everything else is optional.
- `bld.bat`---Windows commands to build the package.
- `build.sh`---macOS and Linux commands to build the package.

Now that you have the conda-build recipe ready, you can use conda-build to create the package:

```
conda-build r-fansi
```

When conda-build is finished, it displays the exact path and filename of the conda package. See *Troubleshooting a sample issue* if the `conda-build` command fails. If you receive an error with SDK on macOS, review our *resources for macOS and SDK*.

Example file path:

```
/Users/jsmith/anaconda3/conda-bld/osx-64/r-fansi-0.4.0-r353h46e59ec_0.tar.bz2
```

Note: Your path and filename will vary depending on your installation and operating system. Save the path and filename information for the next step.

Now you can install your newly built package in your conda environment by using the use-local flag:

```
conda install --use-local r-fansi
```

Now verify that fansi installed successfully:

```
conda list
```

Scroll through the list until you find `r-fansi`.

Notice that fansi is coming from the local conda-build channel.

```
(base) 0561:~ jsmith$ conda list
# packages in environment at /Users/Jsmith/anaconda3:
# Name                    Version                   Build  Channel
qtpy                      1.5.0                     py37_0
r-base                    3.5.1                h539fb6c_1
r-fansi                   0.4.0           r353h46e59ec_0    local
```

The version of R will be what you have in your base environment.

See *Optional---Building for a different R version* to set your own R version.

At this point you now have a conda package for fansi that can be installed in any conda environment of its R version.

Building an R package with dependencies

The fansi package was a simple one that didn't have dependencies. To build an R package with dependencies, let's look at the example of janitor. Janitor is a package hosted on CRAN that is used for examining and cleaning up data.

To begin building it, type:

```
conda skeleton cran janitor
```

This creates a directory named `r-janitor` and creates one skeleton file in that directory: `meta.yaml`. Many other files can be added there as necessary, such as `build.sh` and `bld.bat`, test scripts, or anything else you need to build your software. Use the `ls` command to verify that this file has been created. The `meta.yaml` file has been populated with information from the CRAN metadata and in many cases will not need to be edited.

Now that you have the conda-build recipe ready, you can use conda-build to create the package:

```
conda-build r-janitor
```

What may happen at this point is that you will have dependencies of this package that do not exist as conda packages yet. They need to be turned into conda packages. Use conda skeleton to recursively build out recipes for the packages that it depends on:

```
conda skeleton cran janitor --recursive
```

You can manually build each package individually by typing:

```
conda-build package-name
```

Note: Replace "package-name" with the name of each package.

Once all of the package dependencies are resolved, you can build the R package by using:

```
conda-build .
```

Now you can install your newly built package in your conda environment by using the use-local flag:

```
conda install --use-local r-janitor
```

The remaining optional sections show you how to make R packages for other R versions and other architectures and how to upload them to your Anaconda.org account.

Optional---Building for a different R version

By default, conda-build creates packages for the version of R installed in the root environment. To build packages for other versions of R, you use the `--R` flag followed by a version.

For example, to explicitly build a version of the fansi package for R 3.6.1, use:

```
conda-build --R 3.6.1 r-fansi
```

Notice that the file printed at the end of the conda-build output has changed to reflect the requested version of R. Conda install will look in the package directory for the file that matches your current R version.

Example file path:

```
/Users/jsmith/anaconda3/conda-bld/osx-64/r-fansi-0.4.0-r353h46e59ec_0.tar.bz2
```

Note: Your path and filename will vary depending on your installation and operating system. Save the path and filename information for the next task.

Optional---Uploading packages to Anaconda.org

Anaconda.org is a repository for public or private packages. Uploading to Anaconda.org allows you to easily install your package in any environment with just the `conda install` command, rather than manually copying or moving the tarball file from one location to another. You can choose to make your files public or private.

For more information about Anaconda.org, see the Anaconda.org documentation.

1. Create a free Anaconda.org account and record your new Anaconda.org username and password.
2. Run `conda install anaconda-client` and enter your Anaconda.org username and password.
3. Log into your Anaconda.org account from your terminal with the command `anaconda login`.

Now you can upload the new local packages to Anaconda.org.

```
anaconda upload /Users/jsmith/anaconda3/conda-bld/osx-64/r-fansi-0.4.0-r353h46e59ec_0.
↪tar.bz2
```

Note: Change your path and filename to the exact path and filename you saved in *Optional---Building for a different R version*. Your path and filename will vary depending on your installation and operating system. If you created packages for multiple versions of R, you must use the `anaconda upload` command to upload each one.

Tip: If you want to always automatically upload a successful build to Anaconda.org, run: `conda config --set anaconda_upload yes`

You can log out of your Anaconda.org account with the command:

```
anaconda logout
```

More information

For more options, see the full *conda skeleton command documentation*.

3.3 Recipes

Review *recipe concepts* for more information about conda-build recipes.

3.3.1 Building a package without a recipe (bdist_conda)

- *Setup options*
  - *Build number*
  - *Build string*
  - *Import tests*
  - *Command line tests*
  - *Binary files relocatable*
  - *Preserve egg directory*
- *Command line options*
  - *Build number*
- *Notes*

You can use conda-build to build packages for Python to install rather than conda by using `setup.py bdist_conda`. This is a quick way to build packages without using a recipe, but it has limitations. The script is limited to the Python version used in the build and it is not as reproducible as using a recipe. We recommend using a recipe with conda-build.

Note: If you use Setuptools, you must first import Setuptools and then import `distutils.command.bdist_conda`, because Setuptools monkey patches `distutils.dist.Distribution`.

EXAMPLE: A minimal `setup.py` file using the setup options `name` and `version`:

```
 from distutils.core import setup, Extension
 import distutils.command.bdist_conda

setup(
    name="foo",
    version="1.0",
    distclass=distutils.command.bdist_conda.CondaDistribution,
    conda_buildnum=1,
)
```

Setup options

You can pass the following options to `setup()`. You must include `distclass=distutils.command.bdist_conda.CondaDistribution)`.

Build number

The number of the build. Can be overridden on the command line with the `--buildnum` flag. Defaults to `0`.

```
conda_buildnum=1
```

Build string

The build string. Default is generated automatically from the Python version, NumPy version---if relevant---and the build number, such as `py34_0`.

```
conda_buildstr=py34_0
```

Import tests

Whether to automatically run import tests. The default is `True`, which runs import tests for all the modules in `packages`. Also allowed are `False`, which runs no tests, or a list of module names to be tested on import.

```
conda_import_tests=False
```

Command line tests

Command line tests to run. Default is `True`, which runs `command --help` for each command in the console_scripts and gui_scripts entry_points. Also allowed are `False`, which does not run any command tests, or a list of command tests to run.

```
conda_command_tests=False
```

Binary files relocatable

Whether binary files should be made relocatable, using install_name_tool on macOS or patchelf on Linux. The default is `True`.

```
conda_binary_relocation=False
```

For more information, see *Making packages relocatable*.

Preserve egg directory

Whether to preserve the egg directory as installed by Setuptools. The default is `True` if the package depends on Setuptools or has Setuptools entry_points other than console_scripts and gui_scripts.

```
conda_preserve_egg_dir=False
```

Command line options

Build number

Set the build number. Defaults to the conda_buildnum passed to `setup()` or `0`. Overrides any conda_buildnum passed to `setup()`.

```
--buildnum=1
```

Notes

- You must install `bdist_conda` into a root conda environment, as it imports `conda` and `conda_build`. It is included as part of the `conda-build` package.
- All metadata is gathered from the standard metadata from the `setup()` function. Metadata that are not directly supported by `setup()` can be added using one of the options specified above.
- By default, import tests are run for each subpackage specified by packages, and command line tests `command --help` are run for each `setuptools entry_points` command. This is done to ensure that the package is built correctly. You can disable or change these using the `conda_import_tests` and `conda_command_tests` options specified above.
- The Python version used in the build must be the same as where conda is installed, as `bdist_conda` uses `conda-build`.
- `bdist_conda` uses the metadata provided to the `setup()` function.
- If you want to pass any `bdist_conda` specific options to `setup()`, in `setup()` you must set `distclass=distutils.command.bdist_conda.CondaDistribution`.

3.3.2 Sample recipes

Conda offers you the flexibility of being able to build things that are not Python related. The first 2 sample recipes, `boost` and `libtiff`, are examples of non-Python libraries, meaning they do not require Python to run or build.

- boost is an example of a popular programming library and illustrates the use of selectors in a recipe.
- libtiff is another example of a compiled library, which shows how conda can apply patches to source directories before building the package.
- msgpack, blosc, and cytoolz are examples of Python libraries with extensions.
- toolz, sympy, six, and gensim are examples of Python-only libraries.

`gensim` works on Python 2, and all of the others work on both Python 2 and Python 3.

3.3.3 Debugging conda recipes

Recipes are something that you'll rarely get exactly right on the first try. Something about the build will be wrong, and the build will break. Maybe you only notice a problem during tests, but you need more info than you got from the tests running in conda-build. Conda-build 3.17.0 added the subcommand, `conda debug`, that is designed to facilitate the recipe debugging process.

Fundamentally, debugging is a process of getting into or recreating the environment and set of shell environment variables that conda-build creates during its build or test processes. This has been possible for a very long time---you could observe the build output, figure out where the files from your build were placed, navigate there, and finally, activate the appropriate environment(s). Then you might also need to set some environment variables manually.

What `conda debug` does is to create environments for you and provide you with a single command line that you can copy/paste to enter a debugging environment.

Usage

The `conda debug` command accepts 1 of 2 kinds of inputs: a recipe folder or a path to a built package.

If a path to a recipe folder is provided, `conda debug` creates the build and host environments. It provisions any source code that your recipe specifies. It leaves the build-time scripts in the work folder for you. When complete, `conda debug` prints something like this:

```
##################################################################################
Build and/or host environments created for debugging.  To enter a debugging environment:

cd /Users/UserName/miniconda3/conda-bld/debug_1542385789430/work && source /Users/
↪UserName/miniconda3/conda-bld/debug_1542385789430/work/build_env_setup.sh

To run your build, you might want to start with running the conda_build.sh file.
##################################################################################
```

If a path to a built package is provided, `conda debug` creates the test environment. It prepares any test files that the recipe specified. When complete, `conda debug` prints something like this:

```
##################################################################################
Test environment created for debugging.  To enter a debugging environment:

cd /Users/UserName/miniconda3/conda-bld/conda-build_1542302975704/work && source /Users/
↪UserName/miniconda3/conda-bld/conda-build_1542302975704/work/build_env_setup.sh

To run your tests, you might want to start with running the conda_test_runner.sh file.
##################################################################################
```

Next steps

Given the output above, you can now enter an environment to start debugging. Copy paste from your terminal and go:

```
cd /Users/UserName/miniconda3/conda-bld/debug_1542385789430/work && source /Users/
↪UserName/miniconda3/conda-bld/debug_1542385789430/work/build_env_setup.sh
```

This is where you'll hopefully know what build commands you want to run to help you debug. Every build is different so your experience will vary. However, if you have no idea at all, you could probably start by running the appropriate

build or test script, as mentioned in the output. If you do this, remember that these scripts might be written to exit on error, which may close your shell session. It may be wise to only run these scripts in an explicit subshell:

```
bash conda_build.sh
bash conda_test_runner.sh
```

Complications with multiple outputs

Multiple outputs effectively give the recipe many build phases to consider. The `--output-id` argument is the mechanism to specify which of these should be used to create the debug envs and scripts. The `--output-id` argument accepts an fnmatch pattern. You can match any part of the output filenames. This really only works for conda packages, not other output types, such as wheels, because conda-build can't currently predict their filenames without actually carrying out a build.

For example, our NumPy recipe has multiple outputs. If we wanted to debug the NumPy-base output, we would specify it with a command like:

```
conda debug numpy-feedstock --output-id="numpy-base*"
```

If you have a matrix build, you may need to be more specific:

```
Specified --output-id matches more than one output (['/Users/msarahan/miniconda3/conda-
↪bld/debug_1542387301945/osx-64/numpy-base-1.14.6-py27h1a60bec_4.tar.bz2', '/Users/
↪msarahan/miniconda3/conda-bld/debug_1542387301945/osx-64/numpy-base-1.14.6-
↪py27h8a80b8c_4.tar.bz2', '/Users/msarahan/miniconda3/conda-bld/debug_1542387301945/osx-
↪64/numpy-base-1.14.6-py36h1a60bec_4.tar.bz2', '/Users/msarahan/miniconda3/conda-bld/
↪debug_1542387301945/osx-64/numpy-base-1.14.6-py36h8a80b8c_4.tar.bz2', '/Users/msarahan/
↪miniconda3/conda-bld/debug_1542387301945/osx-64/numpy-base-1.14.6-py37h1a60bec_4.tar.
↪bz2', '/Users/msarahan/miniconda3/conda-bld/debug_1542387301945/osx-64/numpy-base-1.14.
↪6-py37h8a80b8c_4.tar.bz2']).  Please refine your output id so that only a single␣
↪output is found.
```

You could either reduce your matrix by changing your `conda_build_config.yaml`, or making a simpler one and passing it on the CLI, or by using the CLI to reduce it.

```
conda debug numpy-feedstock --output-id="numpy-base*" --python=3.6 --variants="{blas_
↪impl: openblas}"
```

```
Specified --output-id matches more than one output (['/Users/UserName/miniconda3/conda-
↪bld/debug_1542387443190/osx-64/numpy-base-1.14.6-py36h28eea48_4.tar.bz2', '/Users/
↪UserName/miniconda3/conda-bld/debug_1542387443190/osx-64/numpy-base-1.14.6-
↪py36ha711998_4.tar.bz2']).  Please refine your output id so that only a single output␣
↪is found.
```

However, this is still not enough as our matrix includes two BLAS implementations, MKL and OpenBLAS.

Further reduction:

```
conda debug numpy-feedstock --output-id="numpy-base*" --python=3.6 --variants="{blas_
↪impl: 'openblas'}"
```

Cleanup

Debugging folders are named in a way that the `conda build purge` command will find and clean up. If you use the -p/--path CLI argument, conda-build will not detect these and you'll need to manually clean up yourself. `conda build purge-all` will also remove previously built packages.

Quirks

You can specify where you want the root of your debugging stuff to go with the -p/--path CLI argument. The way this works is that conda-build treats that as its "croot" where packages get cached as necessary, as well as potentially indexed. When using the --path argument, you may see folders like "osx-64" or other platform subdirs in the path you specify. It is safe to remove them or ignore them.

3.4 Environment variables

- *Dynamic behavior based on state of build process*
- *Environment variables set during the build process*
- *Git environment variables*
- *Mercurial environment variables*
- *Inherited environment variables*
- *Environment variables that affect the build process*
- *Environment variables that affect the test process*

3.4.1 Dynamic behavior based on state of build process

There are times when you may want to process a single file in different ways at more than 1 step in the render-build-test flow of conda-build. Conda-build sets the CONDA_BUILD_STATE environment variable during each of these phases. The possible values are:

- RENDER---Set during evaluation of the `meta.yaml` file.
- BUILD---Set during processing of the `bld.bat` or `build.sh` script files.
- TEST---Set during the running of any `run_test` scripts, which also includes any commands defined in `meta.yaml` in the `test/commands` section.

The CONDA_BUILD_STATE variable is undefined outside of these locations.

3.4.2 Environment variables set during the build process

During the build process, the following environment variables are set, on Windows with `bld.bat` and on macOS and Linux with `build.sh`. By default, these are the only variables available to your build script. Unless otherwise noted, no variables are inherited from the shell environment in which you invoke `conda-build`. To override this behavior, see *Inherited environment variables*.

ARCH	Either `32` or `64`, to specify whether the build is 32-bit or 64-bit. The value depends on the ARCH environment variable and defaults to the architecture the interpreter running conda was compiled with.
CMAKE_GENERATOR	The CMake generator string for the current build environment. On Linux systems, this is always `Unix Makefiles`. On Windows, it is generated according to the Visual Studio version activated at build time, for example, `Visual Studio 9 2008 Win64`.
CONDA_BUILD=1	Always set.
CPU_COUNT	The number of CPUs on the system, as reported by `multiprocessing.cpu_count()`.
SHLIB_EXT	The shared library extension.
DIRTY	Set to 1 if the `--dirty` flag is passed to the `conda-build` command. May be used to skip parts of a build script conditionally for faster iteration time when developing recipes. For example, downloads, extraction and other things that need not be repeated.
HTTP_PROXY	Inherited from your shell environment.
HTTPS_PROXY	Inherited from your shell environment.
LANG	Inherited from your shell environment.
MAKEFLAGS	Inherited from your shell environment. May be used to set additional arguments to make, such as `-j2`, which uses 2 CPU cores to build your recipe.
PY_VER	Python version building against. Set with the `--python` argument or with the CONDA_PY environment variable.
NPY_VER	NumPy version to build against. Set with the `--numpy` argument or with the CONDA_NPY environment variable.
PATH	Inherited from your shell environment and augmented with `$PREFIX/bin`.
PREFIX	Build prefix to which the build script should install.
PKG_BUILDNUM	Build number of the package being built.
PKG_NAME	Name of the package being built.
PKG_VERSION	Version of the package being built.
PKG_BUILD_STRING	Complete build string of the package being built, including hash. EXAMPLE: py27h21422ab_0 . Conda-build 3.0+.
PKG_HASH	Hash of the package being built, without leading h. EXAMPLE: 21422ab . Conda-build 3.0+.
PYTHON	Path to the Python executable in the host prefix. Python is installed only in the host prefix when it is listed as a host requirement.
PY3K	`1` when Python 3 is installed in the build prefix, otherwise `0`.
R	Path to the R executable in the build prefix. R is only installed in the build prefix when it is listed as a build requirement.
RECIPE_DIR	Directory of the recipe.
SP_DIR	Python's site-packages location.
SRC_DIR	Path to where source is unpacked or cloned. If the source file is not a recognized file type---zip, tar, tar.bz2, or tar.xz---this is a directory containing a copy of the source file.
STDLIB_DIR	Python standard library location.
build_platform	The native subdir of the conda executable

Unix-style packages on Windows, which are usually statically linked to executables, are built in a special `Library` directory under the build prefix. The environment variables listed in the following table are defined only on Windows.

CYGWIN_PREFIX	Same as PREFIX, but as a Unix-style path, such as `/cygdrive/c/path/to/prefix`.
LIBRARY_BIN	`<build prefix>\Library\bin`.
LIBRARY_INC	`<build prefix>\Library\include`.
LIBRARY_LIB	`<build prefix>\Library\lib`.
LIBRARY_PREFIX	`<build prefix>\Library`.
SCRIPTS	`<build prefix>\Scripts`.
VS_MAJOR	The major version number of the Visual Studio version activated within the build, such as `9`.
VS_VERSION	The version number of the Visual Studio version activated within the build, such as `9.0`.
VS_YEAR	The release year of the Visual Studio version activated within the build, such as `2008`.

The environment variables listed in the following table are defined only on macOS and Linux.

HOME	Standard $HOME environment variable.
PKG_CONFIG_PATH	Path to `pkgconfig` directory.

The environment variables listed in the following table are defined only on macOS.

CFLAGS	`-arch` flag.
CXXFLAGS	Same as CFLAGS.
LDFLAGS	Same as CFLAGS.
MACOSX_DEPLOYMENT_TARGET	Same as the Anaconda Python macOS deployment target. Currently `10.9`.
OSX_ARCH	`i386` or `x86_64`, depending on Python build.

The environment variable listed in the following table is defined only on Linux.

LD_RUN_PATH	`<build prefix>/lib`.

3.4.3 Git environment variables

The environment variables listed in the following table are defined when the source is a git repository, specifying the source either with git_url or path.

GIT_BUILD_STR	String that joins GIT_DESCRIBE_NUMBER and GIT_DESCRIBE_HASH by an underscore.
GIT_DESCRIBE_HASH	The current commit short-hash as displayed from `git describe --tags`.
GIT_DESCRIBE_NUMBER	String denoting the number of commits since the most recent tag.
GIT_DESCRIBE_TAG	String denoting the most recent tag from the current commit, based on the output of `git describe --tags`.
GIT_FULL_HASH	String with the full SHA1 of the current HEAD.

These can be used in conjunction with templated `meta.yaml` files to set things---such as the build string---based on the state of the git repository.

3.4.4 Mercurial environment variables

The environment variables listed in the following table are defined when the source is a mercurial repository.

HG_BRANCH	String denoting the presently active branch.
HG_BUILD_STR	String that joins HG_NUM_ID and HG_SHORT_ID by an underscore.
HG_LATEST_TAG	String denoting the most recent tag from the current commit.
HG_LATEST_TAG_DISTANCE	String denoting number of commits since the most recent tag.
HG_NUM_ID	String denoting the revision number.
HG_SHORT_ID	String denoting the hash of the commit.

3.4.5 Inherited environment variables

Other than those mentioned above, no variables are inherited from the environment in which you invoke conda-build. You can choose to inherit additional environment variables by adding them to `meta.yaml`:

```
build:
  script_env:
   - TMPDIR
   - LD_LIBRARY_PATH # [linux]
   - DYLD_LIBRARY_PATH # [osx]
```

If an inherited variable is missing from your shell environment, it remains unassigned, but a warning is issued noting that it has no value assigned.

Additionally, values can be set by including = followed by the desired value:

```
build:
  script_env:
   - MY_VAR=some value
```

Warning: Inheriting environment variables can make it difficult for others to reproduce binaries from source with your recipe. Use this feature with caution or explicitly set values using the = syntax.

Note: If you split your build and test phases with `--no-test` and `--test`, you need to ensure that the environment variables present at build time and test time match. If you do not, the package hashes may use different values and your package may not be testable because the hashes will differ.

3.4.6 Environment variables that affect the build process

CONDA_PY	The Python version used to build the package. Should be `27`, `34`, `35`, `36`, or `37`.
CONDA_NPY	The NumPy version used to build the package, such as `19`, `110`, or `111`.
CONDA_PREFIX	The path to the conda environment used to build the package, such as `/path/to/conda/env`. Useful to pass as the environment prefix parameter to various conda tools, usually labeled `-p` or `--prefix`.

3.4.7 Environment variables that affect the test process

All of the above environment variables are also set during the test process, using the test prefix instead of the build prefix.

3.5 Using wheel files with conda

If you have software in a Python wheel file and want to use it with conda or install it in a conda environment, there are 3 ways.

The best way is to obtain the source code for the software and build a conda package from the source and not from a wheel. This helps ensure that the new package uses other conda packages to satisfy its dependencies.

The second best way is to build a conda package from the wheel file. This tells conda more about the files present than a pip install. It is also less likely than a pip install to cause errors by overwriting (or "clobbering") files. Building a conda package from the wheel file also has the advantage that any clobbering is more likely to happen at build time and not runtime.

The third way is to use pip to install a wheel file into a conda environment. Some conda users have used this option safely. The first 2 ways are still the safest and most reliable.

3.5.1 Building a conda package from a wheel file

To build a conda package from a wheel file, install the .whl file in the conda recipe's `bld.bat` or `build.sh` file.

You may download the .whl file in the source section of the conda recipe's `meta.yaml` file.

You may instead put the URL directly in the `pip install` command.

EXAMPLE: The conda recipe for TensorFlow has a `pip install` command in build.sh with the URL of a .whl file. The meta.yaml file does not download or list the .whl file.

Note: It is important to `pip install` only the one desired package. Whenever possible, install dependencies with conda and not pip.

We strongly recommend using the `--no-deps` option in the `pip install` command.

If you run `pip install` without the `--no-deps` option, pip will often install dependencies in your conda recipe and those dependencies will become part of your package. This wastes space in the package and increases the risk of file overlap, file clobbering, and broken packages.

Tutorials

The *tutorials* will guide you through how to build conda packages---whether you're creating a package with compilers, using conda skeleton, creating from scratch, or building R packages using skeleton CRAN.

Recipes

Conda-build uses *recipes* to create conda packages. We have guides on debugging conda recipes, sample recipes for you to use, and information on how to build a package without a recipe.

Environment variables

Use our *environment variables* guide to understand which environment variables are available, set, and inherited, and how they affect different processes.

Wheel files

The user guide includes information about *wheel files* and how to build conda packages from them.

CHAPTER

FOUR

RESOURCES

These resources will help you accomplish more using conda-build. We provide information on topics including build scripts, build variants, making packages relocatable, and defining metadata in the meta.yaml. We also provide guidelines and a template for submitting your own documentation.

4.1 Build scripts (build.sh, bld.bat)

The `build.sh` file is the build script for Linux and macOS and `bld.bat` is the build script for Windows. These scripts contain the logic that carries out your build steps. Traditionally it has also included install steps. With the traditional one-package-per-recipe way of doing things, anything that your build script copies into the `$PREFIX` or `%PREFIX%` folder will be included in your output package. For example, this `build.sh`:

```
mkdir -p $PREFIX/bin
cp $RECIPE_DIR/my_script_with_recipe.sh $PREFIX/bin/super-cool-script.sh
```

If you don't care about deploying your package with pip on PyPI, this can save you a lot of time in figuring out the proper way to include additional files with setup.py.

There are many environment variables defined for you to use in build.sh and bld.bat. Please see *Environment variables* for more information.

As of conda-build 2.1, you can also define multiple output packages. Each package has its own script or list of files to include. The rules for these outputs are documented at *Outputs section*. When any output is defined, this overrides the default behavior of bundling anything in `$PREFIX`. So to output multiple packages from a single recipe, remove any installation steps from `build.sh` or `bld.bat` and do them instead in your install script(s) for each output.

`build.sh` and `bld.bat` are optional. You can instead use the `build/script` key in your `meta.yaml`, with each value being either a string command or a list of string commands. Any commands you put there must be able to run on every platform for which you build. For example, you can't use the `cp` command because cmd.exe won't understand it in Windows.

`build.sh` is run with `bash` and `bld.bat` is run with `cmd.exe`.

There is some development towards the ability to use bash scripts in Windows, but this is not currently supported. You may write your script as a .sh file, and then call it in your bld.bat file, but there is no way to directly run build.sh on Windows. The conda recipe at https://github.com/AnacondaRecipes/conda-feedstock/tree/master/recipe is an example of this method.

4.2 Anaconda compiler tools

Anaconda 5.0 switched from OS-provided compiler tools to our own toolsets. This allows improved compiler capabilities, including better security and performance. This page describes how to use these tools and enable these benefits.

4.2.1 Compiler packages

Before Anaconda 5.0, compilers were installed using system tools such as XCode or `yum install gcc`. Now there are conda packages for Linux and macOS compilers. Unlike the previous GCC 4.8.5 packages that included GCC, g++, and GFortran all in the same package, these conda packages are split into separate compilers:

macOS:

- clang_osx-64.
- clangxx_osx-64.
- gfortran_osx-64.

Linux:

- gcc_linux-64.
- gxx_linux-64.
- gfortran_linux-64.

A compiler's "build platform" is the platform where the compiler runs and builds the code.

A compiler's "host platform" is the platform where the built code will finally be hosted and run.

Notice that all of these package names end in a platform identifier which specifies the host platform. All compiler packages are specific to both the build platform and the host platform.

4.2.2 Using the compiler packages

The compiler packages can be installed with conda. Because they are designed with (pseudo) cross-compiling in mind, all of the executables in a compiler package are "prefixed." Instead of `gcc`, the executable name of the compiler you use will be something like `x86_64-conda_cos6-linux-gnu-gcc`. These full compiler names are shown in the build logs, recording the host platform and helping prevent the common mistake of using the wrong compiler.

Many build tools such as `make` and `CMake` search by default for a compiler named simply `gcc`, so we set environment variables to point these tools to the correct compiler.

We set these variables in conda `activate.d` scripts, so any environment in which you will use the compilers must first be activated so the scripts will run. Conda-build does this activation for you using activation hooks installed with the compiler packages in `CONDA_PREFIX/etc/conda/activate.d`, so no additional effort is necessary.

You can activate the root environment with the command `conda activate root`.

4.2.3 macOS SDK

The macOS compilers require the macOS 10.9 SDK or above. The SDK license prevents it from being bundled in the conda package. We know of 2 current sources for the macOS SDKs:

- https://github.com/devernay/xcodelegacy
- https://github.com/phracker/MacOSX-SDKs

We usually install the 10.10 SDK at `/opt/MacOSX10.10.sdk` but you may install it anywhere. Edit your `conda_build_config.yaml` file to point to it, like this:

```
CONDA_BUILD_SYSROOT:
  - /opt/MacOSX10.10.sdk        # [osx]
```

At Anaconda, we have this configuration setting in a centralized `conda_build_config.yaml` at the root of our recipe repository. Since we run build commands from that location, the file and the setting are used for all recipes. The `conda_build_config.yaml` search order is described further at *Creating conda-build variant config files*.

Build scripts for macOS should make use of the variables `MACOSX_DEPLOYMENT_TARGET` and `CONDA_BUILD_SYSROOT`, which are set by conda-build (see *Environment variables*). These variables should be translated into correct compiler arguments, e.g. for Clang this would be:

```
clang .. -isysroot ${CONDA_BUILD_SYSROOT} -mmacosx-version-min=${MACOSX_DEPLOYMENT_
↪TARGET} ..
```

Most build tools, e.g. CMake and distutils (setuptools), will automatically pick up `MACOSX_DEPLOYMENT_TARGET` but you need to pass `CONDA_BUILD_SYSROOT` explicitly. For CMake, this can be done with the option `-DCMAKE_OSX_SYSROOT=${CONDA_BUILD_SYSROOT}`. When building Python extensions with distutils, one should always extend `CFLAGS` before calling `setup.py`:

```
export CFLAGS="${CFLAGS} -i sysroot ${CONDA_BUILD_SYSROOT}"
```

When building C++ extensions with Cython, `CXXFLAGS` must be similarly modified.

4.2.4 Backward compatibility

Some users want to use the latest Anaconda packages but do not yet want to use the Anaconda compilers. To enable this, the latest Python package builds have a default `_sysconfigdata` file. This file sets the compilers provided by the system, such as `gcc` and `g++`, as the default compilers. This way allows legacy recipes to keep working.

Python packages also include an alternative `_sysconfigdata` file that sets the Anaconda compilers as the default compilers. The Anaconda Python executable itself is made with these Anaconda compilers.

The compiler packages set the environment variable `_PYTHON_SYSCONFIGDATA_NAME`, which tells Python which `_sysconfigdata` file to use. This variable is set at activation time using the activation hooks described above.

The new `_sysconfigdata` customization system is only present in recent versions of the Python package. Conda-build automatically tries to use the latest Python version available in the currently configured channels, which normally gets the latest from the default channel. If you're using something other than conda-build while working with the new compilers, conda does not automatically update Python, so make sure you have the correct `_sysconfigdata` files by updating your Python package manually.

4.2.5 Anaconda compilers and conda-build 3

The Anaconda 5.0 compilers and conda-build 3 are designed to work together.

Conda-build 3 defines a special jinja2 function, `compiler()`, to make it easy to specify compiler packages dynamically on many platforms. The `compiler` function takes at least 1 argument, the language of the compiler to use:

```
requirements:
  build:
    - {{ compiler('c') }}
```

"Cross-capable" recipes can be used to make packages with a host platform different than the build platform where conda-build runs. To write cross-capable recipes, you may also need to use the "host" section in the requirements section. In this example we set "host" to "zlib" to tell conda-build to use the zlib in the conda environment and not the system zlib. This makes sure conda-build uses the zlib for the host platform and not the zlib for the build platform.

```
requirements:
  build:
    - {{ compiler('c') }}
  host:
    - zlib
```

Generally, the build section should include compilers and other build tools and the host section should include everything else, including shared libraries, Python, and Python libraries.

4.2.6 An aside on CMake and sysroots

Anaconda's compilers for Linux are built with something called crosstool-ng. They include not only GCC, but also a "sysroot" with glibc, as well as the rest of the toolchain (binutils). Ordinarily, the sysroot is something that your system provides, and it is what establishes the libc compatibility bound for your compiled code. Any compilation that uses a sysroot other than the system sysroot is said to be "cross-compiling." When the target OS and the build OS are the same, it is called a "pseudo-cross-compiler." This is the case for normal builds with Anaconda's compilers on Linux.

Unfortunately, some software tools do not handle sysroots in intuitive ways. CMake is especially bad for this. Even though the compiler itself understands its own sysroot, CMake insists on ignoring that. We've filed issues at:

- https://gitlab.kitware.com/cmake/cmake/issues/17483

Additionally, this Stack Overflow issue has some more information: https://stackoverflow.com/questions/36195791/cmake-missing-sysroot-when-cross-compiling

In order to teach CMake about the sysroot, you must do additional work. As an example, please see our recipe for libnetcdf at https://github.com/AnacondaRecipes/libnetcdf-feedstock/tree/master/recipe

In particular, you'll need to copy the `cross-linux.cmake` file there, and reference it in your build.sh file:

```
CMAKE_PLATFORM_FLAGS+=(-DCMAKE_TOOLCHAIN_FILE="${RECIPE_DIR}/cross-linux.cmake")

cmake -DCMAKE_INSTALL_PREFIX=${PREFIX} \
  ${CMAKE_PLATFORM_FLAGS[@]} \
  ${SRC_DIR}
```

4.2.7 Customizing the compilers

The compiler packages listed above are small packages that only include the activation scripts and list most of the software they provide as runtime dependencies.

This design is intended to make it easy for you to customize your own compiler packages by copying these recipes and changing the flags. You can then edit the `conda_build_config.yaml` file to specify your own packages.

We have been careful to select good, general purpose, secure, and fast flags. We have also used them for all packages in Anaconda Distribution 5.0.0, except for some minor customizations in a few recipes. When changing these flags, remember that choosing the wrong flags can reduce security, reduce performance, and cause incompatibilities.

With that warning in mind, let's look at good ways to customize Clang.

1. Download or fork the code from https://github.com/anacondarecipes/aggregate. The Clang package recipe is in the `clang` folder. The main material is in the llvm-compilers-feedstock folder.

2. Edit `clang/recipe/meta.yaml`:

```
package:
  name: clang_{{ target_platform }}
  version: {{ version }}
```

   The name here does not matter but the output names below do. Conda-build expects any compiler to follow the BASENAME_PLATFORMNAME pattern, so it is important to keep the `{{target_platform}}` part of the name.

   `{{ version }}` is left as an intentionally undefined jinja2 variable. It is set later in `conda_build_config.yaml`.

3. Before any packaging is done, run the build.sh script: https://github.com/AnacondaRecipes/aggregate/blob/master/clang/build.sh

   In this recipe, values are changed here. Those values are inserted into the activate scripts that are installed later.

```
#!/bin/bash

CHOST=${macos_machine}

FINAL_CPPFLAGS="-D_FORTIFY_SOURCE=2 -mmacosx-version-min=${macos_min_version}"
FINAL_CFLAGS="-march=core2 -mtune=haswell -mssse3 -ftree-vectorize -fPIC -fPIE -
↪fstack-protector-strong -O2 -pipe"
FINAL_CXXFLAGS="-march=core2 -mtune=haswell -mssse3 -ftree-vectorize -fPIC -fPIE -
↪fstack-protector-strong -O2 -pipe -stdlib=libc++ -fvisibility-inlines-hidden -
↪std=c++14 -fmessage-length=0"
# These are the LDFLAGS for when the linker is being called directly, without "-Wl,"
FINAL_LDFLAGS="-pie -headerpad_max_install_names"
# These are the LDFLAGS for when the linker is being driven by a compiler, with "-
↪Wl,"
FINAL_LDFLAGS_CC="-Wl,-pie -Wl,-headerpad_max_install_names"
FINAL_DEBUG_CFLAGS="-Og -g -Wall -Wextra -fcheck=all -fbacktrace -fimplicit-none -
↪fvar-tracking-assignments"
FINAL_DEBUG_CXXFLAGS="-Og -g -Wall -Wextra -fcheck=all -fbacktrace -fimplicit-none -
↪fvar-tracking-assignments"
FINAL_DEBUG_FFLAGS="-Og -g -Wall -Wextra -fcheck=all -fbacktrace -fimplicit-none -
↪fvar-tracking-assignments"
```

(continues on next page)

(continued from previous page)

```
find "${RECIPE_DIR}" -name "*activate*.sh" -exec cp {} . \;

find . -name "*activate*.sh" -exec sed -i.bak "s|@CHOST@|${CHOST}|g" "{}" \;
find . -name "*activate*.sh" -exec sed -i.bak "s|@CPPFLAGS@|${FINAL_CPPFLAGS}|g"
↪         "{}" \;
find . -name "*activate*.sh" -exec sed -i.bak "s|@CFLAGS@|${FINAL_CFLAGS}|g"
↪         "{}" \;
find . -name "*activate*.sh" -exec sed -i.bak "s|@DEBUG_CFLAGS@|${FINAL_DEBUG_
↪CFLAGS}|g"     "{}" \;
find . -name "*activate*.sh" -exec sed -i.bak "s|@CXXFLAGS@|${FINAL_CXXFLAGS}|g"
↪         "{}" \;
find . -name "*activate*.sh" -exec sed -i.bak "s|@DEBUG_CXXFLAGS@|${FINAL_DEBUG_
↪CXXFLAGS}|g" "{}" \;
find . -name "*activate*.sh" -exec sed -i.bak "s|@DEBUG_CXXFLAGS@|${FINAL_DEBUG_
↪CXXFLAGS}|g" "{}" \;
# find . -name "*activate*.sh" -exec sed -i.bak "s|@FFLAGS@|${FINAL_FFLAGS}|g"
↪           "{}" \;
# find . -name "*activate*.sh" -exec sed -i.bak "s|@DEBUG_FFLAGS@|${FINAL_DEBUG_
↪FFLAGS}|g"     "{}" \;
find . -name "*activate*.sh" -exec sed -i.bak "s|@LDFLAGS@|${FINAL_LDFLAGS}|g"
↪         "{}" \;
find . -name "*activate*.sh" -exec sed -i.bak "s|@LDFLAGS_CC@|${FINAL_LDFLAGS_CC}|g
↪"         "{}" \;
find . -name "*activate*.sh.bak" -exec rm "{}" \;
```

4. With those changes to the activate scripts in place, it's time to move on to installing things. Look back at the `clang` folder's `meta.yaml`. Here's where we change the package name. Notice what comes before the `{{ target_platform }}`.

```
outputs:
  - name: super_duper_clang_{{ target_platform }}
    script: install-clang.sh
    requirements:
      - clang {{ version }}
```

   The script reference here is another place you might add customization. You'll either change the contents of those install scripts or change the scripts that those install scripts are installing.

   Note that we make the package `clang` in the main material agree in version with our output version. This is implicitly the same as the top-level recipe. The `clang` package sets no environment variables at all, so it may be difficult to use directly.

5. Let's examine the script `install-clang.sh`:

```
#!/bin/bash

set -e -x

CHOST=${macos_machine}

mkdir -p "${PREFIX}"/etc/conda/{de,}activate.d/
cp "${SRC_DIR}"/activate-clang.sh "${PREFIX}"/etc/conda/activate.d/activate_"${PKG_
↪NAME}".sh
```

(continues on next page)

(continued from previous page)

```
cp "${SRC_DIR}"/deactivate-clang.sh "${PREFIX}"/etc/conda/deactivate.d/deactivate_"$
↪{PKG_NAME}".sh

pushd "${PREFIX}"/bin
  ln -s clang ${CHOST}-clang
popd
```

Nothing here is too unusual.

Activate scripts are named according to our package name so they won't conflict with other activate scripts.

The symlink for Clang is a Clang implementation detail that sets the host platform.

We define `macos_machine` in aggregate's `conda_build_config.yaml`: https://github.com/AnacondaRecipes/aggregate/blob/master/conda_build_config.yaml#L79

The activate scripts that are being installed are where we actually set the environment variables. Remember that these have been modified by build.sh.

6. With any of your desired changes in place, go ahead and build the recipe.

   You should end up with a super_duper_clang_osx-64 package. Or, if you're not on macOS and are modifying a different recipe, you should end up with an equivalent package for your platform.

4.2.8 Using your customized compiler package with conda-build 3

Remember the Jinja2 function, `{{ compiler('c') }}`? Here's where that comes in. Specific keys in `conda_build_config.yaml` are named for the language argument to that jinja2 function. In your `conda_build_config.yaml`, add this:

```
c_compiler:
  - super_duper_clang
```

Note that we're not adding the `target_platform` part, which is separate. You can define that key, too:

```
c_compiler:
  - super_duper_clang
target_platform:
  - win-64
```

With those two keys defined, conda-build will try to use a compiler package named `super_duper_clang_win-64`. That package needs to exist for your native platform. For example, if you're on macOS, your native platform is `osx-64`.

The package subdirectory for your native platform is the build platform. The build platform and the `target_platform` can be the same, and they are the same by default, but they can also be different. When they are different, you're cross-compiling.

If you ever needed a different compiler key for the same language, remember that the language key is arbitrary. For example, we might want different compilers for Python and for R within one ecosystem. On Windows, the Python ecosystem uses the Microsoft Visual C compilers, while the R ecosystem uses the Mingw compilers.

Let's start in `conda_build_config.yaml`:

```
python_c_compiler:
  - vs2015
r_c_compiler:
```

(continues on next page)

(continued from previous page)

```
  - m2w64-gcc
target_platform:
  - win-64
```

In Python recipes, you'd have:

```
requirements:
  build:
    - {{ compiler('python_c') }}
```

In R recipes, you'd have:

```
requirements:
  build:
    - {{ compiler('r_c') }}
```

This example is a little contrived, because the `m2w64-gcc_win-64` package is not available. You'd need to create a metapackage `m2w64-gcc_win-64` to point at the `m2w64-gcc` package, which does exist on the msys2 channel on repo.anaconda.com.

4.2.9 Anaconda compilers implicitly add RPATH pointing to the conda environment

You might want to use the Anaconda compilers outside of `conda-build` so that you use the same versions, flags, and configuration, for maximum compatibility with Anaconda packages (but in a case where you want simple tarballs, for example). In this case, there is a gotcha.

Even if Anaconda compilers are used from outside of `conda-build`, the GCC specs are customized so that, when linking an executable or a shared library, an RPATH pointing to `lib/` inside the current enviroment prefix directory (`$CONDA_PREFIX/lib`) is added. This is done by changing the `link_libgcc:` section inside GCC `specs` file, and this change is done so that `LD_LIBRARY_PATH` isn't required for basic libraries.

`conda-build` knows how to make this automatically relocatable, so that this `RPATH` will be changed to point to the environment where the package is being installed (at installation time, by `conda`). But if you only pack this binary in a tarball, it will continue containing this hardcoded `RPATH` to an environment in your machine. In this case, it is recommended to manually remove the `RPATH`.

4.3 Defining metadata (meta.yaml)

- *Package section*
- *Source section*
- *Build section*
- *Requirements section*
- *Test section*
- *Outputs section*
- *About section*
- *App section*

- *Extra section*
- *Templating with Jinja*
- *Preprocessing selectors*

All the metadata in the conda-build recipe is specified in the `meta.yaml` file. See the example below:

```
{% set version = "1.1.0" %}

package:
  name: imagesize
  version: {{ version }}

source:
  url: https://pypi.io/packages/source/i/imagesize/imagesize-{{ version }}.tar.gz
  sha256: f3832918bc3c66617f92e35f5d70729187676313caa60c187eb0f28b8fe5e3b5

build:
  noarch: python
  number: 0
  script: python -m pip install --no-deps --ignore-installed .

requirements:
  host:
    - python
    - pip
  run:
    - python

test:
  imports:
    - imagesize

about:
  home: https://github.com/shibukawa/imagesize_py
  license: MIT
  summary: 'Getting image size from png/jpeg/jpeg2000/gif file'
  description: |
    This module analyzes jpeg/jpeg2000/png/gif image header and
    return image size.
  dev_url: https://github.com/shibukawa/imagesize_py
  doc_url: https://pypi.python.org/pypi/imagesize
  doc_source_url: https://github.com/shibukawa/imagesize_py/blob/master/README.rst
```

All sections are optional except for `package/name` and `package/version`.

Headers must appear only once. If they appear multiple times, only the last is remembered. For example, the `package:` header should appear only once in the file.

4.3.1 Package section

Specifies package information.

Package name

The lower case name of the package. It may contain "-", but no spaces.

```
package:
  name: bsdiff4
```

Package version

The version number of the package. Use the PEP-386 verlib conventions. Cannot contain "-". YAML interprets version numbers such as 1.0 as floats, meaning that 0.10 will be the same as 0.1. To avoid this, put the version number in quotes so that it is interpreted as a string.

```
package:
  version: "1.1.4"
```

Note: Post-build versioning: In some cases, you may not know the version, build number, or build string of the package until after it is built. In these cases, you can perform *Templating with Jinja* or utilize *Git environment variables* and *Inherited environment variables*.

4.3.2 Source section

Specifies where the source code of the package is coming from. The source may come from a tarball file, git, hg, or svn. It may be a local path and it may contain patches.

Source from tarball or zip archive

```
source:
  url: https://pypi.python.org/packages/source/b/bsdiff4/bsdiff4-1.1.4.tar.gz
  md5: 29f6089290505fc1a852e176bd276c43
  sha1: f0a2c9a30073449cfb7d171c57552f3109d93894
  sha256: 5a022ff4c1d1de87232b1c70bde50afbb98212fd246be4a867d8737173cf1f8f
```

If an extracted archive contains only 1 folder at its top level, its contents will be moved 1 level up, so that the extracted package contents sit in the root of the work folder.

Source from git

The git_url can also be a relative path to the recipe directory.

```
source:
  git_url: https://github.com/ilanschnell/bsdiff4.git
  git_rev: 1.1.4
  git_depth: 1 # (Defaults to -1/not shallow)
```

The depth argument relates to the ability to perform a shallow clone. A shallow clone means that you only download part of the history from Git. If you know that you only need the most recent changes, you can say, `git_depth: 1`, which is faster than cloning the entire repo. The downside to setting it at 1 is that, unless the tag is on that specific commit, then you won't have that tag when you go to reference it in `git_rev` (for example). If your `git_depth` is insufficient to capture the tag in `git_rev`, you'll encounter an error. So in the example above, unless the 1.1.4 is the very head commit and the one that you're going to grab, you may encounter an error.

Source from hg

```
source:
  hg_url: ssh://hg@bitbucket.org/ilanschnell/bsdiff4
  hg_tag: 1.1.4
```

Source from svn

```
source:
  svn_url: https://github.com/ilanschnell/bsdiff
  svn_rev: 1.1.4
  svn_ignore_externals: True # (defaults to False)
```

Source from a local path

If the path is relative, it is taken relative to the recipe directory. The source is copied to the work directory before building.

```
source:
  path: ../src
```

If the local path is a git or svn repository, you get the corresponding environment variables defined in your build environment. The only practical difference between git_url or hg_url and path as source arguments is that git_url and hg_url would be clones of a repository, while path would be a copy of the repository. Using path allows you to build packages with unstaged and uncommitted changes in the working directory. git_url can build only up to the latest commit.

Patches

Patches may optionally be applied to the source.

```
source:
  #[source information here]
  patches:
    - my.patch # the patch file is expected to be found in the recipe
```

Conda-build automatically determines the patch strip level.

Destination path

Within conda-build's work directory, you may specify a particular folder to place source into. This feature is new in conda-build 3.0. Conda-build will always drop you into the same folder (build folder/work), but it's up to you whether you want your source extracted into that folder, or nested deeper. This feature is particularly useful when dealing with multiple sources, but can apply to recipes with single sources as well.

```
source:
  #[source information here]
  folder: my-destination/folder
```

Filename

The filename key is `fn`. It was formerly required with URL source types. It is not required now.

If the `fn` key is provided, the file is saved on disk with that name. If the `fn` key is not provided, the file is saved on disk with a name matching the last part of the URL.

For example, `http://www.something.com/myfile.zip` has an implicit filename of `myfile.zip`. Users may change this by manually specifying `fn`.

```
source:
  url: http://www.something.com/myfile.zip
  fn: otherfilename.zip
```

Source from multiple sources

Some software is most easily built by aggregating several pieces. For this, conda-build 3.0 has added support for arbitrarily specifying many sources.

The syntax is a list of source dictionaries. Each member of this list follows the same rules as the single source for earlier conda-build versions (listed above). All features for each member are supported.

Example:

```
source:
  - url: https://package1.com/a.tar.bz2
    folder: stuff
  - url: https://package1.com/b.tar.bz2
    folder: stuff
  - git_url: https://github.com/conda/conda-build
    folder: conda-build
```

Here, the two URL tarballs will go into one folder, and the git repo is checked out into its own space. Git will not clone into a non-empty folder.

Note: Dashes denote list items in YAML syntax.

4.3.3 Build section

Specifies build information.

Each field that expects a path can also handle a glob pattern. The matching is performed from the top of the build environment, so to match files inside your project you can use a pattern similar to the following one: "**/myproject/**/*.txt". This pattern will match any .txt file found in your project.

Note: The quotation marks ("") are required for patterns that start with a *.

Recursive globbing using ** is supported only in conda-build >= 3.0.

Build number and string

The build number should be incremented for new builds of the same version. The number defaults to `0`. The build string cannot contain "-". The string defaults to the default conda-build string plus the build number.

```
build:
  number: 1
  string: abc
```

A hash will appear when the package is affected by one or more variables from the conda_build_config.yaml file. The hash is made up from the "used" variables - if anything is used, you have a hash. If you don't use these variables then you won't have a hash. There are a few special cases that do not affect the hash, such as Python and R or anything that already had a place in the build string.

The build hash will be added to the build string if these are true for any dependency:

- package is an explicit dependency in build, host, or run deps
- package has a matching entry in conda_build_config.yaml which is a pin to a specific version, not a lower bound
- that package is not ignored by ignore_version

OR

- package uses {{ compiler() }} jinja2 function

Python entry points

The following example creates a Python entry point named "bsdiff4" that calls `bsdiff4.cli.main_bsdiff4()`.

```
build:
  entry_points:
    - bsdiff4 = bsdiff4.cli:main_bsdiff4
    - bspatch4 = bsdiff4.cli:main_bspatch4
```

Python.app

If osx_is_app is set, entry points use `python.app` instead of Python in macOS. The default is `False`.

```
build:
  osx_is_app: True
```

Track features

Adding track_features to one or more of the options will cause conda to de-prioritize it or "weigh it down." The lowest priority package is the one that would cause the most track_features to be activated in the environment. The default package among many variants is the one that would cause the least track_features to be activated.

No two packages in a given subdir should ever have the same track_feature.

```
build:
  track_features:
    - feature2
```

Preserve Python egg directory

This is needed for some packages that use features specific to setuptools. The default is `False`.

```
build:
  preserve_egg_dir: True
```

Skip compiling some .py files into .pyc files

Some packages ship `.py` files that cannot be compiled, such as those that contain templates. Some packages also ship `.py` files that should not be compiled yet, because the Python interpreter that will be used is not known at build time. In these cases, conda-build can skip attempting to compile these files. The patterns used in this section do not need the ** to handle recursive paths.

```
build:
  skip_compile_pyc:
    - "*/templates/*.py"          # These should not (and cannot) be compiled
    - "*/share/plugins/gdb/*.py"  # The python embedded into gdb is unknown
```

No link

A list of globs for files that should always be copied and never soft linked or hard linked.

```
build:
  no_link:
    - bin/*.py  # Don't link any .py files in bin/
```

Script

Used instead of `build.sh` or `bld.bat`. For short build scripts, this can be more convenient. You may need to use *selectors* to use different scripts for different platforms.

```
build:
  script: python setup.py install --single-version-externally-managed --record=record.txt
```

RPATHs

Set which RPATHs are used when making executables relocatable on Linux. This is a Linux feature that is ignored on other systems. The default is `lib/`.

```
build:
  rpaths:
    - lib/
    - lib/R/lib/
```

Force files

Force files to always be included, even if they are already in the environment from the build dependencies. This may be needed, for example, to create a recipe for conda itself.

```
build:
  always_include_files:
    - bin/file1
    - bin/file2
```

Relocation

Advanced features. You can use the following 4 keys to control relocatability files from the build environment to the installation environment:

- binary_relocation.
- has_prefix_files.
- binary_has_prefix_files.
- ignore_prefix_files.

For more information, see *Making packages relocatable*.

Binary relocation

Whether binary files should be made relocatable using install_name_tool on macOS or patchelf on Linux. The default is `True`. It also accepts `False`, which indicates no relocation for any files, or a list of files, which indicates relocation only for listed files.

```
build:
  binary_relocation: False
```

Detect binary files with prefix

Binary files may contain the build prefix and need it replaced with the install prefix at installation time. Conda can automatically identify and register such files. The default is `True`.

Note: The default changed from `False` to `True` in conda build 2.0. Setting this to `False` means that binary relocation---RPATH---replacement will still be done, but hard-coded prefixes in binaries will not be replaced. Prefixes in text files will still be replaced.

```
build:
  detect_binary_files_with_prefix: False
```

Windows handles binary prefix replacement very differently than Unix-like systems such as macOS and Linux. At this time, we are unaware of any executable or library that uses hardcoded embedded paths for locating other libraries or program data on Windows. Instead, Windows follows DLL search path rules or more natively supports relocatability using relative paths. Because of this, conda ignores most prefixes. However, pip creates executables for Python entry points that do use embedded paths on Windows. Conda-build thus detects prefixes in all files and records them by default. If you are getting errors about path length on Windows, you should try to disable detect_binary_files_with_prefix. Newer versions of Conda, such as recent 4.2.x series releases and up, should have no problems here, but earlier versions of conda do erroneously try to apply any binary prefix replacement.

Binary has prefix files

By default, conda-build tries to detect prefixes in all files. You may also elect to specify files with binary prefixes individually. This allows you to specify the type of file as binary, when it may be incorrectly detected as text for some reason. Binary files are those containing NULL bytes.

```
build:
  binary_has_prefix_files:
    - bin/binaryfile1
    - lib/binaryfile2
```

Text files with prefix files

Text files---files containing no NULL bytes---may contain the build prefix and need it replaced with the install prefix at installation time. Conda will automatically register such files. Binary files that contain the build prefix are generally handled differently---see *Binary has prefix files*---but there may be cases where such a binary file needs to be treated as an ordinary text file, in which case they need to be identified.

```
build:
  has_prefix_files:
    - bin/file1
    - lib/file2
```

Ignore prefix files

Used to exclude some or all of the files in the build recipe from the list of files that have the build prefix replaced with the install prefix.

To ignore all files in the build recipe, use:

```
build:
  ignore_prefix_files: True
```

To specify individual filenames, use:

```
build:
  ignore_prefix_files:
    - file1
```

This setting is independent of RPATH replacement. Use the *Detect binary files with prefix* setting to control that behavior.

Skipping builds

Specifies whether conda-build should skip the build of this recipe. Particularly useful for defining recipes that are platform specific. The default is `False`.

```
build:
  skip: True  # [not win]
```

Architecture independent packages

Allows you to specify "no architecture" when building a package, thus making it compatible with all platforms and architectures. Noarch packages can be installed on any platform.

Starting with conda-build 2.1, and conda 4.3, there is a new syntax that supports different languages. Assigning the noarch key as `generic` tells conda to not try any manipulation of the contents.

```
build:
  noarch: generic
```

`noarch: generic` is most useful for packages such as static javascript assets and source archives. For pure Python packages that can run on any Python version, you can use the `noarch: python` value instead:

```
build:
  noarch: python
```

The legacy syntax for `noarch_python` is still valid, and should be used when you need to be certain that your package will be installable where conda 4.3 is not yet available. All other forms of noarch packages require conda >=4.3 to install.

```
build:
  noarch_python: True
```

Warning: At the time of this writing, `noarch` packages should not make use of *preprocess-selectors*: `noarch` packages are built with the directives which evaluate to `True` in the platform it was built, which probably will result in incorrect/incomplete installation in other platforms.

Include build recipe

The full conda-build recipe and rendered `meta.yaml` file is included in the *Package metadata* by default. You can disable this with:

```
build:
  include_recipe: False
```

Use environment variables

Normally the build script in `build.sh` or `bld.bat` does not pass through environment variables from the command line. Only environment variables documented in *Environment variables* are seen by the build script. To "white-list" environment variables that should be passed through to the build script:

```
build:
  script_env:
    - MYVAR
    - ANOTHER_VAR
```

If a listed environment variable is missing from the environment seen by the conda-build process itself, a UserWarning is emitted during the build process and the variable remains undefined.

Additionally, values can be set by including = followed by the desired value:

```
build:
  script_env:
   - MY_VAR=some value
```

Note: Inheriting environment variables can make it difficult for others to reproduce binaries from source with your recipe. Use this feature with caution or explicitly set values using the = syntax.

Note: If you split your build and test phases with `--no-test` and `--test`, you need to ensure that the environment variables present at build time and test time match. If you do not, the package hashes may use different values, and your package may not be testable, because the hashes will differ.

Export runtime requirements

Some build or host *Requirements section* will impose a runtime requirement. Most commonly this is true for shared libraries (e.g. libpng), which are required for linking at build time, and for resolving the link at run time. With `run_exports` (new in conda-build 3) such a runtime requirement can be implicitly added by host requirements (e.g. libpng exports libpng), and with `run_exports/strong` even by build requirements (e.g. GCC exports libgcc).

```
# meta.yaml of libpng
build:
  run_exports:
    - libpng
```

Here, because no specific kind of `run_exports` is specified, libpng's `run_exports` are considered "weak." This means they will only apply when libpng is in the host section, when they will add their export to the run section. If libpng were listed in the build section, the `run_exports` would not apply to the run section.

```
# meta.yaml of gcc compiler
build:
  run_exports:
    strong:
      - libgcc
```

Strong `run_exports` are used for things like runtimes, where the same runtime needs to be present in the host and the run environment, and exactly which runtime that should be is determined by what's present in the build section. This mechanism is how we line up appropriate software on Windows, where we must match MSVC versions used across all of the shared libraries in an environment.

```
# meta.yaml of some package using gcc and libpng
requirements:
  build:
    - gcc            # has a strong run export
  host:
    - libpng         # has a (weak) run export
    # - libgcc       <-- implicitly added by gcc
  run:
    # - libgcc       <-- implicitly added by gcc
    # - libpng       <-- implicitly added by libpng
```

You can express version constraints directly, or use any of the Jinja2 helper functions listed at *Extra Jinja2 functions*.

For example, you may use *Pinning expressions* to obtain flexible version pinning relative to versions present at build time:

```
build:
  run_exports:
    - {{ pin_subpackage('libpng', max_pin='x.x') }}
```

With this example, if libpng were version 1.6.34, this pinning expression would evaluate to `>=1.6.34,<1.7`.

If build and link dependencies need to impose constraints on the run environment but not necessarily pull in additional packages, then this can be done by altering the *Run_constrained* entries. In addition to `weak`/`strong` `run_exports` which add to the `run` requirements, `weak_constrains` and `strong_constrains` add to the `run_constrained` requirements. With these, e.g., minimum versions of compatible but not required packages (like optional plugins for the linked dependency, or certain system attributes) can be expressed:

```
requirements:
  build:
    - build-tool                # has a strong run_constrained export
  host:
    - link-dependency           # has a weak run_constrained export
  run:
  run_constrained:
    # - system-dependency >=min  <-- implicitly added by build-tool
    # - optional-plugin >=min    <-- implicitly added by link-dependency
```

Note that `run_exports` can be specified both in the build section and on a per-output basis for split packages.

`run_exports` only affects directly named dependencies. For example, if you have a metapackage that includes a compiler that lists `run_exports`, you also need to define `run_exports` in the metapackage so that it takes effect when people install your metapackage. This is important, because if `run_exports` affected transitive dependencies, you would see many added dependencies to shared libraries where they are not actually direct dependencies. For example, Python uses bzip2, which can use `run_exports` to make sure that people use a compatible build of bzip2. If people list python as a build time dependency, bzip2 should only be imposed for Python itself and should not be automatically imposed as a runtime dependency for the thing using Python.

The potential downside of this feature is that it takes some control over constraints away from downstream users. If an upstream package has a problematic `run_exports` constraint, you can ignore it in your recipe by listing the upstream package name in the `build/ignore_run_exports` section:

```
build:
  ignore_run_exports:
    - libstdc++
```

You can also list the package the `run_exports` constraint is coming from using the `build/ignore_run_exports_from` section:

```
build:
  ignore_run_exports_from:
    - {{ compiler('cxx') }}
```

Pin runtime dependencies

The `pin_depends` build key can be used to enforce pinning behavior on the output recipe or built package.

There are 2 possible behaviors:

```
build:
  pin_depends: record
```

With a value of `record`, conda-build will record all requirements exactly as they would be installed in a file called info/requires. These pins will not show up in the output of `conda render` and they will not affect the actual run dependencies of the output package. It is only adding in this new file.

```
build:
  pin_depends: strict
```

With a value of `strict`, conda-build applies the pins to the actual metadata. This does affect the output of `conda render` and also affects the end result of the build. The package dependencies will be strictly pinned down to the build string level. This will supersede any dynamic or compatible pinning that conda-build may otherwise be doing.

Whitelisting shared libraries

The `missing_dso_whitelist` build key is a list of globs for dynamic shared object (DSO) files that should be ignored when examining linkage information.

During the post-build phase, the shared libraries in the newly created package are examined for linkages which are not provided by the package's requirements or a predefined list of system libraries. If such libraries are detected, either a warning `--no-error-overlinking` or error `--error-overlinking` will result.

```
build:
  missing_dso_whitelist:
```

These keys allow additions to the list of allowed libraries.

The `runpath_whitelist` build key is a list of globs for paths which are allowed to appear as runpaths in the package's shared libraries. All other runpaths will cause a warning message to be printed during the build.

```
build:
  runpath_whitelist:
```

4.3.4 Requirements section

Specifies the build and runtime requirements. Dependencies of these requirements are included automatically.

Versions for requirements must follow the conda match specification. See *Package match specifications*.

Build

Tools required to build the package. These packages are run on the build system and include things such as revision control systems (Git, SVN) make tools (GNU make, Autotool, CMake) and compilers (real cross, pseudo-cross, or native when not cross-compiling), and any source pre-processors.

Packages which provide "sysroot" files, like the `CDT` packages (see below) also belong in the build section.

```
requirements:
  build:
    - git
    - cmake
```

Host

This section was added in conda-build 3.0. It represents packages that need to be specific to the target platform when the target platform is not necessarily the same as the native build platform. For example, in order for a recipe to be "cross-capable", shared libraries requirements must be listed in the host section, rather than the build section, so that the shared libraries that get linked are ones for the target platform, rather than the native build platform. You should also include the base interpreter for packages that need one. In other words, a Python package would list `python` here and an R package would list `mro-base` or `r-base`.

```
requirements:
  build:
    - {{ compiler('c') }}
    - {{ cdt('xorg-x11-proto-devel') }}  # [linux]
```

(continues on next page)

(continued from previous page)

```
  host:
    - python
```

Note: When both build and host sections are defined, the build section can be thought of as "build tools" - things that run on the native platform, but output results for the target platform. For example, a cross-compiler that runs on linux-64, but targets linux-armv7.

The PREFIX environment variable points to the host prefix. With respect to activation during builds, both the host and build environments are activated. The build prefix is activated before the host prefix so that the host prefix has priority over the build prefix. Executables that don't exist in the host prefix should be found in the build prefix.

As of conda-build 3.1.4, the build and host prefixes are always separate when both are defined, or when `{{ compiler() }}` Jinja2 functions are used. The only time that build and host are merged is when the host section is absent, and no `{{ compiler() }}` Jinja2 functions are used in meta.yaml. Because these are separate, you may see some build failures when migrating your recipes. For example, let's say you have a recipe to build a Python extension. If you add the compiler Jinja2 functions to the build section, but you do not move your Python dependency from the build section to the host section, your recipe will fail. It will fail because the host environment is where new files are detected, but because you have Python only in the build environment, your extension will be installed into the build environment. No files will be detected. Also, variables such as PYTHON will not be defined when Python is not installed into the host environment.

On Linux, using the compiler packages provided by Anaconda Inc. in the `defaults` meta-channel can prevent your build system leaking into the built software by using our `CDT` (Core Dependency Tree) packages for any "system" dependencies. These packages are repackaged libraries and headers from CentOS6 and are unpacked into the sysroot of our pseudo-cross compilers and are found by them automatically.

Note that what qualifies as a "system" dependency is a matter of opinion. The Anaconda Distribution chose not to provide X11 or GL packages, so we use CDT packages for X11. Conda-forge chose to provide X11 and GL packages.

On macOS, you can also use `{{ compiler() }}` to get compiler packages provided by Anaconda Inc. in the `defaults` meta-channel. The environment variables `MACOSX_DEPLOYMENT_TARGET` and `CONDA_BUILD_SYSROOT` will be set appropriately by conda-build (see *Environment variables*). `CONDA_BUILD_SYSROOT` will specify a folder containing a macOS SDK. These settings achieve backwards compatibility while still providing access to C++14 and C++17. Note that conda-build will set `CONDA_BUILD_SYSROOT` by parsing the `conda_build_config.yaml`. For more details, see *Anaconda compiler tools*.

TL;DR: If you use `{{ compiler() }}` Jinja2 to utilize our new compilers, you must also move anything that is not strictly a build tool into your host dependencies. This includes Python, Python libraries, and any shared libraries that you need to link against in your build. Examples of build tools include any `{{ compiler() }}`, Make, Autoconf, Perl (for running scripts, not installing Perl software), and Python (for running scripts, not for installing software).

Run

Packages required to run the package. These are the dependencies that are installed automatically whenever the package is installed. Package names should follow the package match specifications.

```
requirements:
  run:
    - python
    - argparse # [py26]
    - six >=1.8.0
```

To build a recipe against different versions of NumPy and ensure that each version is part of the package dependencies, list `numpy x.x` as a requirement in `meta.yaml` and use `conda-build` with a NumPy version option such as `--numpy 1.7`.

The line in the `meta.yaml` file should literally say `numpy x.x` and should not have any numbers. If the `meta.yaml` file uses `numpy x.x`, it is required to use the `--numpy` option with `conda-build`.

```
requirements:
  run:
    - python
    - numpy x.x
```

Note: Instead of manually specifying run requirements, since conda-build 3 you can augment the packages used in your build and host sections with *run_exports* which are then automatically added to the run requirements for you.

Run_constrained

Packages that are optional at runtime but must obey the supplied additional constraint if they are installed.

Package names should follow the package match specifications.

```
requirements:
  run_constrained:
    - optional-subpackage ={{ version }}
```

For example, let's say we have an environment that has package "a" installed at version 1.0. If we install package "b" that has a run_constrained entry of "a>1.0", then conda would need to upgrade "a" in the environment in order to install "b".

This is especially useful in the context of virtual packages, where the *run_constrained* dependency is not a package that conda manages, but rather a virtual package that represents a system property that conda can't change. For example, a package on linux may impose a *run_constrained* dependency on *__glibc>=2.12*. This is the version bound consistent with CentOS 6. Software built against glibc 2.12 will be compatible with CentOS 6. This *run_constrained* dependency helps conda tell the user that a given package can't be installed if their system glibc version is too old.

4.3.5 Test section

If this section exists or if there is a `run_test.[py,pl,sh,bat]` file in the recipe, the package is installed into a test environment after the build is finished and the tests are run there.

Test files

Test files that are copied from the recipe into the temporary test directory and are needed during testing.

```
test:
  files:
    - test-data.txt
```

Source files

Test files that are copied from the source work directory into the temporary test directory and are needed during testing.

```
test:
  source_files:
    - test-data.txt
    - some/directory
    - some/directory/pattern*.sh
```

This capability was added in conda-build 2.0.

Test requirements

In addition to the runtime requirements, you can specify requirements needed during testing. The runtime requirements that you specified in the "run" section described above are automatically included during testing.

```
test:
  requires:
    - nose
```

Test commands

Commands that are run as part of the test.

```
test:
  commands:
    - bsdiff4 -h
    - bspatch4 -h
```

Python imports

List of Python modules or packages that will be imported in the test environment.

```
test:
  imports:
    - bsdiff4
```

This would be equivalent to having a `run_test.py` with the following:

```
import bsdiff4
```

Run test script

The script `run_test.sh`---or `.bat`, `.py`, or `.pl`---is run automatically if it is part of the recipe.

Note: Python .py and Perl .pl scripts are valid only as part of Python and Perl packages, respectively.

Downstream tests

Knowing that your software built and ran its tests successfully is necessary, but not sufficient, for keeping whole systems of software running. To have confidence that a new build of a package hasn't broken other downstream software, conda-build supports the notion of downstream testing.

```
test:
  downstreams:
    - some_downstream_pkg
```

This is saying "When I build this recipe, after you run my test suite here, also download and run some_downstream_pkg which depends on my package." Conda-build takes care of ensuring that the package you just built gets installed into the environment for testing some_downstream_pkg. If conda-build can't create that environment due to unsatisfiable dependencies, it will skip those downstream tests and warn you. This usually happens when you are building a new version of a package that will require you to rebuild the downstream dependencies.

Downstreams specs are full conda specs, similar to the requirements section. You can put version constraints on your specs in here:

```
test:
  downstreams:
    - some_downstream_pkg  >=2.0
```

More than one package can be specified to run downstream tests for:

```
test:
  downstreams:
    - some_downstream_pkg
    - other_downstream_pkg
```

However, this does not mean that these packages are tested together. Rather, each of these are tested for satisfiability with your new package, then each of their test suites are run separately with the new package.

4.3.6 Outputs section

Explicitly specifies packaging steps. This section supports multiple outputs, as well as different package output types. The format is a list of mappings. Build strings for subpackages are determined by their runtime dependencies. This support was added in conda-build 2.1.0.

```
outputs:
  - name: some-subpackage
    version: 1.0
  - name: some-other-subpackage
    version: 2.0
```

Note: If any output is specified in the outputs section, the default packaging behavior of conda-build is bypassed. In other words, if any subpackage is specified, then you do not get the normal top-level build for this recipe without explicitly defining a subpackage for it. This is an alternative to the existing behavior, not an addition to it. For more information, see *Implicit metapackages*. Each output may have its own version and requirements. Additionally, subpackages may impose downstream pinning similarly to *Pin downstream* to help keep your packages aligned.

Specifying files to include in output

You can specify files to be included in the package in 1 of 2 ways:

- Explicit file lists.
- Scripts that move files into the build prefix.

Explicit file lists are relative paths from the root of the build prefix. Explicit file lists support glob expressions. Directory names are also supported, and they recursively include contents.

```
outputs:
  - name: subpackage-name
    files:
      - a-file
      - a-folder
      - *.some-extension
      - somefolder/*.some-extension
```

Scripts that create or move files into the build prefix can be any kind of script. Known script types need only specify the script name. Currently the list of recognized extensions is py, bat, ps1, and sh.

```
outputs:
  - name: subpackage-name
    script: move-files.py
```

The interpreter command must be specified if the file extension is not recognized.

```
outputs:
  - name: subpackage-name
    script: some-script.extension
    script_interpreter: program plus arguments to run script
```

For scripts that move or create files, a fresh copy of the working directory is provided at the start of each script execution. This ensures that results between scripts are independent of one another.

Note: For either the file list or the script approach, having more than 1 package contain a given file is not explicitly forbidden, but may prevent installation of both packages simultaneously. Conda disallows this condition because it creates ambiguous runtime conditions.

Subpackage requirements

Like a top-level recipe, a subpackage may have zero or more dependencies listed as build requirements and zero or more dependencies listed as run requirements.

The dependencies listed as subpackage build requirements are available only during the packaging phase of that subpackage.

A subpackage does not automatically inherit any dependencies from its top-level recipe, so any build or run requirements needed by the subpackage must be explicitly specified.

```
outputs:

  - name: subpackage-name
    requirements:
      build:
        - some-dep
      run:
        - some-dep
```

It is also possible for a subpackage requirements section to have a list of dependencies, but no build section or run section. This is the same as having a build section with this dependency list and a run section with the same dependency list.

```
outputs:
  - name: subpackage-name
    requirements:
      - some-dep
```

You can also impose runtime dependencies whenever a given (sub)package is installed as a build dependency. For example, if we had an overarching "compilers" package, and within that, had `gcc` and `libgcc` outputs, we could force recipes that use GCC to include a matching libgcc runtime requirement:

```
outputs:
  - name: gcc
    run_exports:
      - libgcc 2.*
  - name: libgcc
```

See the *Export runtime requirements* section for additional information.

Note: Variant expressions are very powerful here. You can express the version requirement in the `run_exports` entry as a Jinja function to insert values based on the actual version of libgcc produced by the recipe. Read more about them at *Referencing subpackages*.

Implicit metapackages

When viewing the top-level package as a collection of smaller subpackages, it may be convenient to define the top-level package as a composition of several subpackages. If you do this and you do not define a subpackage name that matches the top-level package/name, conda-build creates a metapackage for you. This metapackage has runtime requirements drawn from its dependency subpackages, for the sake of accurate build strings.

EXAMPLE: In this example, a metapackage for `subpackage-example` will be created. It will have runtime dependencies on `subpackage1`, `subpackage2`, `some-dep`, and `some-other-dep`.

```
package:
  name: subpackage-example
  version: 1.0

requirements:
  run:
    - subpackage1
    - subpackage2

outputs:
  - name: subpackage1
    requirements:
      - some-dep
  - name: subpackage2
    requirements:
      - some-other-dep
  - name: subpackage3
    requirements:
      - some-totally-exotic-dep
```

Subpackage tests

You can test subpackages independently of the top-level package. Independent test script files for each separate package are specified under the subpackage's test section. These files support the same formats as the top-level `run_test.*` scripts, which are .py, .pl, .bat, and .sh. These may be extended to support other script types in the future.

```
outputs:
  - name: subpackage-name
    test:
      script: some-other-script.py
```

By default, the `run_test.*` scripts apply only to the top-level package. To apply them also to subpackages, list them explicitly in the script section:

```
outputs:
  - name: subpackage-name
    test:
      script: run_test.py
```

Test requirements for subpackages are not supported. Instead, subpackage tests install their runtime requirements---but not the run requirements for the top-level package---and the test-time requirements of the top-level package.

EXAMPLE: In this example, the test for `subpackage-name` installs `some-test-dep` and `subpackage-run-req`, but not `some-top-level-run-req`.

```yaml
requirements:
  run:
    - some-top-level-run-req

test:
  requires:
    - some-test-dep

outputs:
  - name: subpackage-name
    requirements:
      - subpackage-run-req
    test:
      script: run_test.py
```

Output type

Conda-build supports creating packages other than conda packages. Currently that support includes only wheels, but others may come as demand appears. If type is not specified, the default value is `conda`.

```yaml
requirements:
  build:
    - wheel

outputs:
  - name: name-of-wheel-package
    type: wheel
```

Currently you must include the wheel package in your top-level requirements/build section in order to build wheels.

When specifying type, the name field is optional and it defaults to the package/name field for the top-level recipe.

```yaml
requirements:
  build:
    - wheel

outputs:
  - type: wheel
```

Conda-build currently knows how to test only conda packages. Conda-build does support using Twine to upload packages to PyPI. See the conda-build help output (`conda-build --help`) for the list of arguments accepted that will be passed through to Twine.

Note: You must use pip to install Twine in order for this to work.

4.3.7 About section

Specifies identifying information about the package. The information displays in the Anaconda.org channel.

```
about:
  home: https://github.com/ilanschnell/bsdiff4
  license: BSD
  license_file: LICENSE
  summary: binary diff and patch using the BSDIFF4-format
```

License file

Add a file containing the software license to the package metadata. Many licenses require the license statement to be distributed with the package. The filename is relative to the source or recipe directory. The value can be a single filename or a YAML list for multiple license files. Values can also point to directories with license information. Directory entries must end with a / suffix (this is to lessen unintentional inclusion of non-license files; all of the directory's contents will be unconditionally and recursively added).

```
about:
  license_file:
    - LICENSE
    - vendor-licenses/
```

Prelink Message File

Similar to the license file, the user can add prelink message files to the conda package.

```
about:
  prelink_message:
    - prelink_message_file.txt
    - folder-with-all-prelink-messages/
```

4.3.8 App section

If the app section is present, the package is an app, meaning that it appears in Anaconda Navigator.

Entry point

The command that is called to launch the app in Navigator.

```
app:
  entry: ipython notebook
```

Icon file

The icon file contained in the recipe.

```
app:
  icon: icon_64x64.png
```

Summary

Summary of the package used in Navigator.

```
app:
  summary:  "The Jupyter Notebook"
```

Own environment

If `True`, installing the app through Navigator installs into its own environment. The default is `False`.

```
app:
  own_environment: True
```

4.3.9 Extra section

A schema-free area for storing non-conda-specific metadata in standard YAML form.

EXAMPLE: To store recipe maintainer information:

```
extra:
  maintainers:
   - name of maintainer
```

4.3.10 Templating with Jinja

Conda-build supports Jinja templating in the `meta.yaml` file.

EXAMPLE: The following `meta.yaml` would work with the GIT values defined for Git repositores. The recipe is included at the base directory of the Git repository, so the `git_url` is `../`:

```
package:
  name: mypkg
  version: {{ GIT_DESCRIBE_TAG }}

build:
  number: {{ GIT_DESCRIBE_NUMBER }}

  # Note that this will override the default build string with the Python
  # and NumPy versions
  string: {{ GIT_BUILD_STR }}

source:
  git_url: ../
```

Conda-build checks if the Jinja2 variables that you use are defined and produces a clear error if it is not.

You can also use a different syntax for these environment variables that allows default values to be set, although it is somewhat more verbose.

EXAMPLE: A version of the previous example using the syntax that allows defaults:

```
package:
  name: mypkg
  version: {{ environ.get('GIT_DESCRIBE_TAG', '') }}

build:
  number: {{ environ.get('GIT_DESCRIBE_NUMBER', 0) }}

  # Note that this will override the default build string with the Python
  # and NumPy versions
  string: {{ environ.get('GIT_BUILD_STR', '') }}

source:
  git_url: ../
```

One further possibility using templating is obtaining data from your downloaded source code.

EXAMPLE: To process a project's `setup.py` and obtain the version and other metadata:

```
{% set data = load_setup_py_data() %}

package:
  name: conda-build-test-source-setup-py-data
  version: {{ data.get('version') }}

# source will be downloaded prior to filling in jinja templates
# Example assumes that this folder has setup.py in it
source:
  path_url: ../
```

These functions are completely compatible with any other variables such as Git and Mercurial.

Extending this arbitrarily to other functions requires that functions be predefined before Jinja processing, which in practice means changing the conda-build source code. See the conda-build issue tracker.

For more information, see the Jinja2 template documentation and *the list of available environment variables*.

Jinja templates are evaluated during the build process. To retrieve a fully rendered `meta.yaml`, use the *conda render* command.

Conda-build specific Jinja2 functions

Besides the default Jinja2 functionality, additional Jinja functions are available during the conda-build process: `pin_compatible`, `pin_subpackage`, `compiler`, and `resolved_packages`. Please see *Extra Jinja2 functions* for the definition of the first 3 functions. Definition of `resolved_packages` is given below:

- `resolved_packages('environment_name')`: Returns the final list of packages (in the form of `package_name version build_string`) that are listed in `requirements:host` or `requirements:build`. This includes all packages (including the indirect dependencies) that will be installed in the host or build environment. `environment_name` must be either `host` or `build`. This function is useful for creating meta-packages that will want to pin all of their *direct* and *indirect* dependencies to their exact match. For example:

```
requirements:
  host:
    - curl 7.55.1
  run:
  {% for package in resolved_packages('host') %}
    - {{ package }}
  {% endfor %}
```

might render to (depending on package dependencies and the platform):

```
requirements:
    host:
        - curl 7.55.1
    run:
        - ca-certificates 2017.08.26 h1d4fec5_0
        - curl 7.55.1 h78862de_4
        - libgcc-ng 7.2.0 h7cc24e2_2
        - libssh2 1.8.0 h9cfc8f7_4
        - openssl 1.0.2n hb7f436b_0
        - zlib 1.2.11 ha838bed_2
```

Here, output of `resolved_packages` was:

```
['ca-certificates 2017.08.26 h1d4fec5_0', 'curl 7.55.1 h78862de_4',
'libgcc-ng 7.2.0 h7cc24e2_2', 'libssh2 1.8.0 h9cfc8f7_4',
'openssl 1.0.2n hb7f436b_0', 'zlib 1.2.11 ha838bed_2']
```

4.3.11 Preprocessing selectors

You can add selectors to any line, which are used as part of a preprocessing stage. Before the `meta.yaml` file is read, each selector is evaluated and if it is `False`, the line that it is on is removed. A selector has the form `# [<selector>]` at the end of a line.

```
source:
  url: http://path/to/unix/source    # [not win]
  url: http://path/to/windows/source # [win]
```

Note: Preprocessing selectors are evaluated after Jinja templates.

A selector is a valid Python statement that is executed. The following variables are defined. Unless otherwise stated, the variables are booleans.

x86	True if the system architecture is x86, both 32-bit and 64-bit, for Intel or AMD chips.
x86_64	True if the system architecture is x86_64, which is 64-bit, for Intel or AMD chips.
linux	True if the platform is Linux.
linux32	True if the platform is Linux and the Python architecture is 32-bit.
linux64	True if the platform is Linux and the Python architecture is 64-bit.
armv6l	True if the platform is Linux and the Python architecture is armv6l.
armv7l	True if the platform is Linux and the Python architecture is armv7l.
aarch64	True if the platform is Linux and the Python architecture is aarch64.
ppc64le	True if the platform is Linux and the Python architecture is ppc64le.
osx	True if the platform is macOS.
arm64	True if the platform is macOS and the Python architecture is arm64.
unix	True if the platform is either macOS or Linux.
win	True if the platform is Windows.
win32	True if the platform is Windows and the Python architecture is 32-bit.
win64	True if the platform is Windows and the Python architecture is 64-bit.
py	The Python version as an int, such as `27` or `36`. See the CONDA_PY *environment variable*.
py3k	True if the Python major version is 3.
py2k	True if the Python major version is 2.
py27	True if the Python version is 2.7. Use of this selector is discouraged in favor of comparison operators (e.g. py==27).
py34	True if the Python version is 3.4. Use of this selector is discouraged in favor of comparison operators (e.g. py==34).
py35	True if the Python version is 3.5. Use of this selector is discouraged in favor of comparison operators (e.g. py==35).
py36	True if the Python version is 3.6. Use of this selector is discouraged in favor of comparison operators (e.g. py==36).
np	The NumPy version as an integer such as `111`. See the CONDA_NPY *environment variable*.
build_platform	The native subdir of the conda executable

The use of the Python version selectors, *py27*, *py34*, etc. is discouraged in favor of the more general comparison operators. Additional selectors in this series will not be added to conda-build.

Because the selector is any valid Python expression, complicated logic is possible:

```
source:
  url: http://path/to/windows/source      # [win]
  url: http://path/to/python2/unix/source # [unix and py2k]
  url: http://path/to/python3/unix/source # [unix and py>=35]
```

Note: The selectors delete only the line that they are on, so you may need to put the same selector on multiple lines:

```
source:
  url: http://path/to/windows/source     # [win]
  md5: 30fbf531409a18a48b1be249052e242a  # [win]
  url: http://path/to/unix/source        # [unix]
  md5: 88510902197cba0d1ab4791e0f41a66e  # [unix]
```

Note: To select multiple operating systems use the `or` statement. While it might be tempting to use `skip: True # [win and osx]`, this will only work if the platform is both windows and osx simultaneously (i.e. never).

```
build:
   skip: True  # [win or osx]
```

4.4 Adding pre-link, post-link, and pre-unlink scripts

You can add scripts to a recipe. They must be located in the same directory as the meta.yaml file. The following scripts can be added:

- `pre-link`---Executed before the package is installed. An error is indicated by a nonzero exit and causes conda to stop and causes the installation to fail.
- `post-link`---Executed after the package is installed. An error is indicated by a nonzero exist and causes installation to fail. If there is an error, conda does not write any package metadata.
- `pre-unlink`---Executed before the package is removed. An error is indicated by a nonzero exist and causes the removal to fail.

In addition to being co-located with the meta.yaml file, they must be named simply `post-link.sh` or `post-link.bat`. Conda-build will rename them to .<name>-<action>.sh (or .bat) where <name> is the package name and <action> is one of the preceeding actions.

These scripts are executed in a subprocess by conda, using `%COMSPEC% /c <script>` on Windows and `/bin/bash <script>` on macOS and Linux.

The convention for the path and filenames of these scripts on Windows is:

```
Scripts/.<name>-<action>.bat
```

On Linux and macOS the convention is:

```
bin/.<name>-<action>.sh
```

The scripts set the following environment variables:

PREFIX	The install prefix.
PKG_NAME	The name of the package.
PKG_VERSION	The version of the package.
PKG_BUILDNUM	The build number of the package.

The scripts are:

- Windows:
  - `pre-link.bat`
  - `post-link.bat`
  - `pre-unlink.bat`
- macOS and Linux:
  - `pre-link.sh`
  - `post-link.sh`
  - `pre-unlink.sh`

Post-link and pre-unlink scripts should:

- Be avoided whenever possible.
- Not touch anything other than the files being installed.
- Not write anything to stdout or stderr, unless an error occurs.
- Not depend on any installed or to-be-installed conda packages.
- Depend only on simple system tools such as `rm`, `cp`, `mv`, and `ln`.

The scripts should not write to stdout or stderr unless an error occurs, but they may write to `$PREFIX/.messages.txt`, which is shown after conda completes all actions.

4.5 Activate scripts

Recipes are allowed to have activate scripts which will be sourced or called when the environment is activated. It is generally recommended to avoid using activate scripts when another option is possible because people do not always activate environments the expected way and these packages may then misbehave.

When using them in a recipe, feel free to name them activate.bat, activate.sh, deactivate.bat, and deactivate.sh in the recipe. The installed scripts are recommended to be prefixed by the package name and a separating -.

Below is some sample code for Unix and Windows that will make this install process easier.

In `build.sh`:

```
# Copy the [de]activate scripts to $PREFIX/etc/conda/[de]activate.d.
# This will allow them to be run on environment activation.
for CHANGE in "activate" "deactivate"
do
    mkdir -p "${PREFIX}/etc/conda/${CHANGE}.d"
    cp "${RECIPE_DIR}/${CHANGE}.sh" "${PREFIX}/etc/conda/${CHANGE}.d/${PKG_NAME}_$
↪{CHANGE}.sh"
done
```

In `build.bat`:

```
setlocal EnableDelayedExpansion

:: Copy the [de]activate scripts to %PREFIX%\etc\conda\[de]activate.d.
:: This will allow them to be run on environment activation.
for %%F in (activate deactivate) DO (
    if not exist %PREFIX%\etc\conda\%%F.d mkdir %PREFIX%\etc\conda\%%F.d
    copy %RECIPE_DIR%\%%F.bat %PREFIX%\etc\conda\%%F.d\%PKG_NAME%_%%F.bat
    :: Copy unix shell activation scripts, needed by Windows Bash users
    copy %RECIPE_DIR%\%%F.sh %PREFIX%\etc\conda\%%F.d\%PKG_NAME%_%%F.sh
)
```

4.6 Making packages relocatable

Often, the most difficult thing about building a conda package is making it relocatable. Relocatable means that the package can be installed into any prefix. Otherwise, the package would be usable only in the environment in which it was built.

Conda-build does the following things automatically to make packages relocatable:

- Binary object files are converted to use relative paths using `install_name_tool` on macOS and patchelf on Linux.
- Any text file without NULL bytes that contains the build prefix or the placeholder prefix `/opt/anaconda1anaconda2anaconda3` is registered in the `info/has_prefix` file in the package metadata. When conda installs the package, any files in `info/has_prefix` have the registered prefix replaced with the install prefix. For more information, see *Package metadata*.
- Any binary file containing the build prefix can automatically be registered in `info/has_prefix` using `build/detect_binary_files_with_prefix` in `meta.yaml`. Alternatively, individual binary files can be registered by listing them in `build/binary_has_prefix_files` in `meta.yaml`. The registered files will have their build prefix replaced with the install prefix at install time. This works by padding the install prefix with null terminators, such that the length of the binary file remains the same. The build prefix must therefore be long enough to accommodate any reasonable installation prefix. On macOS and Linux, conda-build pads the build prefix to 255 characters by appending `_placehold`s to the end of the build directory name.

Note: The prefix length was changed in conda-build 2.0 from 80 characters to 255 characters. Legacy packages with 80-character prefixes must be rebuilt to take advantage of the longer prefix.

- There may be cases where conda identified a file as binary, but it needs to have the build prefix replaced as if it were text---no padding with null terminators. Such files can be listed in `build/has_prefix_files` in `meta.yaml`.

4.7 Conda package specification

- *Package metadata*
- *Link and unlink scripts*
- *Repository structure and index*
- *Package match specifications*

A conda package is a bzipped tar archive---.tar.bz2---that contains:

- Metadata under the `info/` directory.
- A collection of files that are installed directly into an install prefix.

The format is identical across platforms and operating systems. During the install process, all files are extracted into the install prefix, with the exception of the ones in `info/`. Installing a conda package into an environment is similar to executing the following commands:

```
cd <environment prefix>
tar xjf some-package-1.0-0.tar.bz2
```

Only files, including symbolic links, are part of a conda package. Directories are not included. Directories are created and removed as needed, but you cannot create an empty directory from the tar archive directly.

4.7.1 Package metadata

The `info/` directory contains all metadata about a package. Files in this location are not installed under the install prefix. Although you are free to add any file to this directory, conda only inspects the content of the files discussed below.

info/index.json

This file contains basic information about the package, such as name, version, build string, and dependencies. The content of this file is stored in `repodata.json`, which is the repository index file, hence the name `index.json`. The JSON object is a dictionary containing the keys shown below. The filename of the conda package is composed of the first 3 values, as in: `<name>-<version>-<build>.tar.bz2`.

Key	Type	Description
name	string	The lowercase name of the package. May contain the "-" character.
version	string	The package version. May not contain "-". Conda acknowledges PEP 440.
build	string	The build string. May not contain "-". Differentiates builds of packages with otherwise identical names and versions, such as: • A build with other dependencies, such as Python 3.4 instead of Python 2.7. • A bug fix in the build process. • Some different optional dependencies, such as MKL versus ATLAS linkage. Nothing in conda actually inspects the build string. Strings such as `np18py34_1` are designed only for human readability and conda never parses them.
build_number	integer	A non-negative integer representing the build number of the package. Unlike the build string, the `build_number` is inspected by conda. Conda uses it to sort packages that have otherwise identical names and versions to determine the latest one. This is important because new builds that contain bug fixes for the way a package is built may be added to a repository.
depends	list of strings	A list of dependency specifications, where each element is a string, as outlined in *Package match specifications*.
arch	string	Optional. The architecture the package is built for. EXAMPLE: `x86_64` Conda currently does not use this key.
platform	string	Optional. The OS that the package is built for. EXAMPLE: `osx` Conda currently does not use this key. Packages for a specific architecture and platform are usually distinguished by the repository subdirectory that contains them---see *Repository structure and index*.

info/files

Lists all files that are part of the package itself, 1 per line. All of these files need to get linked into the environment. Any files in the package that are not listed in this file are not linked when the package is installed. The directory delimiter for the files in `info/files` should always be "/", even on Windows. This matches the directory delimiter used in the tarball.

info/has_prefix

Optional file. Lists all files that contain a hard-coded build prefix or placeholder prefix, which needs to be replaced by the install prefix at installation time.

Note: Due to the way the binary replacement works, the placeholder prefix must be longer than the install prefix.

Each line of this file should be either a path, in which case it is considered a text file with the default placeholder `/opt/anaconda1anaconda2anaconda3`, or a space-separated list of placeholder, mode, and path, where:

- Placeholder is the build or placeholder prefix.
- Mode is either `text` or `binary`.
- Path is the relative path of the file to be updated.

EXAMPLE: On Windows:

```
"Scripts/script1.py"
"C:\Users\username\anaconda\envs\_build" text "Scripts/script2.bat"
"C:/Users/username/anaconda/envs/_build" binary "Scripts/binary"
```

EXAMPLE: On macOS or Linux:

```
bin/script.sh
/Users/username/anaconda/envs/_build binary bin/binary
/Users/username/anaconda/envs/_build text share/text
```

Note: The directory delimiter for the relative path must always be "/", even on Windows. The placeholder may contain either "\" or "/" on Windows, but the replacement prefix will match the delimiter used in the placeholder. The default placeholder `/opt/anaconda1anaconda2anaconda3` is an exception, being replaced with the install prefix using the native path delimiter. On Windows, the placeholder and path always appear in quotes to support paths with spaces.

---

info/license.txt

Optional file. The software license for the package.

info/no_link

Optional file. Lists all files that cannot be linked---either soft-linked or hard-linked---into environments and are copied instead.

info/about.json

Optional file. Contains the entries in the *About section* of the `meta.yaml` file. The following keys are added to `info/about.json` if present in the build recipe:

- home.
- dev_url.
- doc_url.
- license_url.
- license.
- summary.
- description.
- license_family.

info/recipe

A directory containing the full contents of the build recipe.

meta.yaml.rendered

The fully rendered build recipe. See *conda render*.

This directory is present only when the the `include_recipe` flag is `True` in the *Build section*.

4.7.2 Link and unlink scripts

You may optionally execute scripts before and after the link and unlink steps. For more information, see *Adding pre-link, post-link, and pre-unlink scripts*.

4.7.3 Repository structure and index

A conda repository---or channel---is a directory tree, usually served over HTTPS, which has platform subdirectories, each of which contains conda packages and a repository index. The index file `repodata.json` lists all conda packages in the platform subdirectory. Use `conda index` to create such an index from the conda packages within a directory. It is simple mapping of the full conda package filename to the dictionary object in `info/index.json` described in *Adding pre-link, post-link, and pre-unlink scripts*.

In the following example, a repository provides the conda package `misc-1.0-np17py27_0.tar.bz2` on 64-bit Linux and 32-bit Windows:

```
<some path>/linux-64/repodata.json
                    repodata.json.bz2
                    misc-1.0-np17py27_0.tar.bz2
          /win-32/repodata.json
                  repodata.json.bz2
                  misc-1.0-np17py27_0.tar.bz2
```

Note: Both conda packages have identical filenames and are distinguished only by the repository subdirectory that contains them.

4.7.4 Package match specifications

This match specification is not the same as the syntax used at the command line with `conda install`, such as `conda install python=3.4`. Internally, conda translates the command line syntax to the spec defined in this section.

EXAMPLE: python=3.4 is translated to python 3.4*.

Package dependencies are specified using a match specification. A match specification is a space-separated string of 1, 2, or 3 parts:

- The first part is always the exact name of the package.
- The second part refers to the version and may contain special characters:
  - | means OR.

    EXAMPLE: `1.0|1.2` matches version 1.0 or 1.2.
  - * matches 0 or more characters in the version string. In terms of regular expressions, it is the same as `r'.*'`.

    EXAMPLE: 1.0|1.4* matches 1.0, 1.4 and 1.4.1b2, but not 1.2.
  - <, >, <=, >=, ==, and != are relational operators on versions, which are compared using PEP-440. For example, <=`1.0` matches `0.9`, `0.9.1`, and `1.0`, but not `1.0.1`. == and != are exact equality.

    Pre-release versioning is also supported such that >`1.0b4` will match `1.0b5` and `1.0rc1` but not `1.0b4` or `1.0a5`.

    EXAMPLE: <=1.0 matches 0.9, 0.9.1, and 1.0, but not 1.0.1.
  - , means AND.

    EXAMPLE: >=2,<3 matches all packages in the 2 series. 2.0, 2.1, and 2.9 all match, but 3.0 and 1.0 do not.
  - , has higher precedence than |, so >=1,<2|>3 means greater than or equal to 1 AND less than 2 or greater than 3, which matches 1, 1.3 and 3.0, but not 2.2.

  Conda parses the version by splitting it into parts separated by |. If the part begins with <, >, =, or !, it is parsed as a relational operator. Otherwise, it is parsed as a version, possibly containing the "*" operator.
- The third part is always the exact build string. When there are 3 parts, the second part must be the exact version.

Remember that the version specification cannot contain spaces, as spaces are used to delimit the package, version, and build string in the whole match specification. `python >= 2.7` is an invalid match specification. Furthermore, `python>=2.7` is matched as any version of a package named `python>=2.7`.

When using the command line, put double quotes around any package version specification that contains the space character or any of the following characters: <, >, *, or |.

EXAMPLE:

```
conda install numpy=1.11
conda install numpy==1.11
conda install "numpy>1.11"
conda install "numpy=1.11.1|1.11.3"
conda install "numpy>=1.8,<2"
```

Examples

The OR constraint "numpy=1.11.1|1.11.3" matches with 1.11.1 or 1.11.3.

The AND constraint "numpy>=1.8,<2" matches with 1.8 and 1.9 but not 2.0.

The fuzzy constraint numpy=1.11 matches 1.11, 1.11.0, 1.11.1, 1.11.2, 1.11.18, and so on.

The exact constraint numpy==1.11 matches 1.11, 1.11.0, 1.11.0.0, and so on.

The build string constraint "numpy=1.11.2=*nomkl*" matches the NumPy 1.11.2 packages without MKL but not the normal MKL NumPy 1.11.2 packages.

The build string constraint "numpy=1.11.1|1.11.3=py36_0" matches NumPy 1.11.1 or 1.11.3 built for Python 3.6 but not any versions of NumPy built for Python 3.5 or Python 2.7.

The following are all valid match specifications for numpy-1.8.1-py27_0:

- numpy
- numpy 1.8*
- numpy 1.8.1
- numpy >=1.8
- numpy ==1.8.1
- numpy 1.8|1.8*
- numpy >=1.8,<2
- numpy >=1.8,<2|1.9
- numpy 1.8.1 py27_0
- numpy=1.8.1=py27_0

4.8 Using shared libraries

Shared libraries are libraries that are loosely coupled to the programs and extensions that depend on them. When loading an executable into memory, an operating system finds all dependent shared libraries and links them to the executable so that it can run.

Windows, macOS, and Linux all provide a way to build executables and libraries that contain links to the shared libraries they depend on, instead of directly linking the libraries themselves.

4.8.1 Shared libraries in Windows

Unlike macOS and Linux, Windows does not have the concept of embedding links into binaries. Instead, Windows depends primarily on searching directories for matching filenames, as documented in Search Path Used by Windows to Locate a DLL.

There is an alternate configuration, called side-by-side assemblies, that requires specification of DLL versions in either an embedded manifest or an appropriately named XML file alongside the binary in question. Conda does not currently use side-by-side assemblies, but it may turn towards that in the future to resolve complications with multiple versions of the same library on the same system.

For now, most DLLs are installed into `(install prefix)\\Library\\bin`. This path is added to `os.environ["PATH"]` for all Python processes, so that DLLs can be located, regardless of the value of the system's PATH environment variable.

Note: PATH is searched from left to right, with the first DLL name match being picked up, in the absence of a manifest specifying otherwise. This means that installing software with other matching DLLs may give you a system that crashes in unpredictable ways. When troubleshooting or asking for support on Windows, always consider PATH as a potential source of issues.

4.8.2 Shared libraries in macOS and Linux

In macOS and Linux, dynamic links are discovered in a similar manner to the way that Python modules are discovered via PYTHONPATH, and executables are discovered via PATH. A list of search locations is made, and then the library objects are searched for in the search locations. By default, as well as by design, the system dynamic linker does not have any special preference for the conda environment `lib` directories.

You can specify both absolute links and relative links. If the links are absolute paths, such as `/Users/UserName/my_build_env`, the library works only on a system where that exact path exists. Therefore, relative links are preferred in conda packages.

Relative links require a special variable in the link itself:

- On Linux, the $ORIGIN variable allows you to specify "relative to this file as it is being executed".
- On macOS, the variables are:
  - @rpath---Allows you to set relative links from the system load paths.
  - @loader_path---Equivalent to $ORIGIN.
  - @executable_path---Supports the Apple `.app` directory approach, where libraries know where they live relative to their calling application.

Conda-build uses @loader_path on macOS and $ORIGIN on Linux because we install into a common root directory and can assume that other libraries are also installed into that root. The use of the variables allows you to build relocatable binaries that can be built on one system and sent everywhere.

On Linux, `conda-build` modifies any shared libraries or generated executables to use a relative dynamic link by calling the patchelf tool. On macOS, the install_name_tool tool is used.

Warning: Setting LD_LIBRARY_PATH on Linux or DYLD_LIBRARY_PATH on macOS can interfere with this because the dynamic linker short-circuits link resolution by first looking at LD_LIBRARY_PATH.

EXAMPLE: You install an old version of libcurl into your conda environment due to some compatibility issues with the code you're using. Then, you set `export LD_LIBRARY_PATH=/home/UserName/envs/curl_env/lib`. From

that point on, every program that you execute in that session will favor this libcurl to your system libcurl because it is now effectively at the "front" of the dynamic load path.

Including conda environment paths in LD_LIBRARY_PATH or DYLD_LIBRARY_PATH is not recommended.

4.9 Build variants

The nature of binary compatibility (and incompatibility) means that we sometimes need to build binary packages (and any package containing binaries) with several variants to support different usage environments. For example, using NumPy's C API means that a package must be used with the same version of NumPy at runtime that was used at build time.

There has been limited support for this for a long time. Including Python in both build and run requirements resulted in a package with Python pinned to the version of Python used at build time, and a corresponding addition to the filename such as "py27". Similar support existed for NumPy with the addition of an `x.x` pin in the recipe after Conda-build PR 573 was merged. Before conda-build version 3.0, there were also many longstanding proposals for general support (Conda-build issue 1142).

As of conda-build 3.0, a new configuration scheme has been added, dubbed "variants." Conceptually, this decouples pinning values from recipes, replacing them with Jinja2 template variables. It adds support for the notion of "compatible" pinnings to be integrated with ABI compatibility databases, such as ABI Laboratory. Note that the concept of "compatible" pinnings is currently still under heavy development.

Variant input is ultimately a dictionary. These dictionaries are mostly very flat. Keys are made directly available in Jinja2 templates. As a result, keys in the dictionary (and in files read into dictionaries) must be valid jinja2 variable names (no - characters allowed). This example builds Python 2.7 and 3.5 packages in one build command:

conda_build_config.yaml like:

```
python:
    - 2.7
    - 3.5
```

meta.yaml contents like:

```
package:
    name: compiled-code
    version: 1.0

requirements:
    build:
        - python {{ python }}
    run:
        - python
```

The command to build recipes is unchanged relative to earlier conda-build versions. For example, with our shell in the same folder as meta.yaml and conda_build_config.yaml, we just call the `conda build .` command.

4.9.1 General pinning examples

There are a few characteristic use cases for pinning. Please consider this a map for the content below.

1. Shared library providing a binary interface. All uses of this library use the binary interface. It is convenient to apply the same pin to all of your builds. Example: boost

   conda_build_config.yaml in your HOME folder:

```
boost:
  - 1.61
  - 1.63
pin_run_as_build:
  boost: x.x
```

   meta.yaml:

```
package:
    name: compiled-code
    version: 1.0

requirements:
    build:
        - boost  {{ boost }}
    run:
        - boost
```

   This example demonstrates several features:

   - User-wide configuration with a specifically named config file (conda_build_config.yaml in your home folder). More options below in *Creating conda-build variant config files*.
   - Building against multiple versions of a single library (set versions installed at build time).
   - Pinning runtime requirements to the version used at build time. More information below at *Pinning at the variant level*.
   - Specify granularity of pinning. `x.x` pins major and minor version. More information at *Pinning expressions*.

2. Python package with externally accessible binary component. Not all uses of this library use the binary interface (some only use pure Python). Example: NumPy.

   conda_build_config.yaml in your recipe folder (alongside meta.yaml):

```
numpy:
  - 1.11
  - 1.12
```

   meta.yaml:

```
package:
    name: numpy_using_pythonAPI_thing
    version: 1.0

requirements:
    build:
        - python
```

(continues on next page)

(continued from previous page)

```
        - numpy
    run:
        - python
        - numpy
```

This example demonstrates a particular feature: reduction of builds when pins are unnecessary. Since the example recipe above only requires the Python API to NumPy, we will only build the package once and the version of NumPy will not be pinned at runtime to match the compile-time version. There's more information at *Avoiding unnecessary builds*.

For a different package that makes use of the NumPy C API, we will need to actually pin NumPy in this recipe (and only in this recipe, so that other recipes don't unnecessarily build lots of variants). To pin NumPy, you can use the variant key directly in meta.yaml:

```
package:
    name: numpy_using_cAPI_thing
    version: 1.0

requirements:
    build:
        - numpy  {{ numpy }}
    run:
        - numpy  {{ numpy }}
```

For legacy compatibility, Python is pinned implicitly without specifying `{{ python }}` in your recipe. This is generally intractable to extend to all package names, so in general, try to get in the habit of always using the Jinja2 variable substitution for pinning using versions from your conda_build_config.yaml file.

There are also more flexible ways to pin using the *Pinning expressions*. See *Pinning at the recipe level* for examples.

3. One recipe splits into multiple packages, and package dependencies need to be dynamically pinned among one another. Example: GCC/libgcc/libstdc++/gfortran/etc.

   The dynamic pinning is the tricky part. Conda-build provides new ways to refer to other subpackages within a single recipe.

```
package:
    name: dynamic_supackage
    version: 1.0

requirements:
    run:
        - {{ pin_subpackage('my_awesome_subpackage') }}

outputs:
  - name: my_awesome_subpackage
    version: 2.0
```

By referring to subpackages this way, you don't need to worry about what the end version of `my_awesome_subpackage` will be. Update it independently and just let conda-build figure it out and keep things consistent. There's more information below in the *Referencing subpackages* section.

4.9.2 Transition guide

Let's say we have a set of recipes that currently builds a C library, as well as Python and R bindings to that C library. xgboost, a recent machine learning library, is one such example. Under conda-build 2.0 and earlier, you needed to have 3 recipes - 1 for each component. Let's go over some simplified `meta.yaml` files. First, the C library:

```
package:
    name: libxgboost
    version: 1.0
```

Next, the Python bindings:

```
package:
    name: py-xgboost
    version: 1.0

requirements:
    build:
        - libxgboost  # you probably want to pin the version here, but there's no dynamic␣
↪way to do it
        - python
    run:
        - libxgboost  # you probably want to pin the version here, but there's no dynamic␣
↪way to do it
        - python
```

```
package:
    name: r-xgboost
    version: 1.0

requirements:
    build:
        - libxgboost  # you probably want to pin the version here, but there's no dynamic␣
↪way to do it
        - r-base
    run:
        - libxgboost  # you probably want to pin the version here, but there's no dynamic␣
↪way to do it
        - r-base
```

To build these, you'd need several conda-build commands, or a tool like conda-build-all to build out the various Python versions. With conda-build 3.0 and split packages from conda-build 2.1, we can simplify this to one coherent recipe that also includes the matrix of all desired Python and R builds.

First, the `meta.yaml` file:

```
package:
    name: xgboost
    version: 1.0

outputs:
    - name: libxgboost
    - name: py-xgboost
      requirements:
```

(continues on next page)

(continued from previous page)

```
        - {{ pin_subpackage('libxgboost', exact=True) }}
        - python  {{ python }}

  - name: r-xgboost
    requirements:
        - {{ pin_subpackage('libxgboost', exact=True) }}
        - r-base  {{ r_base }}
```

Next, the `conda_build_config.yaml` file, specifying our build matrix:

```
python:
    - 2.7
    - 3.5
    - 3.6
r_base:
    - 3.3.2
    - 3.4.0
```

With this updated method, you get a complete build matrix: 6 builds total. One libxgboost library, 3 Python versions, and 2 R versions. Additionally, the Python and R packages will have exact pins to the libxgboost package that was built by this recipe.

4.9.3 Creating conda-build variant config files

Variant input files are yaml files. Search order for these files is the following:

1. A file named `conda_build_config.yaml` in the user's HOME folder (or an arbitrarily named file specified as the value for the `conda_build/config_file` key in your .condarc file).
2. A file named `conda_build_config.yaml` in the current working directory.
3. A file named `conda_build_config.yaml` in the same folder as `meta.yaml` with your recipe.
4. Any additional files specified on the command line with the `--variant-config-files` or `-m` command line flags, which can be passed multiple times for multiple files. The `conda build` and `conda render` commands accept these arguments.

Values in files found later in this search order will overwrite and replace the values from earlier files.

Note: The key `conda_build/config_file` is a nested value:

```
conda_build:
  config_file: some/path/to/file
```

4.9.4 Using variants with the conda-build API

Ultimately, a variant is just a dictionary. This dictionary is provided directly to Jinja2 and you can use any declared key from your variant configuration in your Jinja2 templates. There are two ways that you can feed this information into the API:

1. Pass the `variants` keyword argument to API functions. Currently, the `build`, `render`, `get_output_file_path`, and `check` functions accept this argument. `variants` should be a dictionary where each value is a list of versions to iterate over. These are aggregated as detailed in the *Aggregation of multiple variants* section below.
2. Set the `variant` member of a Config object. This is just a dictionary. The values for fields should be strings or lists of strings, except "extended keys", which are documented in the *Extended keys* section below.

Again, with `meta.yaml` contents like:

```
package:
    name: compiled-code
    version: 1.0

requirements:
    build:
        - python {{ python }}
    run:
        - python {{ python }}
```

You could supply a variant to build this recipe like so:

```
variants = {"python": ["2.7", "3.5"]}
api.build(path_to_recipe, variants=variants)
```

Note that these Jinja2 variable substitutions are not limited to version numbers. You can use them anywhere, for any string value. For example, to build against different MPI implementations:

With `meta.yaml` contents like:

```
package:
    name: compiled-code
    version: 1.0

requirements:
    build:
        - {{ mpi }}
    run:
        - {{ mpi }}
```

You could supply a variant to build this recipe like this (`conda_build_config.yaml`):

```
mpi:
    - openmpi  # version spec here is totally valid, and will apply in the recipe
    - mpich  # version spec here is totally valid, and will apply in the recipe
```

Selectors are valid in `conda_build_config.yaml`, so you can have one `conda_build_config.yaml` for multiple platforms:

```
mpi:
    - openmpi  # [osx]
    - mpich    # [linux]
    - msmpi    # [win]
```

Jinja is not allowed in `conda_build_config.yaml`, though. It is the source of information to feed into other Jinja templates, and the buck has to stop somewhere.

4.9.5 About reproducibility

A critical part of any build system is ensuring that you can reproduce the same output at some future point in time. This is often essential for troubleshooting bugs. For example, if a package contains only binaries, it is helpful to understand what source code created those binaries, and thus what bugs might be present.

Since conda-build 2.0, conda-build has recorded its rendered `meta.yaml` files into the `info/recipe` folder of each package it builds. Conda-build 3.0 is no different in this regard, but the `meta.yaml` that is recorded is a frozen set of the variables that make up the variant for that build.

Note: Package builders may disable including the recipe with the `build/include_recipe` key in `meta.yaml`. If the recipe is omitted from the package, then the package is not reproducible without the source recipe.

4.9.6 Special variant keys

There are some special keys that behave differently and can be more nested:

- `zip_keys`: a list of strings or a list of lists of strings. Strings are keys in variant. These couple groups of keys, so that particular keys are paired, rather than forming a matrix. This is useful, for example, to couple vc version to Python version on Windows. More info below in the *Coupling keys* section.
- `pin_run_as_build`: should be a dictionary. Keys are package names. Values are "pinning expressions" - explained in more detail in *Customizing compatibility*. This is a generalization of the `numpy x.x` spec, so that you can pin your packages dynamically based on the versions used at build time.
- `extend_keys`: specifies keys that should be aggregated, and not replaced, by later variants. These are detailed below in the *Extended keys* section.
- `ignore_version`: list of package names whose versions should be excluded from `meta.yaml`'s requirements/build when computing hash. Described further in *Avoiding unnecessary builds*.

4.9.7 Coupling keys

Sometimes particular versions need to be tied to other versions. For example, on Windows, we generally follow the upstream Python.org association of Visual Studio compiler version with Python version. Python 2.7 is always compiled with Visual Studio 2008 (also known as MSVC 9). We don't want a `conda_build_config.yaml` like the following to create a matrix of Python/MSVC versions:

```
python:
  - 2.7
  - 3.5
vc:
  - 9
  - 14
```

Instead, we want 2.7 to be associated with 9, and 3.5 to be associated with 14. The `zip_keys` key in `conda_build_config.yaml` is the way to achieve this:

```
python:
  - 2.7
  - 3.5
vc:
  - 9
  - 14
zip_keys:
  - python
  - vc
```

You can also have nested lists to achieve multiple groups of `zip_keys`:

```
zip_keys:
  -
    - python
    - vc
  -
    - numpy
    - blas
```

The rules for `zip_keys` are:

1. Every list in a group must be the same length. This is because without equal length, there is no way to associate earlier elements from the shorter list with later elements in the longer list. For example, this is invalid, and will raise an error:

   ```
   python:
     - 2.7
     - 3.5
   vc:
     - 9
   zip_keys:
     - python
     - vc
   ```

2. `zip_keys` must be either a list of strings, or a list of lists of strings. You can't mix them. For example, this is an error:

   ```
   zip_keys:
     -
       - python
       - vc
     - numpy
     - blas
   ```

Rule #1 raises an interesting use case: How does one combine CLI flags like --python with `zip_keys`? Such a CLI flag will change the variant so that it has only a single entry, but it will not change the `vc` entry in the variant configuration. We'll end up with mismatched list lengths, and an error. To overcome this, you should instead write a very simple YAML file with all involved keys. Let's call it `python27.yaml`, to reflect its intent:

```
python:
  - 2.7
```

(continues on next page)

(continued from previous page)

```
vc:
  - 9
```

Provide this file as a command-line argument:

```
conda build recipe -m python27.yaml
```

You can also specify variants in JSON notation from the CLI as detailed in the *CONDA_* variables and command line arguments to conda-build* section. For example:

```
conda build recipe --variants "{'python': ['2.7', '3.5'], 'vc': ['9', '14']}"
```

4.9.8 Avoiding unnecessary builds

To avoid building variants of packages where pinning does not require having different builds, you can use the `ignore_version` key in your variant. Then all variants are evaluated, but if any hashes are the same, then they are considered duplicates, and are deduplicated. By omitting some packages from the build dependencies, we can avoid creating unnecessarily specific hashes and allow this deduplication.

For example, let's consider a package that uses NumPy in both run and build requirements, and a variant that includes 2 NumPy versions:

```
variants = [{"numpy": ["1.10", "1.11"], "ignore_version": ["numpy"]}]
```

`meta.yaml`:

```
requirements:
    build:
        - numpy {{ numpy }}
    run:
        - numpy
```

Here, the variant says that we'll have two builds - one for each NumPy version. However, since this recipe does not pin NumPy's run requirement (because it doesn't utilize NumPy's C API), it is unnecessary to build it against both NumPy 1.10 and 1.11.

The rendered form of this recipe, with conda-build ignoring NumPy's value in the recipe, is going to be just one build that looks like:

`meta.yaml`:

```
requirements:
    build:
        - numpy
    run:
        - numpy
```

`ignore_version` is an empty list by default. The actual build performed is probably done with the last 'numpy' list element in the variant, but that's an implementation detail that you should not depend on. The order is considered unspecified behavior because the output should be independent of the input versions.

> **Warning:** If the output is not independent of input versions, don't use this key

Any pinning done in the run requirements will affect the hash, and thus builds will be done for each variant in the matrix. Any package that sometimes is used for its compiled interface and sometimes used for only its Python interface may benefit from careful use of `ignore_version` in the latter case.

Note: `pin_run_as_build` is kind of the opposite of `ignore_version`. Where they conflict, `pin_run_as_build` takes priority.

4.9.9 CONDA_* variables and command line arguments to conda-build

To ensure consistency with existing users of conda-build, environment variables such as CONDA_PY behave as they always have, and they overwrite all variants set in files or passed to the API.

The full list of respected environment variables are:

- CONDA_PY
- CONDA_NPY
- CONDA_R
- CONDA_PERL
- CONDA_LUA

CLI flags are also still available. These are sticking around for their usefulness in one-off jobs.

- --python
- --numpy
- --R
- --perl
- --lua

In addition to these traditional options, there's one new flag to specify variants: `--variants`. This flag accepts a string of JSON-formatted text. For example:

```
conda build recipe --variants "{python: [2.7, 3.5], vc: [9, 14]}"
```

4.9.10 Aggregation of multiple variants

The matrix of all variants is first consolidated from several dicts of lists into a single dict of lists, and then transformed in a list of dicts (using the Cartesian product of lists), where each value is a single string from the list of potential values.

For example, general input for `variants` could be something like:

```
a = {"python": ["2.7", "3.5"], "numpy": ["1.10", "1.11"]}
# values can be strings or lists.  Strings are converted to one-element lists internally.
b = {"python": ["3.4", "3.5"], "numpy": "1.11"}
```

Here, let's say `b` is found after `a`, and thus has priority over `a`. Merging these 2 variants yields:

```
merged = {"python": ["3.4", "3.5"], "numpy": ["1.11"]}
```

`b`'s values for `python` have overwritten `a`'s. From here, we compute the Cartesian product of all input variables. The end result is a collection of dicts, each with a string for each value. Output would be something like:

```
variants = [{"python": "3.4", "numpy": "1.11"}, {"python": "3.5", "numpy": "1.11"}]
```

conda-build would loop over these variants where appropriate, such as when building, outputting package output names, and so on.

If `numpy` had had two values instead of one, we'd end up with *four* output variants: 2 variants for `python`, *times* 2 variants for `numpy`:

```
variants = [
    {"python": "3.4", "numpy": "1.11"},
    {"python": "3.5", "numpy": "1.11"},
    {"python": "3.4", "numpy": "1.10"},
    {"python": "3.5", "numpy": "1.10"},
]
```

4.9.11 Bootstrapping pins based on an existing environment

To establish your initial variant, you may point to an existing conda environment. Conda-build will examine the contents of that environment and pin to the exact requirements that make up that environment.

```
conda build --bootstrap name_of_env
```

You may specify either environment name or filesystem path to the environment. Note that specifying environment name does mean depending on conda's environment lookup.

4.9.12 Extended keys

These are not looped over to establish the build matrix. Rather, they are aggregated from all input variants, and each derived variant shares the whole set. These are used internally for tracking which requirements should be pinned, for example, with the `pin_run_as_build` key. You can add your own extended keys by passing in values for the `extend_keys` key for any variant.

For example, if you wanted to collect some aggregate trait from multiple `conda_build_config.yaml` files, you could do something like this:

`HOME/conda_build_config.yaml`:

```
some_trait:
  - dog
extend_keys:
  - some_trait
```

`recipe/conda_build_config.yaml`:

```
some_trait:
  - pony
extend_keys:
  - some_trait
```

Note that *both* of the `conda_build_config.yaml` files need to list the trait as an `extend_keys` entry. If you list it in only one of them, an error will be raised to avoid confusion with one `conda_build_config.yaml` file that would add entries to the build matrix, and another which would not. For example, this should raise an error:

```
some_trait:
  - dog
```

recipe/conda_build_config.yaml:

```
some_trait:
  - pony
extend_keys:
  - some_trait
```

When our two proper YAML config files are combined, ordinarily the recipe-local variant would overwrite the user-wide variant, yielding `{'some_trait': 'pony'}`. However, with the `extend_keys` entry, we end up with what we've always wanted: a dog *and* pony show: `{'some_trait': ['dog', 'pony'])}`

Again, this is mostly an internal implementation detail - unless you find a use for it. Internally, it is used to aggregate the `pin_run_as_build` and `ignore_version` entries from any of your `conda_build_config.yaml` files.

4.9.13 Customizing compatibility

Pinning expressions

Pinning expressions are the syntax used to specify how many parts of the version to pin. They are by convention strings containing `x` characters separated by `.`. The number of version parts to pin is simply the number of things that are separated by `.`. For example, `"x.x"` pins major and minor version. `"x"` pins only major version.

Wherever pinning expressions are accepted, you can customize both lower and upper bounds.

```
# produces pins like >=1.11.2,<1.12
variants = [{"numpy": "1.11", "pin_run_as_build": {"numpy": {"max_pin": "x.x"}}}]
```

Note that the final pin may be more specific than your initial spec. Here, the spec is 1.11, but the produced pin could be 1.11.2, the exact version of NumPy that was used at build time.

```
# produces pins like >=1.11,<2
variants = [
    {"numpy": "1.11", "pin_run_as_build": {"numpy": {"min_pin": "x.x", "max_pin": "x"}}}
]
```

Note that for pre-release versions `min_pin` will be ignored and substituted with the exact input version since pre-releases can never match `>=x.x` (see *Package match specifications* for details on pre-release version matching).

Pinning at the variant level

Some packages, such as boost, *always* need to be pinned at runtime to the version that was present at build time. For these cases where the need for pinning is consistent, pinning at the variant level is a good option. Conda-build will automatically pin run requirements to the versions present in the build environment when the following conditions are met:

1. The dependency is listed in the requirements/build section. It can be pinned, but does not need to be.
2. The dependency is listed by name (no pinning) in the requirements/run section.
3. The `pin_run_as_build` key in the variant has a value that is a dictionary, containing a key that matches the dependency name listed in the run requirements. The value should be a dictionary with up to 4 keys: `min_pin`,

`max_pin`, `lower_bound`, `upper_bound`. The first 2 are pinning expressions. The latter 2 are version numbers, overriding detection of current version.

An example variant/recipe is shown here:

`conda_build_config.yaml`:

```
boost: 1.63
pin_run_as_build:
    boost:
      max_pin: x.x
```

`meta.yaml`:

```
requirements:
    build:
        - boost {{ boost }}
    run:
        - boost
```

The result here is that the runtime boost dependency will be pinned to `>=(current boost 1.63.x version),<1.64`.

More details on the `pin_run_as_build` function is below in the *Extra Jinja2 functions* section.

Note that there are some packages that you should not use `pin_run_as_build` for. Packages that don't *always* need to be pinned should be pinned on a per-recipe basis (described in the next section). NumPy is an interesting example here. It actually would not make a good case for pinning at the variant level. Because you only need this kind of pinning for recipes that use NumPy's C API, it would actually be better not to pin NumPy with `pin_run_as_build`. Pinning it is over-constraining your requirements unnecessarily when you are not using NumPy's C API. Instead, we should customize it for each recipe that uses NumPy. See also the *Avoiding unnecessary builds* section above.

Pinning at the recipe level

Pinning at the recipe level overrides pinning at the variant level, because run dependencies that have pinning values in `meta.yaml` (even as Jinja variables) are ignored by the logic handling `pin_run_as_build`. We expect that pinning at the recipe level will be used when some recipe's pinning is unusually stringent (or loose) relative to some standard pinning from the variant level.

By default, with the `pin_compatible('package_name')` function, conda-build pins to your current version and less than the next major version. For projects that don't follow the philosophy of semantic versioning, you might want to restrict things more tightly. To do so, you can pass one of two arguments to the `pin_compatible` function.

```
variants = [{"numpy": "1.11"}]
```

`meta.yaml`:

```
requirements:
    build:
        - numpy {{ numpy }}
    run:
        - {{ pin_compatible('numpy', max_pin='x.x') }}
```

This would yield a pinning of `>=1.11.2,<1.12`.

The syntax for the `min_pin` and `max_pin` is a string pinning expression. Each can be passed independently of the other. An example of specifying both:

```
variants = [{"numpy": "1.11"}]
```

meta.yaml:

```
requirements:
    build:
        - numpy {{ numpy }}
    run:
        - {{ pin_compatible('numpy', min_pin='x.x', max_pin='x.x') }}
```

This would yield a pinning of >=1.11,<1.12.

You can also pass the minimum or maximum version directly. These arguments supersede the min_pin and max_pin arguments and are thus mutually exclusive.

```
variants = [{"numpy": "1.11"}]
```

meta.yaml:

```
requirements:
    build:
        - numpy {{ numpy }}
    run:
        - {{ pin_compatible('numpy', lower_bound='1.10', upper_bound='3.0') }}
```

This would yield a pinning of >=1.10,<3.0.

4.9.14 Appending to recipes

As of conda-build 3.0, you can add a file named recipe_append.yaml in the same folder as your meta.yaml file. This file is considered to follow the same rules as meta.yaml, except that selectors and Jinja2 templates are not evaluated. Evaluation of selectors and Jinja2 templates will likely be added in future development.

Any contents in recipe_append.yaml will add to the contents of meta.yaml. List values will be extended and string values will be concatenated. The proposed use case for this is to tweak/extend central recipes, such as those from conda-forge, with additional requirements while minimizing the actual changes to recipe files so as to avoid merge conflicts and source code divergence.

4.9.15 Partially clobbering recipes

As of conda-build 3.0, you can add a file named recipe_clobber.yaml in the same folder as your meta.yaml file. This file is considered to follow the same rules as meta.yaml, except that selectors and Jinja2 templates are not evaluated. Evaluation of selectors and Jinja2 templates will likely be added in future development.

Any contents in recipe_clobber.yaml will replace the contents of meta.yaml. This can be useful, for example, for replacing the source URL without copying the rest of the recipe into a fork.

4.9.16 Differentiating packages built with different variants

With only a few things supported, we could just add things to the filename, such as py27 for Python, or np111 for NumPy. Variants are meant to support the general case, and in the general case this is no longer an option. Instead, used variant keys and values are hashed using the SHA1 algorithm, and that hash is a unique identifier. The information that went into the hash is stored with the package in a file at `info/hash_input.json`. Packages only have a hash when there are any "used" variables beyond the ones that are already accounted for in the build string (py, np, etc). The takeaway message is that hashes will appear when binary compatibility matters, but not when it doesn't.

Currently, only the first 7 characters of the hash are stored. Output package names will keep the pyXY and npXYY, but may have added the 7-character hash. Your package names will look like:

`my-package-1.0-py27h3142afe_0.tar.bz2`

As of conda-build 3.1.0, this hashing scheme has been simplified. A hash will be added if all of these are true for any dependency:

- Package is an explicit dependency in build, host, or run deps.
- Package has a matching entry in `conda_build_config.yaml` which is a pin to a specific version, not a lower bound.
- That package is not ignored by ignore_version.

OR

- Package uses {{ compiler() }} Jinja2 function.

Since conflicts only need to be prevented within one version of a package, we think this will be adequate. If you run into hash collisions with this limited subspace, please file an issue on the conda-build issue tracker.

There is a CLI tool that just pretty-prints this JSON file for easy viewing:

```
conda inspect hash-inputs <package path>
```

This produces output such as:

```
{'python-3.6.4-h6538335_1': {'files': [],
                             'recipe': {'c_compiler': 'vs2015',
                                        'cxx_compiler': 'vs2015'}}}
```

4.9.17 Extra Jinja2 functions

Two especially common operations when dealing with these API and ABI incompatibilities are ways of specifying such compatibility, and of explicitly expressing the compiler to be used. Three new Jinja2 functions are available when evaluating `meta.yaml` templates:

- `pin_compatible('package_name', min_pin='x.x.x.x.x.x', max_pin='x', lower_bound=None, upper_bound=None)`: To be used as pin in run and/or test requirements. Takes package name argument. Looks up compatibility of named package installed in the build environment and writes compatible range pin for run and/or test requirements. Defaults to a semver-based assumption: `package_name >=(current version),<(next major version)`. Pass `min_pin` or `max_pin` a *Pinning expressions* . This will be enhanced as time goes on with information from ABI Laboratory.
- `pin_subpackage('package_name', min_pin='x.x.x.x.x.x', max_pin='x', exact=False)`: To be used as pin in run and/or test requirements. Takes package name argument. Used to refer to particular versions of subpackages built by parent recipe as dependencies elsewhere in that recipe. Can use either pinning expressions, or exact (including build string).

- `compiler('language')`: To be used in build requirements most commonly. Run or test as necessary. Takes language name argument. This is shorthand to facilitate cross-compiler usage. This Jinja2 function ties together 2 variant variables, `{language}_compiler` and `target_platform`, and outputs a single compiler package name. For example, this could be used to compile outputs targeting x86_64 and arm in one recipe, with a variant.

There are default "native" compilers that are used when no compiler is specified in any variant. These are defined in conda-build's jinja_context.py file. Most of the time, users will not need to provide compilers in their variants - just leave them empty and conda-build will use the defaults appropriate for your system.

4.9.18 Referencing subpackages

Conda-build 2.1 brought in the ability to build multiple output packages from a single recipe. This is useful in cases where you have a big build that outputs a lot of things at once, but those things really belong in their own packages. For example, building GCC outputs not only GCC, but also GFortran, g++, and runtime libraries for GCC, GFortran, and g++. Each of those should be their own package to make things as clean as possible. Unfortunately, if there are separate recipes to repack the different pieces from a larger, whole package it can be hard to keep them in sync. That's where variants come in. Variants, and more specifically the `pin_subpackage(name)` function, give you a way to refer to the subpackage with control over how tightly the subpackage version relationship should be in relation to other subpackages or the parent package. The following will output 5 conda packages.

`meta.yaml`:

```
package:
  name: subpackage_demo
  version: 1.0

requirements:
  run:
    - {{ pin_subpackage('subpackage_1') }}
    - {{ pin_subpackage('subpackage_2', max_pin='x.x') }}
    - {{ pin_subpackage('subpackage_3', min_pin='x.x', max_pin='x.x') }}
    - {{ pin_subpackage('subpackage_4', exact=True) }}


outputs:
  - name: subpackage_1
    version: 1.0.0
  - name: subpackage_2
    version: 2.0.0
  - name: subpackage_3
    version: 3.0.0
  - name: subpackage_4
    version: 4.0.0
```

Here, the parent package will have the following different runtime dependencies:

- subpackage_1 >=1.0.0,<2 (default uses `min_pin='x.x.x.x.x.x`, `max_pin='x'`, pins to major version with default >= current version lower bound)
- subpackage_2 >=2.0.0,<2.1 (more stringent upper bound)
- subpackage_3 >=3.0,<3.1 (less stringent lower bound, more stringent upper bound)
- subpackage_4 4.0.0 h81241af (exact pinning - version plus build string)

4.9.19 Compiler packages

On macOS and Linux, we can and do ship GCC packages. These will become even more powerful with variants since you can specify versions of your compiler much more explicitly and build against different versions, or with different flags set in the compiler package's activate.d scripts. On Windows, rather than providing the actual compilers in packages, we still use the compilers that are installed on the system. The analogous compiler packages on Windows run any compiler activation scripts and set compiler flags instead of actually installing anything.

Over time, conda-build will require that all packages explicitly list their compiler requirements this way. This is to both simplify conda-build and improve the tracking of metadata associated with compilers - localize it to compiler packages, even if those packages are doing nothing more than activating an already-installed compiler, such as Visual Studio.

Note also the `run_exports` key in `meta.yaml`. This is useful for compiler recipes to impose runtime constraints based on the versions of subpackages created by the compiler recipe. For more information, see the *Export runtime requirements* section of the `meta.yaml` docs. Compiler packages provided by Anaconda use the `run_exports` key extensively. For example, recipes that include the `gcc_linux-cos5-x86_64` package as a build time dependency (either directly, or through a `{{ compilers('c') }}` Jinja2 function) will automatically have a compatible libgcc runtime dependency added.

4.9.20 Compiler versions

Usually the newest compilers are the best compilers, but in some special cases you'll need to use older compilers.

For example, NVIDIA's CUDA libraries only support compilers that they have rigorously tested. Often the latest GCC compiler is not supported for use with CUDA. If your recipe needs to use CUDA, you'll need to use an older version of GCC.

There are special keys associated with the compilers. The key name of each special key is the compiler key name plus `_version`.

For example, if your compiler key is `c_compiler`, the version key associated with it is `c_compiler_version`. If you have a recipe for Tensorflow with GPU support, put a `conda_build_config.yaml` file alongside `meta.yaml`, with contents like:

```
c_compiler_version:    # [linux]
    - 5.4              # [linux]
cxx_compiler_version:  # [linux]
    - 5.4              # [linux]
```

Specify selectors so that this extra version information is not also applied to Windows and macOS. Those platforms have totally different compilers and could have their own versions if necessary.

It is not necessary to specify `c_compiler` or `cxx_compiler` because the default value (`gcc` on Linux) will be used. It is necessary to specify both `c` and `cxx` versions, even if they are the same, because they are treated independently.

By placing this file in the recipe, it will apply only to this recipe. All other recipes will default to the latest compiler.

Note: The version number you specify here must exist as a package in your currently configured channels.

4.9.21 Cross-compiling

The compiler Jinja2 function is written to support cross-compilers. This depends on setting at least 2 variant keys: `(language)_compiler` and `target_platform`. The target platform is appended to the value of `(language)_compiler` with the _ character. This leads to package names like `g++_linux-aarch64`. We recommend a convention for naming your compiler packages as: `<compiler name>_<target_platform>`.

Using a cross-compiler in a recipe would look like the following:

```
variants = {
    "cxx_compiler": ["g++"],
    "target_platform": ["linux-cos5-x86_64", "linux-aarch64"],
}
```

and a `meta.yaml` file:

```
package:
    name: compiled-code
    version: 1.0

requirements:
    build:
        - {{ compiler('cxx') }}
```

This assumes that you have created 2 compiler packages named `g++_linux-cos5-x86_64` and `g++_linux-aarch64` - all conda-build is providing you with is a way to loop over appropriately named cross-compiler toolchains.

4.9.22 Self-consistent package ecosystems

The compiler function is also how you could support a non-standard Visual Studio version, such as using VS 2015 to compile Python 2.7 and packages for Python 2.7. To accomplish this, you need to add the `{{ compiler('<language>') }}` to each recipe that will make up the system. Environment consistency is maintained through dependencies - thus it is useful to have the runtime be a versioned package with only one version being able to be installed at a time. For example, the `vc` package, originally created by Conda-Forge, is a versioned package (only one version can be installed at a time), and it installs the correct runtime package. When the compiler package imposes such a runtime dependency, then the resultant ecosystem is self-consistent.

Given these guidelines, consider a system of recipes using a variant like this:

```
variants = {"cxx_compiler": ["vs2015"]}
```

The recipes include a compiler `meta.yaml` like this:

```
package:
    name: vs2015
    version: 14.0
build:
    run_exports:
        - vc 14
```

They also include some compiler-using `meta.yaml` contents like this:

```
package:
    name: compiled-code
```

(continues on next page)

(continued from previous page)

```
    version: 1.0

requirements:
    build:
        # these are the same (and thus redundant) on windows, but different elsewhere
        - {{ compiler('c') }}
        - {{ compiler('cxx') }}
```

These recipes will create a system of packages that are all built with the VS 2015 compiler, and which have the vc package matched at version 14, rather than whatever default is associated with the Python version.

4.10 Conda-build CLI reference

Command-line interface (CLI) adds onto conda-build's functionality. CLI provides functions enabling you to convert packages between formats, render recipes, develop an index of packages, and more.

4.10.1 conda-build

4.10.2 conda convert

4.10.3 conda develop

4.10.4 conda index

4.10.5 conda inspect

4.10.6 conda inspect channels

4.10.7 conda inspect linkages

4.10.8 conda inspect objects

4.10.9 conda metapackage

4.10.10 conda render

4.10.11 conda skeleton

4.10.12 conda skeleton cpan

4.10.13 conda skeleton cran

4.10.14 conda skeleton luarocks

4.10.15 conda skeleton pypi

4.11 Adding Windows Start menu items

When a package is installed, it can add a shortcut to the Windows **Start** menu. Conda and conda-build handle this with the package menuinst, which currently supports only Windows. For instructions on using menuinst, see the menuinst wiki.

The easiest way to ensure that a package made with conda constructor does not install any menu shortcuts is to remove menuinst from the list of conda packages included. To do this, add the following to the `constuct.yaml` file:

```
exclude:
  - menuinst
```

4.12 Writing style guide

Follow these guidelines for submitting or editing conda-build documentation.

4.12.1 Audience

Identify who your audience is, their skill level, and how they can use the information.

4.12.2 Technical language

Match the level of technical language with the audience's level of proficiency. It's better to uderestimate the knowledge of your readers than overestimate it. Limit technical terms to those the user will encounter. If you must define a large number of terms, use a glossary to supplement definitions in the text.

4.12.3 Addressing the user

Use the active voice (e.g. Click this) and address users directly (write "you" rather than "the user"). When explaining an action, use the "command" form of the verb: "Choose an option from the menu and press Enter."

4.12.4 Format

See the *tutorial template* for the format. Provide descriptive task and subtask titles and do not number headings.

4.13 Tutorial template

- *Overview*
- *Who is this for?*
- *Before you start*
- *Tutorial tasks*
- *More information (optional)*

This document describes the steps for creating a tutorial for conda-build.

- *Copy the template*: https://github.com/conda/conda-build/tree/master/docs/source/resources/tutorial-template.rst.
- *Replace the italicized text with your tutorial content, following the writing style guide.*
- *Review other tutorials for additional guidance.*
- *Contact us at* documentation@anaconda.com *for help.*

4.13.1 Overview

Provide descriptions of the tutorial's:

- *Purpose*.
- *Benefits*.
- *Application*.

4.13.2 Who is this for?

- *Who is your audience*?
- *What skills or prior knowledge do they need*?
- *How will they use this tutorial*?

4.13.3 Before you start

Before you start, check the *Prerequisites*.

Provide descriptions of and links to requisite programs.

Glossary (optional)

If you're using several technical terms that your readers may be unfamiliar with, provide a glossary of key terms.

Glossary	Definition
Term 1	Term 1 defined
Term 2	Term 2 defined
Term 3	Term 3 defined

4.13.4 Tutorial tasks

- *Provide descriptive titles for each section*.
- *Identify the major tasks*.
- *Separate each major task into subtasks*.
- *Write a series of steps that walk the user through each subtask*.

4.13.5 More information (optional)

- *Provide links to related content*.
- *Add final notes for how to expand upon the tutorial*.

CHAPTER

FIVE

RELEASE NOTES

This information is drawn from the GitHub conda-build project changelog.

5.1 3.21.7 (2021-11-30)

5.1.1 Bug fixes

- Handle an import from the vendored auxlib library in Conda 4.11.0 better.

5.1.2 Other

- cran-skeleton: more unit tests

5.1.3 Contributors

- @kenodegard
- @jdblischak
- @jezdez
- @pre-commit

5.2 3.21.6 (2021-11-09)

5.2.1 Enhancements

- Add limited support for platform_system/sys_platform env markers in PyPI skeleton
- cran-skeleton: Adds a flag *--no-comments* to remove the instructional comments from the recipe files.

5.2.2 Bug fixes

- When checking for circular dependencies in cross-compiling mode, `build` requirements are ignored now.
- Make sure symlinked directories are found in always_include_files
- Fix pinning expressions for prerelease builds

5.2.3 Contributors

- @isuruf
- @mbargull
- @kenodegard
- @jdblischak
- @dbast
- @jezdez
- @ChristofKaufmann

5.3 3.21.5 (2021-08-06)

5.3.1 Enhancements

- Revert "Consider any file containing .yaml in its name as maybe a recipe file" (#4235)
- Support setting *build/script_env* values containing "=" (#4211)
- Drop Python 2.7 support from *setup.py* (#4202)
- Make variant configuration error message more informative (#4198)
- Ensure file globs are always sorted (#4186)
- Add preliminary support for *prelink_message* files in conda packages (#4203)

5.3.2 Bug fixes

- Do not munge rpath for non Mach-O files on macOS (#4238)
- Fix Windows test file extension reported by *conda-debug* (#4224)

5.3.3 Documentation

- Document *build/script_env* recipe option (#4211)
- Clarify wording about selecting multiple operating systems (#4139)

5.3.4 Contributors

- @chrisburr
- @gabm
- @isuruf
- @jacobtylerwalls
- @katietz
- @kenodegard
- @marcelotrevisani
- @xhochy

5.4 3.21.4 (2021-01-15)

5.4.1 Enhancements

- Add new centos 7 distribution cleof to rpm skeleton for s390x. (#4181)

5.4.2 Bug fixes

- Fixed bug where symlinks in symlinks caused conda build to exit. (#4180)

5.4.3 Contributors

- @mingwandroid
- @beckermr
- @katietz
- @beckermr

5.5 3.21.3 (2021-01-11)

5.5.1 Enhancements

- Fix stupid error in prefix replacement (#4177)

5.5.2 Contributors

- @mingwandroid

5.6 3.21.2 (2021-01-11)

5.6.1 Contributors

5.7 3.21.1 (2021-01-11)

5.7.1 Bug fixes

- Fix noarch: python version from version-age determination (#4174)

5.7.2 Contributors

- @mingwandroid

5.8 3.21.0 (2021-01-10)

5.8.1 Enhancements

- activate_in_script defaults to true (#4120)
- Add Setting and build/noarch_python_build_age and fix tests not finding packages (#4120)
- Allow directories as license_file source (#4153)
- Consider any file containing .yaml in its name as maybe a recipe file (#4120)
- Add weak_constrains and strong_constrains run_exports types (#4125)
- Issue a single command for the upload command (#4120)
- Print hash_inputs after upload info (#4120)
- Add cross-r-base for cross compiling
- Add --build-id-pat option
- macOS: Delete build_prefix rpaths
- Use smarter build_number
- Combine default_structs with FIELDS
- Fix conda render indent from 4 to 2
- macOS: arm64 ci/test-suite setup
- Removing more conda-forge testing deps
- Variants: Be more informative
- more verbosity in tests

- Use MacOSX10.10.sdk, not MacOS10.9.sdk in tests (#4120)
- Warn when files have been removed from the prefix (#4120)

5.8.2 Bug fixes

- Add conda-verify to install_conda_build_test_deps (#4120)
- Add flaky to testing dependencies (#4138)
- Fix tests not finding packages
- Avoid writing to the package cache in package_has_file (collisions) (#4120)
- Change package_has_file to refresh if out of date (#4120)
- Ensure ~/.condarc does not leak into testing_config (#4120)
- Fix applying patches to read-only files (#4140)
- Fix auth in aboutjson (#4137)
- Fix skeleton URLs for CentOS 6 (EOL) and various CI fixes (#4154)
- Fix typo in cran skeleton (#4143)
- Force channel_targets to be considered used (#4120)
- Fix printing `bytes-like object is required, not 'str'` when applying patches (#4118)
- Set "platform" in index.json to the target platform for cross-platform builds (#4124)
- Reduce get_rpaths_raw/patchelf disagree warnings (#4131)
- LIEF: Allow parsing static libs to fail (#4149)
- pass cache_dir to api.build (#4120)
- Fix symlinks to directories
- Make post-link run_export/library_nature determination less work when CONDA_OFFLINE=1
- Remove Python 2.7 from CI matrix
- Fix test_pypi_installer_metadata (builds against python 3.9 not 3.7)
- tests: Fix test_render_with_python_arg_reduces_subspace
- tests: Update python 3 from 3.5/6 to 3.9 in many
- Set numpy default to 1.16
- tests: Fix pins for numpy_used
- tests: CI: Win: Circumvent delayed expansion
- Install patch or m2-patch, write .sh files as binary, more Win tests
- tests: Avoid issue with coverage==5.0 on Win+Py2.7
- Assume non-revisible patches
- Add flaky marker and --strict-markers to setup.cfg
- Don't sort recipes
- Use extra R_ARGS and fix them
- shell check fix

5.8.3 Contributors

- @mingwandroid
- @isuruf
- @mbargull
- @njalerikson
- @cjmartian
- @chrisburr
- @hugobuddel
- @kurtschelfthout

5.9 3.20.5 (2020-10-26)

5.9.1 Enhancements

- A new feature `build/ignore_run_exports_from` which will ignore `run_exports`
- coming from a package listed in `build/ignore_run_exports_from`. (#4114)

5.9.2 Bug fixes

- Respect PEP440 ~= 'Compatible release clause' (#4113)
- Detect amalgamated patches (#4099)
- Handle realpath properly in unsafe patch check (#4099)
- Force channel_targets to be considered used (#4099)
- Look for git in build_prefix in git_info (#4099)
- Fall back to shutil.copy if shutil.copy2 fails when copying patches (#4099)
- Fix indexing by file (#4111)
- Helper functions to extract keys (#4088)
- Simplify find_config_files call (#4086)
- Refactor dict_of_lists_to_lists_of_dict (#4075)

5.9.3 Contributors

- @mingwandroid
- @isuruf
- @njalerikson
- @cjmartian
- @njalerikson

5.10 3.20.4 (2020-10-14)

5.10.1 Enhancements

- Make stats output more easily human-readable (#4069)
- Prefer meta.yaml build/error_overlinking and error_overdepending (#4074)
- Cleanup variant processing code (#4075)
- Add --file option to indexing (#4076)

5.10.2 Bug Fixes

- Remove old rpath when loader_path is used (#4080)
- Fix MACOSX_DEPLOYMENT_TARGET default for osx-arm64 (#4091)
- Rewrite apply_patch again (#4092)
- Add a .* to conditional_regex (#4092)

5.10.3 Contributors

- @isuruf
- @njalerikson
- @cjmartian
- @mingwandroid

5.11 3.20.3 (2020-09-29)

5.11.1 Enhancements

- Use CONDA_PACKAGE_EXTENSIONS (#4053)
- raise runtimeerror instead of calling sys.exit (#4062)
- Refactor conda_build.build.get_all_replacements (#4055)

5.11.2 Bug fixes

- Do not clobber config argument in conda_build.build.build_tree (#4066)
- Use --dry-run to test that a patch applies. Fixes bug 4054 (#4067)
- Include target_platform in package build string hash (#4065)
- Fix post linking for SDKs with tapi-tbd-v4 (MacOS 11.0 and upwards) (#4048)

5.11.3 Contributors

5.12 3.20.1 (2020-09-04)

5.12.1 Enhancements

5.12.2 Bug fixes

- Run bash with -e in outputs too #4033
- Add target to recognized fields in *outputs* #4034
- Various overlinking fixes for Windows #4036
- variants: remove hard-coded default path for CONDA_BUILD_SYSROOT

5.12.3 Deprecations

5.12.4 Docs

5.12.5 Other

5.12.6 Contributors

- @mingwandroid
- @isuruf
- @mbargull

5.13 3.20.0 (2020-08-27)

5.13.1 Enhancements

- enable Python 3.8 on Azure Pipelines (#3841)
- which_package can be passed avoid_canonical_channel_name (#3952)
- make life easier (less shell exit-y) for those who source test scripts (#3952)
- move old host env instead of deleting it when *--keep-old-work* (#3952)
- convert info.d/*.yaml to info/*.json (#3952)
- allow manual specification of which binary files to prefix replace (#3952)
- filter out '.AppleDouble' folders from recipe searches (#3952)
- re-wrote apply_patch() to be more robust (#3952)
- many fixes for DSO post-processing (#3952, #3953)
- add support for (limited) tbd parsing (#3953)
- Make sure packages in current repo data w/ features have versions without features (#3957)
- Check all sysroot locations for DSOs (#3969)

- More helpful error message if an empty string is passed as the hash ('md5', 'sha1' or 'sha256' fields) (#3971)
- the `GIT_DESCRIBE_HASH` variable will be available regardless of whether the sources of the recipe have a git tag or not (#3982)
- add apple silicon support (#4004, #4015)
- set build_platform for aid in cross compiling (#4005)
- import macho on non apple system for cross compiling (#4025)
- Add ccache as a jinja 2 function (#4026)
- Improve cpan skeleton (#4026)
- Retry moving host prefix due to Windows file locking (#4026)
- Rename ccache method from mklink to sylinks (#4028)

5.13.2 Bug fixes

- conda_build.metadata: fixed typos in FIELDS (#3866)
- add spaces in CRAN templates (fixes #3943) (#3944)
- raise valid CalledProcessException in macho.otool (#3952)
- cache local_output_folder too for get_build_index (#3952)
- fix relocations when cross compiling (#3995)
- use host_platform instead of sys.platform to facilitate cross compiling (#3997)
- Fix parsing UnsatisfiableError from conda>=4.7.8 (#4001)
- allow packages to depend on themselves when cross compiling (#4011)
- set the correct SHLIB_EXT when cross compiling (#4013, #4021)
- inspect linkages with pyldd when not DLL/EXE files (#4019)
- Respect no_rewrite_stdout_env on Windows (#4026)
- Prefix replacement fixes (#4026)
- Use git am -3 when applying patches (#4026)
- Fix env_var=val assertion (#4026)
- Use exit /B from patch files (#4026)

5.13.3 Docs

- extend docs o generating the index (#3877)
- add details to documentation of run_constrained (#3878)
- remove documentation on bdist_conda and environment variables (#3879)
- update cli help information for conda index (#3931)
- Clarify how to install conda-build (#3976)
- Add note for local package install deps (#3980)
- Clarify multiple OS selection (#3984)

- add aarch64 selector to the docs (#4003)
- add docs on build_platform and arm64 (#4020)

5.13.4 Other

- Enable s390x support (#3949, #4030)
- Add xfail test for non-utf-8 charsets (#3972)
- Improve testing on CI (#3987, #4017, #4027)
- Allow python=3.8 for pypi skeletons (#4014)

5.14 3.19.3 (2020-04-13)

5.14.1 Bug fixes

- load log prior to calling warn method (#3925)
- test suite fixes and prefix replacement fixes (#3932)

5.14.2 Other

- Enable ppc64 support (#3921)

5.14.3 Docs

- Update cli help information for conda index (#3931)

5.14.4 Contributors

- @beenje
- @jjhelmus
- @mingwandroid

5.15 3.19.2 (2020-04-01)

5.15.1 Bug fixes

- Show a warning instead of failing if a Mach-O file is prouduced by a build running on a platform other than macOS (#3912)
- Revert #3893, restores behavior of build/binary_has_prefix_files to that found in 3.18.12 (#3916)

5.15.2 Docs

- clarified 'deletes the build environment' in concepts/recipe.rst (#3901)

5.15.3 Contributors

- @jjhelmus
- @timsnyder
- @chrisburr

5.16 3.19.1 (2020-03-17)

5.16.1 Bug fixes

- Fix issues with PREFIX detection in Windows #3899

5.16.2 Other

- Change the CI trigger #3904

5.16.3 Contributors

- @mingwandroid
- @marcelotrevisani
- @jjhelmus

5.17 3.19.0 (2020-03-10)

5.17.1 Enhancements

- Keep python pinning in hashing if there is a space #3895
- ci launcher supporting python d shebangs on Windows #3894
- Allow build/binary_has_prefix_files to specify a list of files #3893

5.17.2 Bug fixes

- Use patchelf to set RPATH by default #3897

5.17.3 Contributors

- @isuruf
- @jjhelmus
- @mingwandroid

5.18 3.18.12 (2020-03-02)

- Keep python pinning in hashing if there is a space #3895
- ci launcher supporting python d shebangs on Windows #3894
- Allow build/binary_has_prefix_files to specify a list of files #3893
- Use patchelf to set RPATH by default #3897
- Prevent non-atomic writes to repodata JSON files #3833
- Audited and updated all docs with formatting, grammar, and accuracy errors.
- Docs: Removed deprecated page on features
- Fixed issue where symlinks to files that do not exist break conda build #3840

5.18.1 Contributors

- @bdice
- @beckermr
- @chrisburr
- @csoja
- @guidara
- @isuruf
- @jakirkham
- @jjhelmus
- @marcelotrevisani
- @mcg1969
- @mingwandroid
- @msarahan
- @rrigdon
- @saraedum
- @sscherfke
- @zeehio

5.19 3.18.11 (2019-11-01)

- Update build.sh files of skeletons to be shellcheck clean including test to lint future updates.
- Corrected documentation on subpackage test requirements.
- Do not move work dir to work/work/
- fixed a missing .lower() on two tar_xf related util functions
- Fix has_prefix detection for Windows.
- conda_build.inspect_pkg: optimise use of fnmatch
- Do not consider .ignore files when searching with ripgrep
- Remove N*N os.lstat calls in build_info_files_json_v1

5.19.1 Contributors

- @msarahan
- @rrigdon
- @marcelotrevisani
- @rrigdon
- @soapy1
- @dbast
- @duncanmmacleod
- @beckermr
- @seanyen
- @AndrewAnnex
- @183amir
- @njzjz

5.20 3.18.10 (2019-10-14)

5.20.1 Enhancements

- Added the error message when an invalid pip dependency version expression is used
- Conda skeleton pypi quoting just *version*, *summary* ` and *description* or attributes with special characters
- Set up CI Azure pipeline for Linux
- Update cran skeleton to match supported optional licenses for license file derivation.
- Migrate Unittests to PyTest
- Update script command on conda skeleton pypi to use *{{ PYTHON }} -m pip install . -vv*
- Add a warning when a received a file on *RECIPE_PATH*
- Refactored the skeletons/pypi.py get_package_metadata to be more modular

- added --suppress-variables switch to hide environment variables from console output

5.20.2 Bug fixes

- Fixed build of '.conda' packages enabled via 'conda config --set conda_build.pkg_format 2'
- Workaround for future deprecations of the SafeConfigParser and readfp of the same module.

5.20.3 Docs

- Remove bzip2 package from build toolkit description.

5.20.4 Other

5.20.5 Contributors

- @msarahan
- @jakirkham
- @marcelotrevisani
- @duncanmmacleod
- @kinow
- @saraedum
- @jjhelmus
- @rrigdon
- @mingwandroid
- @asford
- @timsnyder
- @mcg1969
- @kaitietz
- @stuarteberg
- @isuruf
- @dbast
- @Bezier89

5.21 3.18.9 (2019-07-23)

5.21.1 Enhancements

- add --use-channeldata argument to conda render/build.
- Extract the part in the skeletons pypi responsible to get the package metadata to a free function.
- Creat unittests for the get_package_metadata (skeletons/pypi.py) and for the new functions.

5.21.2 Bug fixes

- Limit threads to 61 on Windows.
- Do not use channeldata for run_exports unless --use-channeldata specified.
- Finalize top-level metadata if not present as an output.

5.21.3 Docs

- Add 3.18.7 release notes

5.21.4 Other

- Add disable_pip to FIELDS

5.21.5 Contributors

- @rrigdon
- @jjhelmus
- @rrigdon
- @Bezier89
- @jakirkham
- @marcelotrevisani

5.22 3.18.8 (2019-07-18)

5.22.1 Enhancements

- license_file can optionally be a yaml list

5.22.2 Bug fixes

- fix readup of existing index.json in cache while extracting
- fix spurious post build errors/warning message
- merge channeldata from all urls

5.22.3 Contributors

- @msarahan
- @rrigdon
- @jjhelmus
- @isuruf
- @ddamiani

5.23 3.18.7 (2019-07-09)

5.23.1 Enhancements

- Update authorship for 3.18.7
- Add note on single threading for indexing during build
- Add in fallback for run_exports when channeldata not available
- Make pins for current_repodata additive - always newest, and pins are additions to that
- Limit indexing in build to using one thread
- Speed up by allowing empty run_exports entries in channeldata be valid results
- Bump conda-package-handling to 1.3+
- Add test for run_exports without channeldata
- Fallback to file-based run_exports if channeldata has no results
- Add Mozilla as valid license family
- Add in fallback for run_exports when channeldata not available
- Updated tutorials and resource documentation

5.23.2 Bug fixes

- Flake8 and test fixes from pytest deprecations
- Fix in render.py::_read_specs_from_package
- Fix for pkg_loc
- Fix conda debug output being suppressed

5.23.3 Contributors

- @msarahan
- @rrigdon
- @rrigdon
- @scopatz
- @mbargull
- @jakirkham
- @oleksandr-pavlyk

5.24 3.18.6 (2019-06-26)

5.24.1 Enhancements

- package sha256 sums are includex in index.html

5.24.2 Bug fixes

- fix bug where package filenames were not included in the index.html

5.24.3 Contributors

- @rrigdon
- @jjhelmus

5.25 3.18.5 (2019-06-25)

5.25.1 Bug fixes

- fix one more keyerror with missing timestamp data
- when indexing, allow .tar.bz2 files to use .conda cache, but not vice versa. This acts as a sanity check on the .conda files.
- add build/rpaths_patcher to meta.yaml, to allow switching between lief and patchelf for binary mangling

5.25.2 Contributors

- @mingwandroid
- @msarahan
- @csosborn

5.26 3.18.4 (2019-06-21)

5.26.1 Enhancements

- channeldata reworked a bit to try to capture any available run_exports for all versions available

5.26.2 Bug fixes

- make "timestamp" an optional field in conda index operations

5.26.3 Contributors

- @msarahan

5.27 3.18.3 (2019-06-20)

5.27.1 Enhancements

- Make VS2017 default Visual Studio
- Add hook for customizing the behavior of conda render
- Drop */usr* from CDT skeleton path
- Update cran skeleton to use m2w64 compilers for windows instead of toolchain. The linter is telling since long: Using toolchain directly in this manner is deprecated.

5.27.2 Bug fixes

- Update cran skeleton to not use toolchain for win
- fix package_has_file so it supports .conda files (use cph)
- fix package_has_file function for .conda format
- fix off-by-one path trimming in prefix_files
- disable overlinking checks when no files in the package have any shared library linkage
- try to avoid finalizing top-level metadata twice
- try to address permission errors on Appveyor and Azure by falling back to copy and warning (not erroring) if removing a file after copying fails
- reduce the files inspected/loaded for channeldata, so that indexing goes faster

5.27.3 Deprecations

- The repodata2.json file is no longer created as part of indexing. It was not used by anything. It has been removed as an optimization. Its purpose was to explore namespaces, and we'll bring its functionality back when we address that fully.

5.27.4 Contributors

- @mingwandroid
- @msarahan
- @rrigdon
- @rrigdon
- @soapy1
- @mariusvniekerk
- @jakirkham
- @dbast
- @duncanmmacleod

5.28 3.18.2 (2019-05-26)

5.28.1 Bug fixes

- speed up post-link checks
- fix activation not running during tests
- improve indexing to show status better, and fix bug where size/hashes were being mixed up between .tar.bz2 and .conda files

5.28.2 Contributors

- @mingwandroid
- @msarahan
- @rrigdon

5.29 3.18.1 (2019-05-18)

5.29.1 Enhancements

- rearrange steps in index.py to optimize away unnecessary work
- restore parallel extract and hash in index operations

5.29.2 Contributors

- @msarahan

5.30 3.18.0 (2019-05-17)

5.30.1 Enhancements

- Set R_USER environment variable when building R packages
- Make Centos 7 default cdt distribution for linux-aarch64
- Bump default python3 version to 3.7 for CI
- Build docs if any docs related file changes
- Add support for conda pkgv2 (.conda) format
- add creation of "current_repodata.json" - like repodata.json, but only has the newest version of each file
- change repodata layout to support .conda files. They live under the "packages.conda" key and have similar subkeys to their .tar.bz2 counterparts.
- Always show display actions, regardless of verbosity level
- Ignore registry autorun for all cmd.exe invocations
- Relax default pinning on r-base for benefit of noarch R packages
- Make conda index produce repodata_from_packages.json{,.bz2} which contains unpatched metadata
- Use a shorter environment prefix when testing on unix-like platforms
- Prevent pip from clobbering conda installed python packages by populating .dist_info INSTALLER file

5.30.2 Bug fixes

- Allow build/missing_dso_whitelist section to be empty
- Make conda-debug honor custom channels passed using -c
- Do not attempt linkages inspection via lief if not installed
- Fix all lief related regressions brought in v3.17.x
- Fix ZeroDivisionError in ELF sections that have zero entries
- *binary_has_prefix_files* and *text_has_prefix_files* now override the automatically detected prefix replacement mode
- Handle special characters properly in pypi conda skeleton
- Build recipes in order of dependencies when passed to CB as directories
- Fix run_test script name for recipes with multiple outputs
- Fix recursion error with subpackages and build_id
- Avoid mutating global variable to fix tests on Windows
- Update CRAN license test case (replace r-ruchardet with r-udpipe)
- Update utils.filter_files to filter out generated .conda_trash files

- Replace stdlib glob with utils.glob. Latter supports recursion (**)

5.30.3 Docs

- Updated Sphinx theme to make notes and warnings more visible
- Added tutorial on building R-language packages using skeleton CRAN
- Add 37 to the list of valid values for CONDA_PY
- Corrected argparse rendering error
- Added tutorials section, reorganized content, and added a Windows tutorial
- Added Concepts section, removed extraneous content
- Added release notes section
- Reorganized sections
- Clarify to use 'where' on Windows and 'which' on Linux to inspect files in PATH
- Add RPATH information to compiler-tools documentation
- Improve the documentation on how to use the macOS SDK in build scripts.
- Document `conda build purge-all`.
- Fix user-guide index
- Add example for meta.yaml
- Updated theme
- Reorganized conda-build topics, updated link-scripts

5.30.4 Contributors

- @mingwandroid
- @msarahan
- @rrigdon
- @jjhelmus
- @nehaljwani
- @scopatz
- @Bezier89
- @rrigdon
- @isuruf
- @teake
- @jdblischak
- @bilderbuchi
- @soapy1
- @ESSS
- @tjd2002

- @tovrstra
- @chrisburr
- @katietz
- @hrzafer
- @zdog234
- @gabrielcnr
- @saraedum
- @uilianries
- @theultimate1
- @scw
- @spalmrot-tic

5.31 3.17.8 (2019-01-26)

5.31.1 Bug fixes

- provide fallback from libarchive back to python tarfile handling for handling tarfiles containing symlinks on windows

5.31.2 Other

- Rever support added for releasing conda-build

5.31.3 Contributors

- @msarahan
- @jjhelmus
- @scopatz
- @rrigdon
- @ax3l
- @rrigdon

5.32 3.17.7 (2019-01-16)

5.32.1 Bug fixes

- respect context.offline setting #3328
- don't write bytecode when building noarch: python packages #3330
- escape path separator in repl #3336

- remove deprecated sudo statement from travis CI configuration #3338
- fix running of test scripts in outputs #3343
- allow overriding one key of zip_keys as long as length of group agrees #3344
- fix compatibility with conda 4.6.0+ #3346
- update centos 7 skeleton (CDT) URL #3350

5.32.2 Contributors

- @iainsgillis
- @isuruf
- @jjhelmus
- @nsoranzo
- @msarahan
- @qwhelan

5.33 3.17.6 (2018-12-19)

5.33.1 Bug fixes

- don't raise when recipe text can't be extracted if manual build string is already set #3326

5.33.2 Contributors

- @msarahan

5.34 3.17.5 (2018-12-14)

5.34.1 Bug fixes

- fix pip build isolation / fix absence of "falsey" env vars. Ignore only if empty string or None. #3319
- pass-through VS20XYINSTALLDIR var (used by intel compiler to locate VS2017 installation) #3322

5.34.2 Contributors

- @jjhelmus
- @msarahan

5.35 3.17.4 (2018-12-12)

5.35.1 Bug fixes

- fix python-3 only JSON decode error handling (make py27 compatible) #3307
- fix too much caching in getting used vars from meta.yaml leading to inaccurate hash contents #3311
- fix merge of build/host not being recognized before an rm_rf call utilized that info #3311

5.35.2 Contributors

- @Lnaden
- @msarahan

5.36 3.17.3 (2018-12-11)

5.36.1 Bug fixes

- ignore non-native binaries in lief for now. Cross-platform inspection still theoretically possible using subdir parameter. #3306

5.36.2 Contributors

- @msarahanl

5.37 3.17.2 (2018-12-11)

5.37.1 Bug fixes

- fix to ignore unsatisfiable pin_compatible calls for packages in other outputs #3277
- add license files to CRAN recipes generated by conda skeleton #3284
- restrict py-lief to running on linux/macos only for now #3291,
- fix lief operation on files that are missing dynamic section (e.g. go binaries) #3292
- expand instructions on how to setup a dev env for conda-build #3296
- fix file= keyword being passed to a logger call #3298
- add test for standalone DLLs with py-lief, don't error out on them #3301
- rename windows build script runner to avoid confusion with existing bld.bat files in root dir #3303
- fix file URL handling on Windows #3303
- use conda's download function rather than requests directly, so that conda's proxy settings are respected #3303
- silence patch output when output verbosity is False #3305

5.37.2 Contributors

- @bergtholdt
- @dsludwig
- @jdblischak
- @msarahan
- @nehaljwani
- @sodre

5.38 3.17.1 (2018-12-04)

5.38.1 Bug fixes

- omit LIEF depedency on Windows until it is better tested #3288
- activate host environment #3288
- allow calls to nm to fail #3290

5.38.2 Contributors

- @jjhelmus
- @msarahan
- @nehaljwani

5.39 3.17.0 (2018-11-28)

5.39.1 Enhancements

- tell pip to not go find things on PyPI (turn off downloading) #3109
- new "conda debug" command for creating build/host or test envs for working on recipes #3237
- new package check: "overdepending" - warns or errors out when your run dependencies include unnecessary shared library packages #3237
- utilize LIEF for analyzing shared object data, extending capabilities beyond pyldd #3237

5.39.2 Bug fixes

- avoid discarding build string during pin_run_as_build and ensure_valid_spec #3264
- fix conda index's handling of packages where 'depends' key doesn't exist #3270
- fix bad inversion assumption about pip's PIP_NO_DEPENDENCIES setting #3271
- fix regex to allow for whitespace after hyphens in outputs section #3274, #3275
- handle unicode decode fails in output rewriting #3279
- fix merge of repodata patches that have keys that don't exist in repo #3280

5.39.3 Contributors

- @bergtholdt
- @isuruf
- @minrk
- @msarahan
- @mingwandroid
- @nehaljwani

5.40 3.16.3 (2018-11-21)

5.40.1 Enhancements

- rewrite long prefix paths as $PREFIX, etc. for more readable build logs #3258
- make the --output-folder switch configurable in condarc #3265
- make the --long-test-prefix switch configurable in condarc, fix logic error with that argument #3266

5.40.2 Bug fixes

- improve robustness of indexing in face of corrupt package data #3238
- change timeouts to 900 instead of 90 #3239
- add activation to wheel bundling script #3240
- fix PermissionError import from utils, undefined on py2.7 #3247
- fix outputs with custom build string getting hash incorrectly #3250
- fix tests not being run on windows #3257

5.40.3 Contributors

- @Bezier89
- @gabm
- @isuruf
- @minrk
- @msarahan
- @teake
- @tomashek
- @tschoonj

5.41 3.16.2 (2018-10-29)

5.41.1 Bug fixes

- Remove noarch binary file check (do this in conda-verify instead) #3212
- Fix utf-8 conversion of check_output_env #3213
- fix thread count when indexing causing oversubscription #3217
- fix behavior of try_acquire_locks during lock contention #3224
- fix test env creation improperly prioritizing local channel #3229
- don't check for error when removing conda-init (in conda recipe for this repo) #3230
- add r-impl to R package template generator #3232
- fix creation of unix and win shell script files for noarch packages #3236
- fix path of python interpreter used for noarch packages being tested on win, when created on linux/mac #3236

5.41.2 Contributors

- @alexandersturm
- @Bezier89
- @dsludwig
- @mandeep
- @mingwandroid
- @msarahan
- @rchateauneu
- @soapy1

5.42 3.16.1 (2018-10-12)

5.42.1 Enhancements

- expand ~ in source paths #3206
- Use binsort when available to sort file list in tar archives, to optimize compressibility #3210
- allow meta.yaml's build/rpaths key to function on macOS, not just linux #3206

5.42.2 Bug fixes

- improve docs on behavior of channel arguments #3197
- remove mention in docs about building .RPM and .DEB files. #3199
- fix dist-info errors where dist-info files didn't match the package name #3206
- fix some hard-coded .tar.bz2 references, to support other compression formats more readily #3206
- batch calls to compiling .pyc files to avoid problems with maximum command length #3206
- use ensure_list in processing files to be extracted #3210
- fix KeyError that happened when a jinja2 rendering error occurred, which hid the rendering error 3211

5.42.3 Contributors

- @mingwandroid
- @msarahan
- @stas00
- @teake

5.43 3.16.0 (2018-10-05)

5.43.1 Enhancements

- incorporate libarchive to support more compression formats (adds libarchive as a package dep) #3163
- add build/ignore_verify_codes key to allow recipes to ignore specific conda-verify error codes #3179

5.43.2 Bug fixes

- fix JSON string encoding error in index cache reading #3156
- restore --variants CLI flag for specifying variants #3168
- handle empty build section in output #3175
- add mro-base_impl as something that causes mro build strings #3163
- fix skeleton PyPI inappropriately dropping package case (needed for URLs) #3163
- fix packages from earlier loops with multiple outputs being removed prior to later loops #3185

- fix conda-index not removing entries from index that no longer exist on disk #3186 #3188
- clean up tempfiles after indexing #3187
- fix indexing of specific subdirs #3190
- fix pypi skeleton when python constraint has no operator #3191
- fix issues testing packages and recipe folders when done separately from build #3192
- fix source looking for patches in wrong folder when dealing with outputs #3194

5.43.3 Contributors

- @dpryan79
- @gabm
- @mbargull
- @mingwandroid
- @msarahan
- @nehaljwani

5.44 3.15.1 (2018-09-18)

5.44.1 Bug fixes

- sort "removed" fns in index repodata.json #3154
- fix deps being merged instead of clobbered #3154
- Handle corrupt packages during indexing better #3154

5.44.2 Contributors

- @msarahan

5.45 3.15.0 (2018-09-17)

5.45.1 Enhancements

- add CLI flag (--strict-verify) to allow erroring out when conda-verify fails a package #3135
- output text stating that the license file has been successfully found and included with a package #3152

5.45.2 Bug fixes

- allow channel auth when checking if a package is built #3133
- If local git cache can't be updated, delete it and barf (for user to re-run) #3136
- clean up duplicate pip requirements produced by skeleton #3138
- replace recipe_log.txt file with recipe_log.json file (for passing recipe history along with package) #3139
- fix decoding to str before passing package contents to JSON loading #3140
- fix loss of "removed" section of index with every other indexing operation #3144
- fix update_index used in tests to index channel, not subdir #3145
- fix ELF sections not included in memory image of process being loaded by pyldd and giving misleading results #3148
- fix index operations outputting debug log messages #3151
- fix private channels showing 404 errors during test phase #3153

5.45.3 Contributors

- @Bezier89
- @gabm
- @jakirkham
- @jjhelmus
- @kalefranz
- @msarahan
- @stuarteberg
- @teake

5.46 3.14.4 (2018-09-11)

5.46.1 Bug fixes

- fix recipe_log.txt not being filtered from info/files #3134

5.46.2 Contributors

- @msarahan

5.47 3.14.3 (2018-09-11)

5.47.1 Enhancements

- add support for index patch instructions as tarballs containing subdirs #3129
- add progress bars for indexing (using tqdm) #3130

5.47.2 Bug fixes

- fix log messages being deduplicated too much #3130
- handle permission errors with moving files in indexing more gracefully #3132

5.47.3 Contributors

- @msarahan

5.48 3.14.2 (2018-09-07)

5.48.1 Enhancements

- add support for a "recipe log" file. This will be used at Anaconda to capture the commit activity of a given recipe, which will be published in the RSS feed. #3123

5.48.2 Bug fixes

- fix indexing of noarch subdir as done by conda-forge #3120
- decode cached index files to utf-8 before reading JSON #3121
- try to address unicode problems in run_exports handling #3121
- skip over index metadata files when they are not present in a package #3125

5.48.3 Contributors

- @msarahan

5.49 3.14.1 (2018-09-06)

5.49.1 Bug fixes

- detect and fall back to old update_index behavior (new is channel-wide; old is specific subdir) #3117
- fix CONDA_BUILD_STATE not being set when load_setup_py_data gets run #3117
- fix channel_name as CLI argument for conda index. It can't be positional. #3318

5.49.2 Contributors

- @msarahan

5.50 3.14.0 (2018-09-04)

5.50.1 Enhancements

- refactor indexing to cache more efficiently #3091
- add `tags`, `identifiers`, and `keywords` to about section. Tie them into channeldata.json. #3091
- filter .la files from packages by default #3102
- memoize read_meta_file #3108

5.50.2 Bug fixes

- fix --check for optionally iterable fields #3098
- fix permission problems prior to fixing shebangs #3101
- do not disable pip's cache directory. Redirect it instead. #3104
- fix usage of config in load_setup_py_data #3110
- show logger message when default numpy is used, to communicate what's happening to the user #3110

5.50.3 Other

- drop python 3.4, add 3.6, 3.7 to skeleton pypi #3103

5.50.4 Contributors

- @jjhelmus
- @kalefranz
- @msarahan
- @nehaljwani
- @nsoranzo
- @ocefpaf
- @teake

5.51 3.13.0 (2018-08-20)

5.51.1 Enhancements

- add run_exports and aggregated post-install metadata indexing outputs #3060
- allow whitelisting runpath entries #3072
- consider `*_compiler_version` entries when looping over variants (allow `*_compiler_version` to be a used variable that affects the hash) #3084

5.51.2 Bug fixes

- fix cached git info for variants #3082
- fix linux temporary channel not being added at test time, leaving package unresolvable #3088

5.51.3 Contributors

- @msarahan
- @teake

5.52 3.12.1 (2018-08-06)

5.52.1 Enhancements

- add the "extra" field of a package's meta.yaml file into the output package's info/about.json file #3048
- add option to omit local channel in is_package_build (used by c3i) #3051
- add pip env vars to prevent it from pulling in external dependencies when used in build scripts #3053

5.52.2 Bug fixes

- fix local channel always being top priority. Allow user-defined channel orders where local is lower than remotes. #3049
- Fix conda-verify import error warning showing up in --output text #3052
- fix RPM skeleton test (point to newer CentOS repo) #3054
- fix test/files and test/source_files looking in the wrong place for info/recipe/parent contents (subpackages) #3061

5.52.3 Contributors

- @Bezier89
- @jakirkham
- @mikecormier
- @mingwandroid
- @msarahan

5.53 3.12.0 (2018-07-24)

5.53.1 Enhancements

- Allow user-specified channels to come ahead of local channel #3038
- Add schema for outputs section in FIELDS; provide method for getting rendered recipe text (to support conda-verify) #3041
- Enable conda-verify by default when it is importable, but only print warnings by default, instead of exiting #3042
- Add --label CLI argument to allow specifying label for uploading packages to #3043

5.53.2 Bug fixes

- fix apply_selectors, leading to excessive detection of used variables #3040

5.53.3 Contributors

- @CJ-Wright
- @msarahan
- @speleo3

5.54 3.11.0 (2018-07-20)

5.54.1 Bug fixes

- improve environment marker support for pypi skeleton #2972
- apply selectors before checking requirements, to better understand per-platform used vars #2973
- Handle conda UnsatisfiableError causing packages to be moved to broken folder without tests actually being run on them #2974 #2975
- use tempfiles when writing index to reduce risk of corrupt index #2978
- handle conda index recipe info for older versions of conda-build #2979
- allow empty missing_dso_whitelist in build section #2983
- fix host_reqs referring to a detached list, leading to requirements/host not being modified by run_exports #2987

- fix for bypassing MITM proxies based on SSL_NO_VERIFY env var #2991
- add missing_dso_whitelist to FIELDS #2994
- Don't skip logic in pyldd based on CB verbosity (--quiet) #2999
- Convert empty git refs to HEAD, so that git_url behavior is more predictable #3003
- set NPY_DISTUTILS_APPEND_FLAGS=1 so the compiler package flags are respected #3015
- fix script file renaming when converting package from win to unix #3014
- allow fn to be omitted when using multiple url sources #3021
- fix default config settings being shared across Config instances #3022
- force text interpretation of CRAN DESCRIPTION files #3020
- fix is_no_link to honor patterns #3023
- fix test/requires being ignored when --no-copy-test-source-files is specified #3027
- fix up dependencies in CRAN skeleton output #3030 #3032

5.54.2 Enhancements

- change skeleton pypi to generate recipes that use pip for install step. Remove description. #2972
- Set environment variable to disable pip environment isolation to prevent problems #2972
- support multiple exclusive_config_files #3022

5.54.3 Docs

- Fix search order for conda_build_config.yaml #3029

5.54.4 Contributors

- @isuruf
- @j-hartshorn
- @kalefranz
- @mandeep
- @mbargull
- @mcg1969
- @mingwandroid
- @minrk
- @msarahan
- @nehaljwani
- @ocefpaf
- @tjd2002

5.55 3.10.9 (2018-06-15)

5.55.1 Miscellany

- docs are moving from the conda-docs repo into conda-build (splitting off from conda docs in general)

5.55.2 Bug fixes

- fix re.escape usage for Python <3.3 #2970

5.55.3 Contributors

- @mbargull
- @msarahan

5.56 3.10.8 (2018-06-12)

5.56.1 Bug fixes

- clean up license field for pypi skeleton #2938
- fix regex to match requirements with trailing spaces #2948
- Check for dash in text with variants #2949
- do not check in build prefix for overlinking when merging build and host #2950
- utils.glob: remove unnecessary normcas, fix test_expand_globs #2952
- add missing "build" fields in FIELDS schema thing #2962

5.56.2 Contributors

- @isuruf
- @mariusvniekerk
- @mbargull
- @mingwandroid

5.57 3.10.7 (2018-06-01)

5.57.1 Enhancements

- replace glob2 by glob for py3 (speed improvement) #2937

5.57.2 Bug fixes

- Fix folder copying in test/files #2941

5.57.3 Contributors

- @mbargull
- @msarahan

5.58 3.10.6 (2018-05-31)

5.58.1 Enhancements

- several rendering speed improvements #2925

5.58.2 Bug fixes

- add regression test for #2912 #2914
- fix a file handle not being closed #2915
- fix an incorrect syntax RuntimeError #2920
- fix custom compiler languages not being possible #2927
- fix OS vars not correctly respecting test prefix; add test #2932
- fix incorrect python versions showing up in test phase paths (SP_DIR) #2932
- fix test/files functionality for outputs; add test #2934

5.58.3 Contributors

- @mbargull
- @msarahan
- @nicoddemus
- @rainwoodman
- @sodre
- @tomashek

5.59 3.10.5 (2018-05-23)

5.59.1 Enhancements

- allow '*' as an ignore_run_exports value to ignore all run_exports #2907

5.59.2 Bug fixes

- fix handling of empty run and test requirements #2908
- fix trailing zeroes in version numbers getting lost by yaml interpreting things as floats #2909
- fix regression in host prefix showing up in the test phase, leading to files/executables not being where they should be #2910
- fix handling of not-yet-available requirements #2912
- fix get_value with default keyword not respecting that user-specified default #2912

5.59.3 Contributors

- @msarahan

5.60 3.10.4 (2018-05-20)

5.60.1 Bug fixes

- fix import tests being run multiple times #2892
- add creative commons as a license family (used to be classified OTHER) #2893
- handle empty packages in checks for duplicated files across subpackages #2894
- set PYTHON and other language path vars based on presence in build/host reqs, rather than binary file in either env. Allows usage of PYTHON and friends in meta.yaml vars. #2895
- fix entry points incorrecty pointing at build prefix (instead of host), leading to prefix replacement failing #2895
- fix merge_build_host functionality. Adding an empty host section now forces build and host to be split. #2896

5.60.2 Contributors

- @msarahan
- @scopatz

5.61 3.10.3 (2018-05-17)

5.61.1 Enhancements

- --skip-existing applies to outputs, not just whole collections of packages. Individual outputs that are already built will be skipped. #2889
- add output of hash contents to what gets printed with conda render (not with --output)

5.61.2 Bug fixes

- fix conda pypi skeleton checking for '~' in None values #2880
- add /B to win exits, so that erroring out of tests does not close out of outer shells #2881
- ensure that merge_build_host is updated correctly for each output #2882
- Remove several env vars from being recorded in about.json, over concerns for GDPR compliance #2883
- remove parent_recipe entry from recipes when recording meta.yaml for output packages, to avoid confusion over used variables #2886
- xfail get_installed_version for new conda and test_build_with_activate_does_activate when PATH is too long #2889
- change os.rename to shutil.move so that there is a copy fallback #2889
- fix mutability of config objects passed to test causing bizarre states of variants
- fix win style slashes in paths.json and `files` that broke things when converting a win package to unix #2891

5.61.3 Contributors

- @mingwandroid
- @msarahan

5.62 3.10.2 (2018-05-08)

5.62.1 Bug fixes

- fix downstream test not using channel list; fix exact specs in downstream testing #2864
- add deprecation notice for msvc_compiler key in meta.yaml. Explain its incompatibility with variants. #2868
- set default cran mirror #2868
- disallow merging of build and host prefixes when host_subdir != build_subdir #2876

5.62.2 Contributors

- @msarahan

5.63 3.10.1 (2018-05-01)

5.63.1 Bug fixes

- fix handling of downstream testing when downstreams don't exist yet (e.g. bootstrapping) #2860
- fix handling of downstream testing in tandem with --output-dir or --croot (add locations as file:// urls) #2860
- fix improperly escaped entries in cran template. Clean up unnecessary changes. #2861

5.63.2 Contributors

- @mingwandroid
- @msarahan

5.64 3.10.0 (2018-05-01)

5.64.1 Enhancements

- Warn user about path conflicts during environment building for test phase #2843
- Add conda 4.6 compatibility #2844
- **remove conda 4.2 and earlier compatibility** #2845
- add info to merge/copy source subdir error #2858
- Add setup for Air Speed Velocity benchmarking #2859

5.64.2 Bug fixes

- fix python handling when python is a tuple (inner python looping) #2854
- fix python not looping in inner packages when top-level doesn't use it. Fix zip_keys handling. #2856

5.64.3 Contributors

- @kalefranz
- @msarahan

5.65 3.9.2 (2018-04-27)

5.65.1 Enhancements

- Optimizations to rendering to speed up dealing with lots of recipes #2838 #2848

5.65.2 Bug fixes

- include `folder` as a field in `source` for linting purposes #2837
- remove merging of constraints. Keep only the clobbering of groups of constraints by exact constraints (of which you can have only one) #2839
- ensure u+w permissions before calling install_name_tool #2840
- remove conversion of dash to underscore in pin_run_as_build #2842

5.65.3 Contributors

- @jakirkham
- @mingwandroid
- @msarahan

5.66 3.9.1 (2018-04-24)

5.66.1 Bug fixes

- Revert #2831 (add license file for R packages from CRAN) due to incompatibility with package layout in defaults
- handle OrderedDict dumping to yaml better; further work on preserving dict key order in config yaml files #2834
- consolidate cran default repo settings, respect variant setting better. #2836
- Add conda-build/skeleton_config_yaml key to condarc to control which conda_build_config.yaml should be used to find the cran_mirror setting. #2836
- Change default cran mirror from mran to cran. #2836

5.66.2 Contributors

- @mingwandroid
- @msarahan

5.67 3.9.0 (2018-04-24)

5.67.1 Enhancements

- Add new key in test section, `downstreams` that accepts a list of package specs to test after the current package is built #2823
- work to prevent unsafe paths in tarballs that would affect paths outside of the work dir #2822
- simplify all constraints for a given package name to a single constraint that represents the tightest combination of them all #2694

5.67.2 Bug fixes

- fix a misnamed cran skeleton key #2817
- Remove unused index command in rendering path #2818
- fix loss of ordering when using recipe_append #2825
- fix usage of dict for default pin_run_as_build data structure. Losing ordering created noise down the line for Conda-Forge. #2830
- fix selector regex being too greedy; reporting wrong used vars #2832

5.67.3 Contributors

- @ceball
- @isuruf
- @jamesabbott
- @jdblischak
- @mingwandroid
- @msarahan

5.68 3.8.1 (2018-04-16)

5.68.1 Bug fixes

- fix shebang rewriting so that it only touches python[w]? shebangs #2786
- fix a regression in ignoring python as a build-only dep being "used" and becoming a loop var #2799
- improve config log warning #2801
- skip, but warn about failures in pyldd #2802
- fix whitespace in multi-line help strings #2808
- fix variables in compound selectors not getting detected as "used" #2814

5.68.2 Contributors

- @bjlittle
- @jbcrail
- @mingwandroid
- @msarahan

5.69 3.8.0 (2018-03-30)

5.69.1 Enhancements

- Add new jinja2 function, resolved_packages, that can be used to pin run or test requirements to indirect dependencies as well as direct dependencies #2715

5.69.2 Bug fixes

- Fix R/Rscript mixup that broke usage of R env var #2782
- Improve error message when additional modules are needed in root env in order to render a recipe #2784
- Fix handling of FEATURE_NOARCH, which was adding specs that conda's solver didn't understand #2787
- allow license_file to be found in either source root or recipe root (common point of confusion) #2792
- fix disable_pip removing setuptools even when it was an explicit dependency. This was due to conda changing its string representation of MatchSpecs, and our regex didn't take that into account. #2793

5.69.3 Contributors

- @183amir
- @msarahan

5.70 3.7.2 (2018-03-22)

5.70.1 Enhancements

- add runpath handling to pyldd #2768
- add lgtm.com configuration #2772

5.70.2 Bug fixes

- fix language issues with finding directory size on windows #2749 #2766 #2760
- ignore non-rendered jinja2 errors when indexing packages #2756
- fix cran skeleton argparse errors when version flag not provided #2751 #2759
- fix exact pinning in subpackages raising errors due to non-final output data conflicting with final top-level data #2763
- skip test_preferred_env until conda more fully implements it #2722
- Don't run mk_relative_osx on linux DSO's #2768
- use Rscript to run R tests, so that console output is shown more clearly. Only add r-base spec if neither r-base nor mro-base are already in deps. #2764
- don't filter out .gitignore and .gitmodules when packaging #2774
- fix pin_* regex that was erroneously picking up wrong usages #2775

5.70.3 Contributors

- @bilderbuchi
- @kfranz
- @m-rossi
- @mingwandroid
- @msarahan
- @wikiped

5.71 3.7.1 (2018-03-08)

5.71.1 Enhancements

- Enable glob2.fnmatch for shared library whitelists. Add more Frameworks to whitelist on Mac. #2732

5.71.2 Bug fixes

- Squelch yaml ScannerError when building index can't read meta.yaml in package #2740
- Fix & simplify "hoisting" of source folders up one level #2741
- Fix build number not getting inherited from top-level metadata #2742
- Allow output creation environment for wheels to be activated #2744
- Fix selector regex for finding "used" variables; was finding too much across lines. #2745
- Ignore empty config files (don't error out on them) #2746

5.71.3 Contributors

- @mingwandroid
- @msarahan
- @neok-m4700

5.72 3.7.0 (2018-03-05)

5.72.1 Enhancements

- raise ValueError when pin_subpackage is used, but no matching output is found #2720
- Add new optional CLI argument, --extra-deps, to add test-time dependencies dynamically when splitting build and test phases (can't apply variants when phases are split) #2724

5.72.2 Bug fixes

- fix cran skeleton py2 invalid list copy syntax #2720
- reconfigure TravisCI to test against conda master #2720
- fix inaccurately raised problems with pin_subpackage #2720
- coerce boolean values in conda_build_config.yaml to booleans (value.lower() == "true") #2723
- change r skeleton cran test to a different package (nmf -> acs); nmf got removed
- fix selectors being applied before variable detection, leading to variables in selectors never being detected #2723
- add filesize calculation to converted script files #2727

5.72.3 Contributors

- @mandeep
- @msarahan

5.73 3.6.0 (2018-02-28)

5.73.1 Enhancements

- Allow per-output {pre,post}-{un,}link scripts #2712
- support mro as part of the build string #2711
- improve interpreter guessing for running output packaging scripts #2709
- improve library overlinkage check, add support for whitelists of always-ok libraries to ignore. #2708
- add support for noarch: generic recipes in cran skeleton generator
- add support for using Rtools on windows when building a package from source
- add support for binary repackaging of CRAN/MRAN artifacts

- add support for cran recipes from source tarballs
- template cran_mirror variable in generated cran output recipes. This allows CRAN and MRAN to easily be switched out. Default is MRAN. #2710

5.73.2 Bug fixes

- Reverse build/host activation order, to give build executables higher priority. Necessary to support proper R packaging. Includes better errors for empty packages caused by build env python being used to install python packages. #2686
- Fix test scripts from subpackage outputs not being detected #2703
- Fix sha in scripts in conversion from linux to windows packages (was not correctly handling hashbang addition/removal). #2713
- Speed up stat gathering, restrict it more to only build, packaging, and test steps (not arbitrary subprocess calls) #2714
- Check for incomplete files when inspecting links. Some files that looked like ELF files, but weren't, would trip up pyldd and kill the build. #2718

5.73.3 Contributors

- @jjhelmus
- @MatthieuDartiailh
- @mingwandroid
- @msarahan

5.74 3.5.1 (2018-02-22)

5.74.1 Enhancements

- Add relative path support for load_setup_py_data jinja2 function #2700

5.74.2 Bug fixes

- fix hoisting of archives containing folders named same as top-level folder. These subfolders were being removed inappropriately. #2692
- Fall back gracefully when psutil fails to import. Disk and total time stats still available; memory and CPU time are not when psutil is unavailable. #2693
- Fix directory size computation not being recursive, use scandir for walk operations on py27 #2699

5.74.3 Contributors

- @mariusvniekerk
- @msarahan

5.75 3.5.0 (2018-02-20)

5.75.1 Enhancements

- Print resource statistics for each step, as well as whole. CPU time, memory usage, disk usage. #2685
- Record resource statistics to JSON file when `--stats-file <output_file_path>` argument is provided #2685

5.75.2 Bug fixes

- save complete parent recipe in info/recipe/parent for packages that are only outputs of a top-level package #2687

5.75.3 Contributors

- @msarahan

5.76 3.4.2 (2018-02-15)

5.76.1 Enhancements

- allow trimming of skipped metadata in rendering to be optional (for sake of conda-forge rendering readme's on platforms that are skipped) #2680
- rename the build prefix prior to the test phase. This will precipitate failures when packages embed paths to the build prefix, which conda does not replace at install time. Fixing these instances is specific enough to packages that we do not attempt to handle it in conda-build. #2681
- add conda_interface.get_install_version function that facilitates checking if a pkg is in an env, and if so, what its version is #2682

5.76.2 Bug fixes

- use lookaheads in extraction regexes to avoid capturing unwanted text #2679

5.76.3 Contributors

- @msarahan

5.77 3.4.1 (2018-02-08)

5.77.1 Bug fixes

- fix interpretation of zip_keys when testing pkgs (ignore empty values) #2673

5.77.2 Contributors

- @msarahan

5.78 3.4.0 (2018-01-31)

5.78.1 Enhancements

- implement "--exclusive-config-file" CLI flag to render & build. This file bypasses detection of config files in $HOME and cwd, but respects any config files in recipe folders. #2661
- Activate output scripts in meta.yaml (#2667), but only when: * output has a build/script entry * output uses `{{ compiler() }}` jinja2 function in its requirements AND output extension is either .sh or .bat * output has `build/activate_in_script` key in meta.yaml set to a truthy value AND output extension is either .sh or .bat

5.78.2 Bug fixes

- fix AttributeError in overlinking check #2650 #2651
- reorder mmap operations to fix problem with WSL #2655
- fix numpy detection as "used" variable when using pin_compatible jinja2 #2659
- silence conda KeyError warnings when indexing legacy packages that don't have newer metadata files #2656
- replace "which" with "type -P" in conda-build's internal recipe. This avoids issues on PowerPC and with long paths. #2664
- Error out when version computation fails in conda-build's setup.py. This will help prevent conda-build packages going out without valid internal versions being recorded (for example, when git is not installed on a build worker). #2665
- ignore tarcheck errors for files in the info folder that don't appear in info/files file. Fixes inclusion of arbitrarily named readme files. #2668
- clean up host prefix in between skeletons when using pypi's --recursive mode. Conda otherwise throws errors on the 2nd recipe. #2669

5.78.3 Contributors

- @kfranz
- @mingwandroid
- @msarahan
- @nehaljwani
- @neok-m4700
- @steamelephant

5.79 3.3.0 (2018-01-23)

5.79.1 Enhancements

- Issue template created for github repo #2632
- Detect overlinking (usage of libraries that are not present in listed dependencies). Warn by default. Error out with --error-overlinking flag. Conda-build 4.0 will error by default. #2635 #2647

5.79.2 Bug fixes

- fix merge_build_host to always be used in CRAN skeletons #2635
- fix macho filename attribute error #2641
- reorder search through files for compatibility bounds for speed #2638
- cache used vars based also on recipe path, to avoid overly caching results #2643
- normalize slashes in file glob lists for explicit output file lists #2644
- silence conda 4.4 better when using quiet operations, such as --output #2645
- fix pypi_url not affecting the url of the actual skeleton output from conda skeleton pypi #2646
- fix overly broad string matching of "None" that caused problems where None may appear as part of a string in meta.yaml #2649

5.79.3 Contributors

- @csoja
- @mingwandroid
- @msarahan
- @nehaljwani
- @neok-m4700

5.80 3.2.2 (2018-01-12)

5.80.1 Enhancements

- Add CLI flag (--merge-build-host) to restore pre-3.1.4 behavior with merging build and host envs #26260

5.80.2 Bug fixes

- Check recipe/metadata skip status in more places, rather than strictly at the top-level #2617
- fix unnecessary conforming of zip keys when distributing variants #2618
- fix matching of unrendered output names when matching rendered names #2618
- fix matching of partial (only used parts) of variants when lining up subpackages #2618
- fix handling of outputs with same name as top level when considering used vars #2618
- exclude top-level run_exports from applying to all outputs #2618
- Fix linking compiler runtimes from build to host prefix (was broken by build/host prefix split in 3.1.4) #2621
- Fix logic errors around merging build/host envs #2623
- fix run_exports in outputs being overwritten #2623

5.80.3 Contributors

- @jjhelmus
- @mingwandroid
- @msarahan

5.81 3.2.1 (2018-01-02)

5.81.1 Enhancements

- Improve "BUILD" environment variable value (especially on powerpc) #2615
- Implement CentOS 7 ppc64le distro for conda skeleton rpm #2615
- Improve handling of outputs that use the build/skip key to skip building #2616

5.81.2 Bug fixes

- Don't loop in all zipped keys when collecting used vars. Leave it to consumers to decide what to do. #2612
- Fix run_exports using pin_subpackage not applying versioning for the implicit top-level output #2613
- Fix run_exports not applying to build-time environment creation for top-level recipe (as opposed to outputs) #2613
- Fix CRAN skeleton to better use host/build envs appropriately #2614

- fix outputs not loading hash input info from files at test time correctly, leading to incorrect hashes and unresolved packages. #2616

5.81.3 Contributors

- @mingwandroid
- @msarahan

5.82 3.2.0 (2017-12-21)

This release bumps the minor version to reflect the change in splitting the build and host folders originally introduced by 3.1.4. That change has proven to be disruptive to many users, and we felt it necessary to bump a minor version to indicate that people should pay attention to this change. There's more info in our docs at https://conda.io/docs/user-guide/tasks/build-packages/define-metadata.html#host

5.82.1 Enhancements

- Add log messages for each source of variants, so that you know where values are coming from #2597

5.82.2 Bug fixes

- remove unnecessary looped rm_rf when cleaning out prefixes between outputs #2587
- fix meta.yaml not found errors when trying to test packages built with --no-include-recipe #2590
- fix zipped key group with single entry causing a list to be passed later for single string values #2596
- fix incomplete change to splitting build and host envs #2595
- fix merging of top-level requirements and output reqs when output named same as top-level #2595
- fix handling of outputs with templates in their name (they were losing their requirements) #2595
- fix test file copying to avoid re-provisioning source during tests #2595
- tweak requirements regex to avoid misinterpreting python executable usage in test commands as usage of the python variant #2595

5.82.3 Contributors

- @msarahan

5.83 3.1.6 (2017-12-15)

5.83.1 Bug fixes

- fix test files in outputs (was losing reference to absolute path of recipe) #2584
- fix several incorrect references to build_prefix that needed to be host_prefix #2584

5.83.2 Contributors

- @msarahan

5.84 3.1.5 (2017-12-15)

5.84.1 Enhancements

- detect "used" variables in selectors #2581

5.84.2 Bug fixes

- Cache used variables for a given output on a given target platform to avoid recomputing this too often. This dramatically speeds up operations relative to 3.1.4. #2581
- fix used variable treatment of target_platform #2581

5.84.3 Contributors

- @msarahan

5.85 3.1.4 (2017-12-14)

5.85.1 Enhancements

- detect "used" variables in build.sh, bld.bat and any output scripts, in addition to what already existed in meta.yaml. Used variables end up in the hash. #2576
- don't merge build and host prefixes. We used to do this when host subdir == build subdir. Keep them separate, so that build tools in build prefix won't ever interfere with software installed to host, to be packaged. #2579

5.85.2 Bug fixes

- exclude grouped keys from zip_keys when computing hashes. Only direct dependencies affect the hash. #2573
- fix always_include_files usage omitting other ordinarily installed files #2580

5.85.3 Contributors

- @msarahan

5.86 3.1.3 (2017-12-13)

5.86.1 Enhancements

- support environment variable expansion in path-related condarc settings #2563
- speed up "fixing linking" on MacOS by ~98% #2564
- Allow build/script and build/script_env entries in outputs, for simple scripts and for passing env vars into output scripts #2572

5.86.2 Bug fixes

- fix run_exports from build section not applying to host early enough and causing conflicts #2560
- order outputs based on build, host, and run dependencies, not just run #2561
- fix always_include_files when used in output sections #2569
- add jinja2 to dependencies in setup.py (not just in conda.recipe) #2570

5.86.3 Contributors

- @akovner
- @mingwandroid
- @msarahan
- @nehaljwani
- @rlizzo

2017-12-9 3.1.2:

5.86.4 Bug fixes

- fix copying of relative paths with source_files in test section #2551
- fix handling of too many x's in max_pin field. If more x's than actual places were present, the incrementing broke. #2552
- refactor upstream pinning, fix extraction of outputs so that run_exports and pin_compatible work with them #2556
- fix bug that occurred when an output had the same name as the top level recipe. Ended up extracting wrong part of recipe with wrong regex. #2556
- fix copying of recipe losing folder structure in the destination copy of the recipe #2557

5.86.5 Contributors

- @msarahan
- @nehaljwani

5.87 3.1.1 (2017-12-06)

5.87.1 Bug fixes

- fix info files filters on windows #2542
- fix icon.png files that needed to be included in the app section of recipes, for usage with Navigator #2545
- fix package matching regex for packages with - in them (regex should find either - or _) #2546
- fix detection of used variant variables within jinja2 conditionals #2547
- fix output extraction regex (was picking up whole outputs section, not just one output). Also, fix top-level variables not being carried into later outputs. #2549

5.87.2 Contributors

- @jjhelmus
- @msarahan

5.88 3.1.0 (2017-12-05)

5.88.1 Enhancements

- Speed up package inspection by optimizing package file lookup #2535
- Simplify hashing scheme. A hash will be added if all of these are true for any dependency:
  - package is an explicit dependency in build, host, or run deps
  - package has a matching entry in conda_build_config.yaml which is a pin to a specific version, not a lower bound

- that package is not ignored by ignore_version

OR

- package uses `{{ compiler() }}` jinja2 function

All other packages will no longer have hashes. The takeaway message is that hashes will appear when binary compatibility matters, but not when it doesn't. #2537

5.88.2 Bug fixes

- Allow packages to store files in info folder #2538
- Fix source_files not working correctly when using test files in packages #2539

5.88.3 Contributors

- @mingwandroid
- @msarahan

5.89 3.0.31 (2017-11-30)

5.89.1 Enhancements

- expose time and datetime modules in jinja2 context, for use in meta.yaml #2513
- jinja: permit recipes to check for existence of a variable without erroring #2529
- add method for getting all variant values used by a recipe, not just those variants with more than one value. #2531

5.89.2 Bug fixes

- allow SSL_NO_PROXY env var to disable SSL checking on proxied connections #2505
- Fix source hoisting issues (incorrectly flattening directory structure of extracted archives) #2507
- Fix build env for output getting lost when output name == top-level name #2511
- add global pin_run_as_build for R (x.x.x) to keep legacy R pinning behavior #2518
- Fix path conversion issues going from windows to unix #2522
- only insert variant versions when testing runtime availability for packages that are also present in build (not just run) #2527

5.89.3 Contributors

- @anton-malakhov
- @bheklilr
- @mandeep
- @msarahan
- @stuarteberg

5.90 3.0.30 (2017-11-15)

5.90.1 Bug fixes

- write all 'about' metadata fields out, not just select few #2488
- fix lists getting nested during merging of configs, leading to TypeErrors #2494
- make always_include_files act on host_prefix, not build_prefix #2497
- warn users when script_env passes env vars through #2502
- fix build string pyXY being just pyX when input didn't have full python version #2504

5.90.2 Contributors

- @jakirkham
- @msarahan

5.91 3.0.29 (2017-11-10)

5.91.1 Enhancements

- interpret ~= in pypi skeletons, map to compatible expressions #2427
- add arm and ppc architectures to conda convert #2472, #2474
- add indentation to index.json and hash_input.json for easier reading #2476
- check arch in index.json for platforms other than linux, mac, win #2478
- update cran skeletonizer for new compilers, add flags for updating, rather than replacing recipe. #2481

5.91.2 Bug fixes

- fix implicit pinning not taking effect in outputs, fix incorrect matching of hashed subpackages #2455
- exclude python from build requirements for purposes of hash computation. This was causing recipes that used python as a build tool to build too many similar packages. #2455
- Support GIT_* vars even when source folders are specified #2477
- silence warnings about `.*` being added to vc deps #2483
- fix non-finalized recipe being used for creating build env, resulting in too few variants in output #2485

5.91.3 Contributors

- @mandeep
- @mingwandroid
- @msarahan
- @stuarteberg

5.92 3.0.28 (2017-11-02)

5.92.1 Enhancements

- Implement "subspace selection" - so you can reduce a larger central set of variants to some smaller subset. Fixes --python=X.Y on windows, with its zip_keys. #2466
- Update cpan skeleton #2156
- Pass through VSXY0COMNTOOLS env vars, so they're available in activate scripts called from outputs #2453
- Add additional index-related files for Anaconda Navigator to use #2463
- Add back CONDA_PY, CONDA_NPY, and friends, for backcompat with conda-build-all #2469

5.92.2 Bug fixes

- Fix build_folder selection in dirty envs #2445
- Fix an os.rename back to copy_into for cross-volume compatibility #2451
- Clean up leftovers created by utils.get_recipe_abspath #2459
- fix path globbing and filtering replacing prefix not at start of path, which broke file copying #2468
- Don't recreate envs unnecessarily for recipes with no outputs section #2470

5.92.3 Contributors

- @jerowe
- @kalefranz
- @msarahan
- @neok-m4700
- @rendinam

5.93 3.0.27 (2017-10-17)

5.93.1 Enhancements

- For windows error checks, assert that the errorlevel is 0, rather than GEQ 1. Makes negative return codes fail builds. #2442
- allow channels to be passed to the metapackage command. Note that channels are not recorded to the package, and need to be passed at package install time, as well as metapackage creation time. #2443

5.93.2 Bug fixes

- Fix windows bits dictionary indexing incorrect type #2441

5.93.3 Contributors

- @msarahan

5.94 3.0.26 (2017-10-16)

5.94.1 Enhancements

- Conda index now generates html index pages in addition to repodata.json #2395
- make bash verbosity (-x flag) depend on setting of --debug flag #2426
- pass test and build sections in any outputs through wholesale, rather than picking out individual fields from them. #2429
- make conda-verify opt-in, rather than opt-out. Use `--verify` cli argument or `verify` keyword to api. #2436
- implement requires_features and provides_features, for compatibility with conda 4.4's new key-value feature #2440

5.94.2 Bug fixes

- fix `FEATURE_*` variables not working due to a type error #2428
- fix misleading error when download_url present but empty #2434
- check HTTP status code of PyPI pkg manifest request before decoding it, to improve error message #2435
- fix 64-bit exe's showing up in 32-bit win packages due to not accounting for host_arch with script files #2439
- fix hardlink-breaking bug where path was being copied instead of specific file. Use better tempdir. #2437

5.94.3 Contributors

- @Bezier89
- @eklitzke
- @kalefranz
- @maddenp
- @msarahan
- @nehaljwani

5.95 3.0.25 (2017-10-06)

5.95.1 Bug fixes

- unify usage of conda_43, learn to let the tests run. #2424

5.95.2 Contributors

- @msarahan

5.96 3.0.24 (2017-10-06)

5.96.1 Enhancements

- add get_used_loop_vars() function to MetaData object, to show which loop variables are actually used by recipe #2410
- Allow multiple file extensions for pypi skeletons, not just .tar.gz #2412

5.96.2 Bug fixes

- make build reqs equivalent to host when cross-compiling and no host section present (helps reduce need to modify python-only recipes) #2406
- reduce logging output from filelock and conda #2418 #2422
- Don't strip files in noarch: python when they are not known file types #2420
- fix infinite loop when trying to build dep from found recipe, but that recipe is wrong version #2423
- update perl used on appveyor for testing to 5.26

5.96.3 Contributors

- @minrk
- @msarahan
- @nehaljwani

5.97 3.0.23 (2017-09-29)

5.97.1 Bug fixes

- simplify handling of blank fields in CRAN metadata #2393
- load conda_build_config.yaml from inside package when testing package separately from build process #2399
- use sets instead of lists for field descriptions #2403
- fix noarch_python packages getting pinned to a specific python version #2409

5.97.2 Contributors

- @Bezier89
- @CJ-Wright
- @jdblischak
- @msarahan

5.98 3.0.22 (2017-09-20)

5.98.1 Bug fixes

- fix filename_hashing setting being ignored when using conda-build API #2385
- fix relpath causing cross-drive issues on windows #2386
- examine .a files when considering prefix replacement #2390
- fix run/test deps check looking at build_subdir rather than host_subdir (broke cross compiling) #2391

5.98.2 Contributors

- @Bezier89
- @mingwandroid
- @msarahan

5.99 3.0.21 (2017-09-18)

5.99.1 Bug fixes

- Fix strong run_exports from build being applied to host too late, running into conflicts (especially with VC features) #2383
- crash properly when patching fails, rather than proceeding with build #2384

5.99.2 Contributors

- @msarahan

5.100 3.0.20 (2017-09-16)

5.100.1 Bug fixes

- Never activate build or host env when building conda, so that symlinks or .bat files are never created. #2381
- Apply "strong" run_exports from build section to host section, not just run section. This is necessary for ensuring that features activated by packages in the build section are used to line up the host section also. #2382

5.100.2 Contributors

- @msarahan

5.101 3.0.19 (2017-09-15)

5.101.1 Bug fixes

- write info/files for noarch. Always sanity check info/files. #2379
- fix build_prefix -> host_prefix in write_pth, fixes cross compiling python packages #2380

5.101.2 Contributors

- @Bezier89
- @msarahan

5.102 3.0.18 (2017-09-14)

5.102.1 Bug fixes

- fix source hash not being verified #2367
- fix several references to arch that should be host_arch to support cross compiling (win-32 on win-64, for example) #2369, #2368
- replace recipe run requirements with contents of index.json's "depends" when testing packages #2370
- update R and perl versions in DEFAULT_VARIANTS #2373
- fix versioneer showing unknown version on windows due to --match argument #2375
- add subdir to moved work folder dirname, to avoid clobbering when cross compiling #2376

5.102.2 Contributors

- @jjhelmus
- @mingwandroid
- @msarahan

5.103 3.0.17 (2017-09-12)

5.103.1 Enhancements

- add track_features and features to output options, to allow per-output configuration of features #2358

5.103.2 Bug fixes

- Fix conda symlinks misbehaving when building conda package #2359

5.103.3 Contributors

- @msarahan

5.104 3.0.16 (2017-09-12)

5.104.1 Enhancements

- allow env check to be bypassed when rendering (for c3i) #2353
- provide mechanism for compiler version to be passed to compiler jinja2 function (match name with _version) #2356

5.104.2 Bug fixes

- use host_subdir instead of build_subdir when setting selectors #2345
- remove downloaded files from source cache if they failed at any download step #2349
- fix variants being merged across multiple builds due to modification of global #2350
- disable pyldd disagrees warning output for now #2352

5.104.3 Contributors

- @mingwandroid
- @msarahan

5.105 3.0.15 (2017-09-04)

5.105.1 Bug fixes

- fix relative paths for croot argument to CLI; test #2335
- fix several issues with zip_keys #2340
- fix output build number never applying #2340
- fix ensure_matching_hashes for strong/weak run_exports #2340
- fix indexing of channels, especially before testing packages #2341
- copy wheels and unextractable files (.sh) into the workdir with their original, un-hashed filename, for simplicity in working with them. #2343
- avoid attempting to overwrite existing files in the source cache #2343
- avoid unsatisfiable requirement errors by adding .* to specs that lack .* or >/</>=/<= #2344

5.105.2 Contributors

- @gabm
- @msarahan

5.106 3.0.14 (2017-08-29)

5.106.1 Bug fixes

- fix config.arch comparison being wrong data type #2325
- fix run_exports handling with dict of lists #2325
- pyldd: disambiguate java .class files from Mach-O fat files (same magic number) #2328
- fix hash regex for downloaded files in src_cache #2330
- fix zip_keys becoming a loop dimension when variants passed as object rather than loaded from file #2333
- fix windows always warning about old compiler activation. Now only warns if {{ compiler() }} is not used. #2333
- Add LD_RUN_PATH back into Linux variables for now (may remove later, but will have deprecation cycle) #2334

5.106.2 Contributors

- @mingwandroid
- @msarahan
- @neok-m4700

5.107 3.0.13 (2017-08-26)

5.107.1 Enhancements

- allow output build number and string to be set independently of top-level metadata #2311
- add file hash to source cache filenames to avoid collisions #2312
- add notion of "strong" or "weak" run exports. Strong apply to run whether parent is in build or host. Weak apply only if in host, or in build with no host present. #2320

5.107.2 Bug fixes

- Fix PY3K value changing from 0/1 to True/False. Keep 0/1.
- make work_dir the cwd when running output bundling scripts. It was the host prefix before now.
- start tmpdir prefixes when getting dependency versions with _ so that conda can be one of the deps #2321
- avoid setting empty compiler variables #2322
- remove meaningless error with glob_files and always_include_files during env creation #2323

5.107.3 Contributors

- @msarahan

5.108 3.0.12 (2017-08-23)

5.108.1 Enhancements

- update default MACOSX_DEPLOYMENT_TARGET to 10.9 #2293
- modernize pin_depends so that it works with conda render #2294
- environment variable pass-throughs now respect variant (env var highest priority; variant, finally default) #2310

5.108.2 Bug fixes

- fix run_exports getting picked up transitively #2298
- fix default compiler not having platform #2300
- fix CONDA_PY formatting (should not have period). PY_VER does have period. #2304
- update index before testing a package, so that that package is conda-installable. #2308
- update index after moving a package after test failure, so that the index is current. #2308
- fix --output-folder not being respected by --output preview of output path #2309

5.108.3 Contributors

- @mingwandroid
- @msarahan

5.109 3.0.11 (2017-08-17)

5.109.1 Enhancements

- set BUILD environment variable (triplet used by cross-compiling) #2285
- respect condarc cache_dir setting for changing the source cache dir location #2278
- run selectors before returning meta.yaml extractions #2284

5.109.2 Bug fixes

- fix CRAN skeleton field truncation with ; characters #2274
- Warn about overlapping files in subpackages within a recipe #2275
- fix --override-channels not taking effect #2277
- fix double-activation on Windows for cross compiling #2280
- fix variant entry duplication with zipped keys #2280
- fix folder hoisting when folder name in archive matches package name #2281
- fix test env showing old cached packages when test env has actually been removed #2282
- fix source code not being present for render when source necessary for render and more than one variant #2283
- fix binary_relocation not allowing lists of files #2288
- fix incorrect python (or none at all) being used for pyc compilation with python only in host reqs #2290

5.109.3 Contributors

- @dsludwig
- @jdblischak
- @jjhelmus
- @mingwandroid
- @msarahan

5.110 3.0.10 (2017-08-11)

5.110.1 Enhancements

- Provide variant variables for use in selector expressions #2258

5.110.2 Bug fixes

- fix ordering of recipe elements in skeletonized pypi recipes #2230
- Trim empty variant sections (due to selectors) prior to zipping keys #2258
- Don't set blank env vars in build scripts #2259
- Fix testing with recipe paths #2262
- add newlines to test scripts #2265
- Fix render command not considering provided channels #2267
- fix get_value being hardcoded to only first entry #2268
- fix setting target (target platform) in output section causing tarcheck validation error #2271
- don't add setuptools to runtime dependencies in skeletonized pypi recipes (only build) #2272

5.110.3 Contributors

- @chaubold
- @msarahan
- @mwcraig
- @neok-m4700
- @ratstache
- @stuarteberg

5.111 3.0.9 (2017-08-02)

5.111.1 Enhancements

- store test files specifed by test/source_files directly in packages. This allows testing of packages that do not include recipes. Recommendation: make subpackages for large data files. #2232
- add new syntax to get_value for accessing list items, such as multiple sources #2247
- add independently configurable source cache path (--cache-dir) #2249
- add PKG_HASH env var, available in meta.yaml. Use this to put the package hash where you want it in your custom build/string field in meta.yaml. #2250

5.111.2 Bug fixes

- Fix test python using incorrect metadata config object, and then using wrong prefix #2226
- Allow testing multiple conda packages or folders at once with wildcard CLI arguments #2227
- Fallback path for ruamel_yaml to ruamel.yaml #2233
- raise exception when both build/script in meta.yaml and build script (build.sh/bld.bat) are provided #2238
- Fix unclosed file handle when loading setup.py data #2242
- Fix 'path' source with multiple source #2247

- improve compatibility with conda 4.4 #2248
- remove hash from manually-specified build/string fields. Use new PKG_HASH env var instead. #2250
- fix windows activate scripts getting included in windows packages #2251
- ignore feature records in index for 'conda inspect' #2253
- fix variant handling when variants affect the downloaded source (need re-extract, re-parse with new source at each step) #2254

5.111.3 Contributors

- @Bezier89
- @jjhelmus
- @kalefranz
- @msarahan
- @mandeep
- @mingwandroid
- @stuarteberg

5.112 3.0.8 (2017-07-20)

5.112.1 Bug fixes

- Fix internal conda-build recipe to include missing setuptools and not use pip #2221
- Try to avoid downloading anything until we absolutely need it for rendering or build #2222
- Fix regexes that were leading to unsatisfiable dependencies, especially with perl #2222
- Tweak internal recipe to include more git info; adjust regex accordingly for this practice #2223

5.112.2 Contributors

@msarahan

5.113 3.0.7 (2017-07-20)

5.113.1 Enhancements

- Rewrite skeleton pypi template; match conda-forge standard #2205

5.113.2 Bug fixes

- Remove entry point links to avoid write-through to root envs #2209
- Properly insert variant versions for x.x in outputs (not just parent recipe) #2210
- Relax version constraints for lua and R in default variant #2213
- fix test of package directly using wrong config object #2214
- Don't check test env satisfiability when --no-test is passed #2218
- Iron out prefix when noarch as host env. Was creating separate build/host envs inappropriately. #2219
- Fix skipping finalization with finalize=False (for c3i speedup). #2219
- Fix implicit variant looping - wasn't keeping track of "used variables" that are used implicitly. #2219

5.113.3 Contributors

- @mandeep
- @mwcraig
- @msarahan

5.114 3.0.6 (2017-07-14)

5.114.1 Bug fixes

- Find git more intelligently, because build_prefix isn't always on PATH #2196
- Fix up assorted RPM skeleton issues #2196
- Fix and test "numpy x.x" recipes #2198
- Fix and test --skip-existing. Ensure that it also works with --croot. #2200
- Fix and test "python x.x" recipes #2201
- Fix inappropriate insertion of variant versions that led to conflicts (for example, numpy) #2202

5.114.2 Contributors

- @mingwandroid
- @msarahan

5.115 3.0.5 (2017-07-12)

5.115.1 Bug fixes

- Fix --skip-existing (was not matching output-dir/croot locations correctly) #2192
- Fix numpy x.x getting .* appended, resulting in unsatisfiable numpy #2193

5.115.2 Contributors

- @msarahan

5.116 3.0.4 (2017-07-11)

5.116.1 Bug fixes

- Don't symlink conda when building conda (clobbers actual scripts) #2167
- Fix pyldd following links #2170
- Preserve mode bit on noarch python bin/Scripts files #2171
- remove logging output showing up with --output option #2174
- Fix CONDA_* variables without . #2176
- pass croot to extraction (file path length issue on win) #2178
- fix uncorrect unpacking of tuples with --skip-existing #2179
- Fix priority of setup.cfg over setup.py #2180
- Remove overly aggressive removal of test prefix at end of test phase #2182
- Fix upper bound increment to account for pre-release versions (alpha, beta, rc, etc.) #2183

5.116.2 Contributors

- @jjhelmus
- @mingwandroid
- @msarahan

5.117 3.0.3 (2017-07-07)

5.117.1 Bug fixes

- fix loss of setup.cfg reading due to bad merge #2163
- avoid error when attempting to sort list, and that list consists of dicts #2163

5.117.2 Contributors

- @msarahan

5.118 3.0.2 (2017-07-06)

5.118.1 Enhancements

- Add SSL_CERT_FILE and REQUESTS_CA_BUNDLE env vars to passed-through variables #2142
- Sort several package aspects for package reproducibility #2143 #2149 #2154
- Add glob2 dependency to allow recursive globs in fields specifying filenames/paths #2146
- Add conda skeleton rpm for creating recipes to repackage RPMs as conda packages #2147
- Improve error messaging when git describe fails due to lack of annotated tags #2158

5.118.2 Bug fixes

- drop setup.py data that is not JSON serializable #2141
- enhance support for recipes containing unicode or non-ascii characters in meta.yaml #2148
- CRAN skeleton: allow some keys to be blank without throwing exceptions #2153
- Fix incorrect arguments passed to pyldd (use keywords) #2160
- fix incorrect distribution of variants when more than one variant key matched #2161

5.118.3 Contributors

- @aburgm
- @dougalsutherland
- @dhirschfeld
- @mandeep
- @MatthieuDartiailh
- @mingwandroid
- @msarahan
- @nehaljwani

5.119 2.1.17 (2017-06-30)

5.119.1 Bug fixes

- Fix disable_pip removing packages even when they are explicit dependencies #2129
- Remove any pyc files for entry point scripts that pip may have created #2134
- Ignore unserializable data when reading setup.py data #2141

5.119.2 Contributors

- @mandeep
- @msarahan

5.120 3.0.1 (2017-06-29)

This release includes all changes to the 2.1.x branch up to and including the 2.1.16 release.

5.120.1 Enhancements

- Raise errors prior to build if any run or test deps are unsatisfiable #2102
- Add skeleton function for RPM packages, to be used for things like Xorg #2109
- Improve test coverage of workdir removal #2111 #2112
- Match variants in conda_build_config.yaml with dep names (implicit jinja2 version spec) #2124

5.120.2 Bug fixes

- fix reference to cc.subdir (it is just subdir) #2015
- fix failing test when using filename_hashing=False (non-existent json file) #2087
- fix dependencies specified to conda-convert not being added #2090
- fix disable_pip removing packages even when they are explicit recipe deps #2129
- fix pin_compatible jinja2 function not respecting lower_bound as None correctly #2138

5.120.3 Contributors

- @jakirkham
- @mandeep
- @mingwandroid
- @msarahan
- @neok-m4700

5.121 2.1.16 (2017-06-23)

5.121.1 Enhancements

- add CLI flag and condarc setting to disable --force for anaconda upload #2047
- add doc_source_url to allowed fields in about section #2048
- add a second pass for getting information from setup.py that is performed in the build environment, so that version-specific logic in setup.py should work. #2071
- add semicolons to print statements in test files to avoid errors with Perl. #2012 #2089
- pass through more CPU-specific environment variables on windows #2072
- pass through DISPLAY environment variable on Linux #2098
- Improve conda_interface for better conda 4.4 and later 4.3.x releases #2113
- skeleton pypi & lua: replace legacy noarch syntax with conda 4.3 style #2120
- Restore --keep-old-work flag: works like --dirty to leave your build intermediaries, but does not reuse build folders like --dirty. #2119
- Speed up and fix-up conda-convert #2116 #2123

5.121.2 Bug fixes

- fix test/imports with multiple language entries #1967
- add missing six dependency in conda recipe for conda-build #2063
- fix dependency addition when converting packages #2091
- don't set build_id in test phase when --no-build-id is given #2100
- fix handling of string literals not being lists in test requirements from setup.py #2107

5.121.3 Contributors

- @aburgm
- @AndresGuzman-Ballen
- @gqmelo
- @isuruf
- @kalefranz
- @mandeep
- @mingwandroid
- @msarahan
- @nehaljwani
- @nsoranzo
- @timsnyder
- @vinjana

5.122 3.0.0 (2017-05-23)

These release notes are an aggregation of all older pre-releases of conda-build 3.0.0. All changes from 2.1.15 and below have been incorporated.

5.122.1 Breaking changes

- Support for post-build metadata (__conda_version__.txt and the like) has been removed.
- pin_downstream has been renamed to run_exports #1911
- exclude_from_build_hash has been renamed to ignore_version #1911
- Package signing and verification have been removed, to follow their removal from conda 4.3. #1950

5.122.2 Enhancements

- greatly extended Jinja2 templating capabilities #1585
- record environment variables at top of build.sh, similar to what is done with bld.bat in win. #1765
- use symlinks when copying to improve performance #1867
- load setup.cfg data in load_setup_py_data #1878
- calculate checksum and simplify cran skeleton #1879
- Check that files are executable when finding them #1899
- use rm_rf to remove prefixes for more cleanliness and better speed #1915
- add support for multiple sources in one meta.yaml #1929
- allow `exact` keyword for pin_compatible jinja2 function #1929
- allow selectors in variant conda_build_config.yaml files #1937
- Avoid duplicate recreation of package index. Speedup of 0-50%, depending on how extensively the recipe uses Jinja2 features. #1954
- Allow per-subpackage specification of target subdir #1961
- Add basic environment marker support to conda skeleton pypi #1984
- allow about section for each subpackage #1987
- add support for optional dependencies (conda 4.4) #2001
- fix windows entry point exe's for unicode #2045
- allow strings for pin_run_as_build values (e.g. x.x) rather than just dictionaries #2042
- add meta.yaml entry to override run_exports pins #2073
- add several condarc entries that can be used to control conda-build behavior #2074
- add new pyldd tool and use it when ldd/otool fail #2082
- Allow configuration of conda-build's loggers by logging configuration files. Default to debug,info going to stdout, warn,error going to stderr. #2078
- rename work dir before tests, rather than removing it, so that build intermediates can be inspected if tests fail. #2078

5.122.3 Bug fixes

- fix symlinks to folders in packaging #1775
- fix detection of patch level when maxlevel=0 #1796
- properly copy permissions when extracting zip files #1855
- Add more important Windows environment variables to the test environment #1859, #1863
- remove build and test envs after each packaging step, to avoid unsatisfiable errors #1866
- remove version pins from requirements added by run_test files (again avoid unsatisfiable errors) #1866
- fix prefix file detection picking up too many files due to env recreation #1866
- fix missing r_bin, make run_test.r scripts work #1869
- fix R's binary path on Windows #1870
- remove tab completion on CLI for compatibility with conda 4.4 #1795
- reduce scope of git try/except block so that GIT_FULL_HASH is available, even if tags are not #1873
- Fix "compiler" jinja2 looping, so that it is respected in subpackages #1874
- Fix license family comparison - case matching #1875
- Fix inspect linkages when multiple packages contain a library #1884
- avoid unnecessary computation of hashing #1888
- fix python imports not being run in test phase #1896
- fix path omission in paths.json for noarch packages #1895
- standardize entry point script template to match pip #1908
- fix cleanup happening even when build fails #1909
- fix bin/conda getting included in conda-build release tarballs #1913
- fix mmap/file problems on virtualbox shared folders #1914
- Correct rendering with --dirty flag if recipe name appears as substring of another's name #1931
- don't set language env vars (PERL, R, LUA, PYTHON) when those packages are not installed #1932
- exclude language env vars from variant being set #1944
- Fix cleanup of folders in outer variant loop - was causing incorrect packages on 2nd variant in windows builds #1950
- Remove variant functionality from bdist_conda. Its split packaging approach is incompatible. #1950
- Fix import of _toposort from conda, reroute through conda_interface #1952
- Match folder substrings more intelligently, for finding previous builds with --dirty #1953
- Fix type error with --skip-existing and some conda recipes (Conda-build's internal conda.recipe was one). #1956
- Fix non-python packages creating python tests where they should not have #1967
- Don't add python.app to run reqs multiple times #1972
- Fix incorrect removal of cc in conda_interface.py #1971
- Fix ORIGIN replacement - trailing slash was messing things up #1982
- Pipe stdin when calling subprocess, in hopes of getting better ctrl-c handling with conda. #1986

- Ensure that lock files are removed after build exit (or crash) to avoid permission errors on central installs #2002
- Process line endings in bytes mode rather than text mode #2035
- add a warning to find_recipe when multiple meta.yaml files are found (bioconda style) #2040
- When applying patches, try 3 line ending options on the patch: 1. unchanged; 2. convert patch to unix line endings; 3. convert patch to windows line endings. #2052
- fix empty target_platform variant entry leading to empty builds #2056
- fix host activation for cross-capable recipes #2060
- fix handling of circularity in subpackages #2065
- fix subdir handling for subdirs with more than one - character #2066
- Install build and host deps when using cross-capable recipe on strictly native (not cross) build #2070
- reduce verbosity of git error messages that people never care about #2075
- hash only direct inputs of subpackages, rather than all files. This limits creation of identical packages with similar hashes. #2079

5.122.4 Contributors

- @abretaud
- @evhub
- @groutr
- @jjhelmus
- @kalefranz
- @ma-ba
- @mandeep
- @mingwandroid
- @minrk
- @msarahan
- @pkgw
- @pwwang
- @rolando
- @stuarteberg
- @tatome
- @ukoethe
- @waltonseymour
- @wulmer

5.123 3.0.0rc1 (2017-05-23)

These release notes are an aggregation of all older pre-releases of conda-build 3.0.0, plus changes since 3.0.0rc0. All changes from 2.1.15 and below have been incorporated.

5.123.1 Breaking changes

- Support for post-build metadata (__conda_version__.txt and the like) has been removed.
- pin_downstream has been renamed to run_exports #1911
- exclude_from_build_hash has been renamed to ignore_version #1911
- Package signing and verification have been removed, to follow their removal from conda 4.3. #1950

5.123.2 Enhancements

- greatly extended Jinja2 templating capabilities #1585
- record environment variables at top of build.sh, similar to what is done with bld.bat in win. #1765
- use symlinks when copying to improve performance #1867
- load setup.cfg data in load_setup_py_data #1878
- calculate checksum and simplify cran skeleton #1879
- Check that files are executable when finding them #1899
- use rm_rf to remove prefixes for more cleanliness and better speed #1915
- add support for multiple sources in one meta.yaml #1929
- allow `exact` keyword for pin_compatible jinja2 function #1929
- allow selectors in variant conda_build_config.yaml files #1937
- Avoid duplicate recreation of package index. Speedup of 0-50%, depending on how extensively the recipe uses Jinja2 features. #1954
- Allow per-subpackage specification of target subdir #1961
- Add basic environment marker support to conda skeleton pypi #1984
- allow about section for each subpackage #1987
- add support for optional dependencies (conda 4.4) #2001
- fix windows entry point exe's for unicode #2045
- allow strings for pin_run_as_build values (e.g. x.x) rather than just dictionaries #2042

5.123.3 Bug fixes

- fix symlinks to folders in packaging #1775
- fix detection of patch level when maxlevel=0 #1796
- properly copy permissions when extracting zip files #1855
- Add more important Windows environment variables to the test environment #1859, #1863
- remove build and test envs after each packaging step, to avoid unsatisfiable errors #1866
- remove version pins from requirements added by run_test files (again avoid unsatisfiable errors) #1866
- fix prefix file detection picking up too many files due to env recreation #1866
- fix missing r_bin, make run_test.r scripts work #1869
- fix R's binary path on Windows #1870
- remove tab completion on CLI for compatibility with conda 4.4 #1795
- reduce scope of git try/except block so that GIT_FULL_HASH is available, even if tags are not #1873
- Fix "compiler" jinja2 looping, so that it is respected in subpackages #1874
- Fix license family comparison - case matching #1875
- Fix inspect linkages when multiple packages contain a library #1884
- avoid unnecessary computation of hashing #1888
- fix python imports not being run in test phase #1896
- fix path omission in paths.json for noarch packages #1895
- standardize entry point script template to match pip #1908
- fix cleanup happening even when build fails #1909
- fix bin/conda getting included in conda-build release tarballs #1913
- fix mmap/file problems on virtualbox shared folders #1914
- Correct rendering with --dirty flag if recipe name appears as substring of another's name #1931
- don't set language env vars (PERL, R, LUA, PYTHON) when those packages are not installed #1932
- exclude language env vars from variant being set #1944
- Fix cleanup of folders in outer variant loop - was causing incorrect packages on 2nd variant in windows builds #1950
- Remove variant functionality from bdist_conda. Its split packaging approach is incompatible. #1950
- Fix import of _toposort from conda, reroute through conda_interface #1952
- Match folder substrings more intelligently, for finding previous builds with --dirty #1953
- Fix type error with --skip-existing and some conda recipes (Conda-build's internal conda.recipe was one). #1956
- Fix non-python packages creating python tests where they should not have #1967
- Don't add python.app to run reqs multiple times #1972
- Fix incorrect removal of cc in conda_interface.py #1971
- Fix ORIGIN replacement - trailing slash was messing things up #1982
- Pipe stdin when calling subprocess, in hopes of getting better ctrl-c handling with conda. #1986

- Ensure that lock files are removed after build exit (or crash) to avoid permission errors on central installs #2002
- Process line endings in bytes mode rather than text mode #2035
- add a warning to find_recipe when multiple meta.yaml files are found (bioconda style) #2040
- When applying patches, try 3 line ending options on the patch: 1. unchanged; 2. convert patch to unix line endings; 3. convert patch to windows line endings. #2052
- fix empty target_platform variant entry leading to empty builds #2056

5.123.4 Contributors

- @abretaud
- @evhub
- @groutr
- @jjhelmus
- @kalefranz
- @ma-ba
- @mandeep
- @mingwandroid
- @minrk
- @msarahan
- @pkgw
- @pwwang
- @rolando
- @stuarteberg
- @tatome
- @ukoethe
- @wulmer

5.124 3.0.0rc0 (2017-05-16)

These release notes are an aggregation of all older pre-releases of conda-build 3.0.0, plus changes since 3.0.0beta1. All changes from 2.1.13 and below have been incorporated.

5.124.1 Breaking changes

- Support for post-build metadata (__conda_version__.txt and the like) has been removed.
- pin_downstream has been renamed to run_exports #1911
- exclude_from_build_hash has been renamed to ignore_version #1911
- Package signing and verification have been removed, to follow their removal from conda 4.3. #1950

5.124.2 Enhancements

- greatly extended Jinja2 templating capabilities #1585
- record environment variables at top of build.sh, similar to what is done with bld.bat in win. #1765
- use symlinks when copying to improve performance #1867
- load setup.cfg data in load_setup_py_data #1878
- calculate checksum and simplify cran skeleton #1879
- Check that files are executable when finding them #1899
- use rm_rf to remove prefixes for more cleanliness and better speed #1915
- add support for multiple sources in one meta.yaml #1929
- allow `exact` keyword for pin_compatible jinja2 function #1929
- allow selectors in variant conda_build_config.yaml files #1937
- Avoid duplicate recreation of package index. Speedup of 0-50%, depending on how extensively the recipe uses Jinja2 features. #1954
- Allow per-subpackage specification of target subdir #1961
- Add basic environment marker support to conda skeleton pypi #1984
- allow about section for each subpackage #1987
- add support for optional dependencies (conda 4.4) #2001

5.124.3 Bug fixes

- fix symlinks to folders in packaging #1775
- fix detection of patch level when maxlevel=0 #1796
- properly copy permissions when extracting zip files #1855
- Add more important Windows environment variables to the test environment #1859, #1863
- remove build and test envs after each packaging step, to avoid unsatisfiable errors #1866
- remove version pins from requirements added by run_test files (again avoid unsatisfiable errors) #1866
- fix prefix file detection picking up too many files due to env recreation #1866
- fix missing r_bin, make run_test.r scripts work #1869
- fix R's binary path on Windows #1870
- remove tab completion on CLI for compatibility with conda 4.4 #1795
- reduce scope of git try/except block so that GIT_FULL_HASH is available, even if tags are not #1873

- Fix "compiler" jinja2 looping, so that it is respected in subpackages #1874
- Fix license family comparison - case matching #1875
- Fix inspect linkages when multiple packages contain a library #1884
- avoid unnecessary computation of hashing #1888
- fix python imports not being run in test phase #1896
- fix path omission in paths.json for noarch packages #1895
- standardize entry point script template to match pip #1908
- fix cleanup happening even when build fails #1909
- fix bin/conda getting included in conda-build release tarballs #1913
- fix mmap/file problems on virtualbox shared folders #1914
- Correct rendering with --dirty flag if recipe name appears as substring of another's name #1931
- don't set language env vars (PERL, R, LUA, PYTHON) when those packages are not installed #1932
- exclude language env vars from variant being set #1944
- Fix cleanup of folders in outer variant loop - was causing incorrect packages on 2nd variant in windows builds #1950
- Remove variant functionality from bdist_conda. Its split packaging approach is incompatible. #1950
- Fix import of _toposort from conda, reroute through conda_interface #1952
- Match folder substrings more intelligently, for finding previous builds with --dirty #1953
- Fix type error with --skip-existing and some conda recipes (Conda-build's internal conda.recipe was one). #1956
- Fix non-python packages creating python tests where they should not have #1967
- Don't add python.app to run reqs multiple times #1972
- Fix incorrect removal of cc in conda_interface.py #1971
- Fix ORIGIN replacement - trailing slash was messing things up #1982
- Pipe stdin when calling subprocess, in hopes of getting better ctrl-c handling with conda. #1986
- Ensure that lock files are removed after build exit (or crash) to avoid permission errors on central installs #2002

5.124.4 Contributors

- @abretaud
- @evhub
- @groutr
- @jjhelmus
- @kalefranz
- @ma-ba
- @mingwandroid
- @msarahan
- @pkgw

- @pwwang
- @stuarteberg
- @tatome
- @ukoethe
- @wulmer

5.125 2.1.13 (2017-05-10)

5.125.1 Bug fixes

- fix missing argument to get_site_packages function; add test coverage #2009
- pin codecov on appveyor to 2.0.5 for now #2009
- fix lock removal (just don't create locks for temporary directories) #2009

5.125.2 Contributors

- @msarahan

5.126 2.1.12 (2017-05-09)

5.126.1 Bug fixes

- Clean up lock files for temporary directories also

5.126.2 Contributors

- @msarahan

5.127 2.1.11 (2017-05-09)

5.127.1 Enhancements

- add libgcc to build dependencies for R skeleton recipes that require compilation $1969

5.127.2 Bug fixes

- fix entry points, test commands, test imports from top-level recipe from applying to subpackages #1933
- fix preferred_env in index.json #1941
- do not add python.app to run_reqs multiple times #1981
- Fix $ORIGIN replacement from extra trailing slash #1981
- Remove error when _license package exists in folder where `conda index` is called #2005
- fix STDLIB_DIR so that it is always defined (based on python version in configuration) #2006
- Clean up lock files after builds complete or fail to avoid permission errors #2007

5.127.3 Contributors

- @johanneskoester
- @kalefranz
- @mingwandroid
- @msarahan

5.128 3.0.0beta1 (2017-04-25)

5.128.1 Breaking changes

- Package signing and verification have been removed, to follow their removal from conda 4.3. #1950

5.128.2 Enhancements

- Avoid duplicate recreation of package index. Speedup of 0-50%, depending on how extensively the recipe uses Jinja2 features. #1954

5.128.3 Bug fixes

- Fix cleanup of folders in outer variant loop - was causing incorrect packages on 2nd variant in windows builds #1950
- Remove variant functionality from bdist_conda. Its split packaging approach is incompatible. #1950
- Fix import of _toposort from conda, reroute through conda_interface #1952
- Match folder substrings more intelligently, for finding previous builds with --dirty #1953
- Fix type error with --skip-existing and some conda recipes (Conda-build's internal conda.recipe was one). #1956

5.128.4 Contributors

- @kalefranz
- @msarahan
- @rendinam

5.129 3.0.0beta0 (2017-04-20)

5.129.1 Breaking changes

- pin_downstream has been renamed to run_exports #1911
- exclude_from_build_hash has been renamed to ignore_version #1911

5.129.2 Enhancements

- use rm_rf to remove prefixes for more cleanliness and better speed #1915
- add support for multiple sources in one meta.yaml #1929
- allow `exact` keyword for pin_compatible jinja2 function #1929
- allow selectors in variant conda_build_config.yaml files #1937

5.129.3 Bug fixes

- fix mmap/file problems on virtualbox shared folders #1914
- Correct rendering with --dirty flag if recipe name appears as substring of another's name #1931
- don't set language env vars (PERL, R, LUA, PYTHON) when those packages are not installed #1932
- exclude language env vars from variant being set #1944

5.129.4 Contributors

- @mingwandroid
- @msarahan
- @rendinam

5.130 2.1.10 (2017-04-17)

5.130.1 Enhancements

- Inspect linkages will now warn when multiple packages contain the same library #1884, #1921

5.130.2 Bug fixes

- Fix bin/conda getting included in packages that also had entry point scripts or binaries starting with 'conda' #1923
- Fix empty create_env, for compatibility with conda 4.4 #1924
- Adapt to Conda's new MatchSpec implementation #1927
- Fix unbound local variables when --no-locking option used. #1928
- Don't set language env vars (PERL, R, LUA, etc.) when packages for those languages are not installed #1930

5.130.3 Contributors

- @jjhelmus
- @kalefranz
- @msarahan

5.131 3.0.0alpha2 (2017-04-05)

5.131.1 Breaking changes

- Support for post-build metadata (__conda_version__.txt and the like) has been removed.

5.131.2 Enhancements

- use symlinks when copying to improve performance #1867
- load setup.cfg data in load_setup_py_data #1878
- calculate checksum and simplify cran skeleton #1879

5.131.3 Bug fixes

- fix R's binary path on Windows #1870
- remove tab completion on CLI for compatibility with conda 4.4 #1795
- reduce scope of git try/except block so that GIT_FULL_HASH is available, even if tags are not #1873
- Fix "compiler" jinja2 looping, so that it is respected in subpackages #1874
- Fix license family comparison - case matching #1875
- Fix inspect linkages when multiple packages contain a library #1884
- avoid unnecessary computation of hashing #1888
- fix python imports not being run in test phase #1896
- fix path omission in paths.json for noarch packages #1895

5.131.4 Contributors

- @abretaud
- @groutr
- @jjhelmus
- @kalefranz
- @ma-ba
- @mingwandroid
- @msarahan

5.132 2.1.9 (2017-04-04)

5.132.1 Enhancements

- calculate checksum and simplify cran skeleton #1879
- backport usage of symlinks for speed from master branch #1881

5.132.2 Bug fixes

- fix import tests not being run, test this functionality #1897

5.132.3 Contributors

- @isuruf
- @jjhelmus
- @johanneskoester
- @msarahan

5.133 2.1.8 (2017-03-24)

5.133.1 Enhancements

- use symlinks when copying files from files sources to improve performance #1867

5.133.2 Bug fixes

- reset build folder for each built package (fixes building multiple recipes in one command) #1842
- wrap copy of test/source_files so that errors don't prevent a successful build #1843
- Restore permissions when extracting from zipfiles #1855
- pass through several Windows-specific environment variables #1859, #1862
- python 2 os.environ string type compatibility fix #1861
- fix indentation breaking perl package testing #1872
- reduce scope of git try/except block so that GIT_FULL_HASH is available even if tags are not. #1873
- fix license family comparison, especially for public-domain #1875
- Remove python header being added to all run_test.* files #1876

5.133.3 Contributors

- @abretaud
- @jjhelmus
- @mingwandroid
- @msarahan
- @stuertz
- @wulmer

5.134 3.0.0alpha1 (2017-03-23)

5.134.1 Bug fixes

- remove build and test envs after each packaging step, to avoid unsatisfiable errors #1866
- remove version pins from requirements added by run_test files (again avoid unsatisfiable errors) #1866
- fix prefix file detection picking up too many files due to env recreation #1866
- fix missing r_bin, make run_test.r scripts work #1869

5.134.2 Contributors

- @msarahan

5.135 3.0.0alpha0 (2017-03-22)

This is a complete revolution in the dynamic rendering capabilities of conda-build. More information is in the docs PR at https://github.com/conda/conda-docs/pull/414. There will be a blog post soon, perhaps coupled with a screencast.

5.135.1 Enhancements

- greatly extended Jinja2 templating capabilities #1585
- record environment variables at top of build.sh, similar to what is done with bld.bat in win. #1765

5.135.2 Bug fixes

- fix symlinks to folders in packaging #1775
- fix detection of patch level when maxlevel=0 #1796
- properly copy permissions when extracting zip files #1855
- Add more important Windows environment variables to the test environment #1859, #1863

5.135.3 Contributors

- @jjhelmus
- @kalefranz
- @mingwandroid
- @msarahan
- @pkgw
- @stuarteberg
- @ukoethe
- @wulmer

5.136 2.1.7 (2017-03-14)

5.136.1 Enhancements

- pass WINDIR env var through on Windows #1837
- make long test prefix an option (default disabled) #1838

5.136.2 Bug fixes

- monkeypatch ensure_use_local to avoid conda-build import clobbering conda CLI arguments #1834
- Fix context conda_build attr error with older conda #1813
- Fix legacy noarch shebang replacement code to account for long prefix paths #1813
- properly initialize 'system' key in linkage inspecting #1839
- backport try mmap from master #1764
- fix wheel output not respecting --output-folder CLI argument #1838
- catch csv dialect sniffing error, try to fallback to to excel_tab. Might work? #1840

5.136.3 Contributors

- @kalefranz
- @mcs07
- @msarahan

5.137 2.1.6 (2017-03-08)

5.137.1 Enhancements

- tests on linux/mac now use 255-character prefix when creating test environment #1799
- allow test/imports for R and lua packages #1806

5.137.2 Bug fixes

- Fix case comparison in license_family.py #1761
- Fix symlinked folders not being included in packages #1770
- Fix extraction of tarballs containing unicode filenames #1779, #1804
- fix unicode in delimiter for noarch py_file_map #1789
- Clean up conda interface #1791
- Confine conda-build 2.1.x to conda >4.1,<=4.3 #1792
- fix detection of patch strip level when maxlevel = 0 #1796
- fix attribute error in exception handling for missing dependencies #1800
- fix osx python_app test for python 3.6 #1801
- don't die when unicode found in patch files #1802
- clarify error messaging when git is not found #1803
- fix shebangs in entry point scripts using legacy noarch_python #1806
- fix test environment variables being set to build prefix values #1806
- fix inspect linkages breaking due to conda index keys changing to different objects in conda 4.3 #1810

5.137.3 Contributors

- @gbrener
- @kalefranz
- @msarahan
- @pkgw
- @stuertz

5.138 2.1.5 (2017-02-20)

5.138.1 Enhancements

- don't crash on unknown selector. Warn, but evaluate as False. #1753
- allow default conda packaging behavior for split package whose name matches top-level name, but lacks both `files` and `script` entry. #1758

5.138.2 Bug fixes

- unify license family text with Anaconda-Verify #1744
- apply post-processing to each split package, not just to post-build prefix files. #1747
- provide fallback lock directory in user's home folder. Allows central installs. #1749
- fix quoting for test paths. Allows croot with spaces. #1750
- fix pypi skeleton recursion #1754
- fix assertion error about leading period when Jinja2 variables have default values #1757
- set default twine target to pypitest to avoid accidental uploads #1758

5.138.3 Contributors

- @gabm
- @msarahan

5.139 2.1.4 (2017-02-07)

5.139.1 Enhancements

- Allow relative paths for --croot option #1736

5.139.2 Bug fixes

- Rename package_metadata.json file to link.json to more accurately reflect contents #1720
- Fix converted packages from unix to Windows having broken entry points #1721
- Fix an infinite loop when creating the test environment failed #1739
- Fix conda 4.3 incompatibility with --pin-depends option #1740

5.139.3 Contributors

- @gabm
- @kalefranze
- @msarahan

5.140 2.1.3 (2017-01-31)

5.140.1 Enhancements

- Add --extra-specs to conda skeleton. Use when a package needs to be available in the temporary env that parses setup.py to make the skeleton. #1697
- Allow wheels as a source format #1700
- Allow github urls as CRAN skeleton sources #1710

5.140.2 Bug fixes

- exclude package/name field from uses_vcs_in_{meta,build} checks #1696
- Fix conda convert wrt info/paths.json (for conda 4.3 compatibility) #1701
- update cpan skeleton to use newer API url, fix conda exception handling #1704
- update R default version to 3.3.2 #1707
- fix attribute error on exception handling (better fix on the way) #1709
- fix bundle_conda removing project files when conda recipe was in the source tree, and utilized relative paths #1715
- fix glob trying to interpret filenames that look like glob patterns #1717

5.140.3 Contributors

- @ElliotJH
- @jerowe
- @kalefranz
- @mingwandroid
- @minrk

- @msarahan
- @rainwoodman

5.141 2.1.2 (2017-01-20)

5.141.1 Enhancements

- iron out compatibility with conda 4.3 #1667
- pytest improvements for a cleaner CI experience #1686 #1687

5.141.2 Bug fixes

- Avoid trailing semicolon in MSYS2_ARG_CONV_EXCL variable setting #1651
- filter .git directories more strictly (keep x.git folders, not .git) #1657
- fix 404's killing CPAN skeleton with newer conda versions #1667
- use pythonw to run tests on OSX when osx_is_app is defined in meta.yaml #1669
- ignore obnoxious .DS_Store files when packaging #1670
- fix --source argument to download source specified in meta.yaml #1671
- fix slashes in file prefix replacement on Windows #1680
- fix multiple source url fallbacks (handle CondaHTTPErrors) #1683
- fix bizarre encoding errors on Windows with projects that embed binary in setup files #1685
- fix CPAN JSON file encoding issue on windows #1688
- revise retry when conda is missing files from a package #1690

5.141.3 Contributors

- @dfroger
- @kalefranz
- @mingwandroid
- @msarahan
- @nicoddemus
- @pkgw

5.142 2.1.1 (2017-01-12)

5.142.1 Bug fixes

- Fix package conversion consistency, wrt entry points #1609
- Fix about.json contents regarding development versions of conda/conda-build #1625
- Fix Appveyor for testing against source branches of conda #1628
- Raise exception when SRC_DIR is used in tests, but meta.yaml has no test/source_files entry. SRC_DIR points at test work folder at test time, for minimal needed changes to recipes - just add test/source_files entry with desired files. #1630
- Fix features list not allowed to be None in bdist_conda #1636
- Fix undefined reference to locks in copy_into #1637
- Fix version comparison in cpan skeleton #1638
- Add dependency on beautifulsoup4 and chardet to better support strangely encoded text files. #1641
- Fix not-yet-fully-rendered versions starting with . from raising exception #1644
- Consolidate _check_call and check_call_env in utils. Fix coercion to string that was missing in latter. #1645

5.142.2 Contributors

- @gomyhr
- @jakirkham
- @kalefranz
- @msarahan

5.143 2.1.0 (2017-01-02)

(includes changes since 2.0.12, including those already listed under 2.1.0beta1)

5.143.1 Enhancements

- Consolidate package metadata from extra.json and noarch.json into package_metadata.json #1535 #1539
- finalize paths.json, (formerly files.json), which supersedes the older separate files for tracking file data #1535
- Support output of multiple packages from one recipe #1576
- Support output of wheels #1576
- Add `--prefix-length` to conda-build. This allows one to set the prefix length manually. It should be used sparingly, preferring creation of a temporary folder on non-encrypted locations, and setting `--croot` to that temporary folder. #1579
- Add `--no-prefix-length-fallback` option to conda-build, to fail builds that encounter short prefixes, rather than falling back to the short prefix #1579
- Change CRAN-skeleton to follow conda-forge style #1596

- Allow relative paths to files in source/url field #1614

5.143.2 Bug fixes

- Rework locks to be more robust #1540
- Call make_hardlink_copy on Windows to prevent conda failures #1575
- Delete the work folder before running the test suite. Exposes faulty links to source files more easily. #1581
- Add support for Python 3.6 in selector expressions #1592
- Don't try to compile pyc files when python is not installed in the build env #1593
- Fix a function call leading to corrupted meta.yaml output #1589
- Fix logger to be package-local. Fixes logger output not showing up. #1583
- Disallow leading periods in package version #1594
- Fix reference to undefined need_source_download #1595
- Disallow - in feature names, to avoid conflicts with conda's handling of package names #1600
- Remove help text about passing multiple --python options or "all" - this has been broken for some time. Replacement coming in 3.0.0. #1610
- Fix clobbering of no_link variable. #1611
- Fix index when --output-folder is specified #1613
- Fix python_d.exe incompatibility with conda 4.3 #1615

5.143.3 Contributors

- @asmeurer
- @hajs
- @johanneskoester
- @kalefranz
- @mingwandroid
- @msarahan
- @mwiebe
- @soapy1

5.144 2.1.0beta1 (2016-12-20)

5.144.1 Enhancements

- Consolidate package metadata from extra.json and noarch.json into package_metadata.json #1535 #1539
- finalize paths.json, (formerly files.json), which supersedes the older separate files for tracking file data #1535
- Support output of multiple packages from one recipe #1576
- Support output of wheels #1576

- Add `--prefix-length` to conda-build. This allows one to set the prefix length manually. It should be used sparingly, preferring creation of a temporary folder on non-encrypted locations, and setting `--croot` to that temporary folder. #1579
- Add `--no-prefix-length-fallback` option to conda-build, to fail builds that encounter short prefixes, rather than falling back to the short prefix #1579
- Change CRAN-skeleton to follow conda-forge style #1596

5.144.2 Bug fixes

- Rework locks to be more robust #1540
- Call make_hardlink_copy on Windows to prevent conda failures #1575
- Delete the work folder before running the test suite. Exposes faulty links to source files more easily. #1581
- Add support for Python 3.6 in selector expressions #1592
- Don't try to compile pyc files when python is not installed in the build env #1593
- Fix a function call leading to corrupted meta.yaml output #1589
- Fix logger to be package-local. Fixes logger output not showing up. #1583
- Disallow leading periods in package version #1594
- Fix reference to undefined need_source_download #1595
- Disallow - in feature names, to avoid conflicts with conda's handling of package names #1600

5.144.3 Contributors

- @asmeurer
- @hajs
- @johanneskoester
- @kalefranz
- @mingwandroid
- @msarahan
- @mwiebe
- @soapy1

5.145 2.0.12 (2016-12-12)

5.145.1 Enhancements

- Whitelist, rather than hardcode, MACOSX_DEPLOYMENT_TARGET. Default to 10.7 #1561
- Allow local relative paths to be passed as channel arguments #1565

5.145.2 Bug fixes

- Keep CONDA_PATH_BACKUP as allowed variable in build/test env activation. Necessary to make deactivation work correctly. #1560
- Define nomkl selector when FEATURE_NOMKL environment variable is not set #1562
- Move test removal of packaged recipe until after test completes #1563
- Allow source_files in recognized meta.yaml fields #1572

5.145.3 Contributors

- @jakirkham
- @mingwandroid
- @msarahan

5.146 2.0.11 (2016-11-28)

5.146.1 Enhancements

- Further develop and update files.json #1501
- New command line option: `--output-folder` allows moving artifact after build (to facilitate CI) #1538
- support globs in *ignore_prefix_files*, *has_prefix_files*, *always_include_files*, *binary_has_prefix_files* #1554
- decouple *ignore_prefix_files* from *binary_relocation*; make *binary_relocation* also accept a list of files or globs #1555

5.146.2 Bug fixes

- rename *short_path* key in files.json to *path* #1501
- allow *!* in package version (used in epoch) #1542
- don't compute SHA256 for folders #1544
- fix merge check for dst starting with src #1546
- use normpath when comparing utils.relative (fixes git clone issue) #1547
- disallow softlinks for conda (>=v.4.2) in conda-build created environments #1548

5.146.3 Contributors

- @mingwandroid
- @msarahan
- @soapy1

5.147 2.0.10 (2016-11-14)

5.147.1 Bug fixes

- Fix backwards incompatibility with conda 4.1 #1528

5.147.2 Contributors

- @msarahan

5.148 2.0.9 (2016-11-11)

5.148.1 Enhancements

- break build string construction out into standalone function for external usage (Concourse CI project) #1513
- add conda-verify support. Defaults to enabled. Adds conda-verify as runtime requirement.
-

5.148.2 Bug fixes

- handle creation of intermediate folders when filenames provided as build/source_files arguments #1511
- Fix passing of version argument to pypi skeleton arguments #1516
- break symlinks and copy files if symlinks point to executable outside of same path (fix RPATH misbehavior on linux/mac, because ld.so follows symlinks) #1521
- specify conda executable name more specifically when getting about.json info. It was not being found in some cases without the file extension. #1525

5.148.3 Contributors

- @jhprinz
- @msarahan
- @soapy1

5.149 2.0.8 (2016-11-03)

5.149.1 Enhancements

- Support otool -h changes in MacOS 10.12 #1479
- Fix lists of empty strings created by ensure_list (patches failing due to empty patch list) #1493
- Improved logic to guess the appropriate license_family to add to package's index. This improves filtering. #1495 #1503

- Logic for the license_family is now shared between open-source conda-build, and proprietary cas-mirror packages. #1495 #1503

5.149.2 Bug fixes

- Centralize locks in memory to avoid lock timeouts within a single process #1496
- fix overly broad regex in detecting whether a recipe uses version control systems #1498
- clarify error message when extracting egg fails due to overlapping file names #1500
- fix regression where subdir was not respecting non-x86 arch (values other than 32 or 64) #1506

5.149.3 Contributors

- @caseyclements
- @minrk
- @msarahan

5.150 2.0.7 (2016-10-24)

5.150.1 Enhancements

- populate SHLIB_EXT environment variable. For example, .so, .dll, .dylib file extensions use this for their native ending. #1478

5.150.2 Bug fixes

- fix metapackage not going through api, thus not showing output file path. #1470
- restore script exe creation on Windows. These are for standalone scripts installed by distutils or setuptools in setup.py. #1471
- fix noarch value propagation from meta.yaml to config. Was causing noarch to not be respected in some cases. #1472
- fix calls to subprocess not converting unicode to str #1473
- restore detect_binary_files_with_prefix behavior - in particular, respect it when false. # 1477

5.150.3 Contributors

- @jschueller
- @mingwandroid
- @msarahan

5.151 2.0.6 (2016-10-13)

5.151.1 Bug fixes

- fix erroneous import that was only compatible with conda 4.2.x #1460

5.151.2 Contributors

- @msarahan

5.152 2.0.5 (2016-10-13)

5.152.1 Enhancements

- Add new jinja function for extracting information from files with regular expressions #1443

5.152.2 Bug fixes

- Quote paths in activation of build and test envs #1448
- Fix source re-copy (leading to IOError) with test as a separate step #1452
- Call conda with an absolute path when gathering metadata for package about.json #1455
- Don't strictly require conda-env to be present for about.json data #1455
- Fix version argument to skeletons not being respected #1456
- Fix infinite recursion when missing dependency is either r or python #1458

5.152.3 Contributors

- @bryanwweber
- @msarahan

5.153 2.0.4 (2016-10-07)

5.153.1 Enhancements

- Add build/skip_compile_pyc meta.yaml option. Use to skip compilation on pyc files listed therein. #1169
- Add build environment metadata to about.json (conda, conda-build versions, channels, root pkgs) #1407
- Make subdir member of config a derived property, so that setting platform or bits is more direct #1427
- Use subprocess call to windows del function to clear .trash folder, rather than conda. Big speedup. #1438

5.153.2 Bug fixes

- fix regression regarding 'config' in pypi skeleton for recipes with entry points #1430
- don't load setup.py data when considering entry points (use only info from meta.yaml) #1431
- fall back to trying to copy files without attributes or metadata if those fail #1436
- Fix permissions on packaged files to be user and group writable, and other readable. #1437
- fix conda develop not respecting python version of target environment #1440

5.153.3 Contributors

- @mingwandroid
- @msarahan

5.154 2.0.3 (2016-09-27)

5.154.1 Enhancements

- add support for noarch: python #1366

5.154.2 Bug fixes

- convert popen args to bytestrings if unicode #1413
- Fix perl file access error on win skeleton cpan #1414
- Catch patchelf failures in post #1418
- fix path walking in get_ext_files #1422

5.154.3 Contributors

- @mingwandroid
- @msarahan
- @soapy1

5.155 2.0.2 (2016-09-27)

5.155.1 Enhancements

- Consider all recipes when printing output paths with --output #1332
- Lay groundwork for noarch packages with different types allowed (not just noarch_python) #1334
- Improve setting RPATH on Linux - handle relative paths better #1336
- Add GPL as a license family #1340

- Skip fixing rpath for files listed in ignore_prefix_files #1345
- Uniformly use conda's rm_rf function, not move_to_trash #1355
- Add support for alternate PKGINFO files. Adds pkginfo dependency. #1353
- Add --croot argument to conda build CLI, to specify build root folder #1358
- Do not index pkgs folder #1381 #1388
- Assert that merge destination is not a subdir of source, to avoid recursion problems #1396
- add UUID to token upload test to avoid race condition that caused intermittent CI failure #1392
- Roll source.get_dir into config.work_dir, to avoid confusion. #1400
- Improve locking in several places #1405 #1408

5.155.2 Bug fixes

- Fix guess_license_family to have LGPL -> LGPL, not public domain #1336
- Restore standard behavior with `__pycache__` folder and pyc files #1333
- Fix pyver_re to not catch python-* packages #1342
- Fix erroneous file argument to logging call #1344
- Fix convert unix -> win not creating entry point py scripts #1348
- Remove pytest timeout for tests. It is responsible for intermittent CI test crashes. #1351
- Fix retrieval of CONDA_NPY setting (only --numpy flag was being respected) #1356
- Fix --no-build-id argument to conda build that was note being respected #1359
- Fix handling of recipe folder specifications coming out blank or . #1360 #1391
- Handle conda 4.2 exceptions better for LinkErrors and PaddingErrors, better support OpenSSL custom prefix replacement #1362
- Fix indentation error leading to skip-existing not working #1364
- Fix skeletonize not passing arguments from CLI #1384 #1387 #1406
- Check if file exists before trying to use stat on it. Might avoid mmap errors. #1389
- Fix no include recipe option when input is metadata (as opposed to recipe file path) #1398
- Normalize slashes in examining files in tarfiles (always forward slashes) #1404

5.155.3 Contributors

- @gabm
- @jakirkham
- @johanneskoester
- @mingwandroid
- @msarahan
- @mwcraig
- @soapy1

- @sooneecheah
- @yoavram

5.156 2.0.1 (2016-09-06)

5.156.1 Enhancements

- Add disable_pip build option to disable conda's automatic add of pip/setuptools #1311
- Add numpy to pypi env creation if it is imported in setup.py #1289
- Improve compatibility with conda >=4.2 regarding prefixes that are too short #1323
- Delete .pyo files prior to compiling pyc files. They are considered harmful. #1328
- Add `conda build purge-all` command that cleans out built packages and build folders #1329

5.156.2 Bug fixes

- Use source.get_dir instead of config.workdir for source_files (was one level too low) #1288
- Import setuptools in windows.py to apply vc9-finding monkeypatch #1290
- Fix convert not updating subdir in index.json #1297
- Update post-build deprecation warning to state 3.0 as release for removal #1298
- Create pkgs folder if it does not exist #1299
- Fix warn_on_old_conda_build to ignore non-final release versions (alpha/beta/rc) #1303 #1315
- Remove coercion of pycache folder into flat pyc files #1304
- Fix metadata retrieval in bdist_conda #1308
- Add supplemental removal of cached packages when conda does not fully remove them #1309
- Simplify updating the package index #1309
- Straighten out when metadata member config is used, relative to config argument #1311
- Catch prefix length errors with OpenSSL's custom prefix replacement program #1312
- Replace all colons with _ in git mirror folders to avoid Windows path errors #1322
- Fix missing file contents in converted packages. Test. #1325

5.156.3 Contributors

- jakirkham
- mingwandroid
- msarahan

5.157 2.0.0 (2016-08-29)

Notes here are a consolidation of all changes between 1.21.14 and 2.0.0.

5.157.1 Enhancements

- Increase placeholder path to 255 bytes (affects only Linux/Mac. Packages need to be rebuilt to support longer embedded paths) #877
- Configuration is local, passed via config argument. No more global config. #953
- Created Python API in conda_build/api.py #953
- Separate build folders per-build to allow parallelism #953
- Add locking to allow safe parallelism #953
- Add build flag to turn off separate build folders (--no-build-id) #953
- Much greater test coverage across all modules #953
- Add CONDA_BUILD_STATE variable that reflects RENDER, BUILD, or TEST state of build #1232
- Add support for `HG_` variables in meta.yaml templates (for hg repos) #207 #1234
- Add source_files test argument in meta.yaml to copy files from source into test #1237
- add a numpy.distutils patch to jinja templating, so that fortran projects using numpy can also use jinja2 (thanks @bladwig1) #1252
- Ensure that the build environment is on PATH during all tooling and testing #1256
- Make failure due to pip requirements in meta.yaml clearer #1279
- Allow API to accept either paths to meta.yaml files or MetaData objects, for better compatibility #1281
- Implement tests to verify api stability #1283
- Add build/noarch to recipe metadata. Use for truly platform independent packages - same folder in every install. #1285

5.157.2 Bug fixes

- Fix error converting linux to win packages due to python version mismatch #481
- Fix infinite loop that occurred with circular dependencies being built #953
- Improve test data structures to allow profiling with pytest-profiling #953
- Fix version sorting in pypi skeleton generator #1238
- improve backwards compatibility* prefix build and test envs with _, so that conda can be installed in them #1242
- fix bdist_conda; add smoke test for it to Travis #1243
- fix windows entry points (duplicate bad logic) #1246
- fix inspect entry point in embedded conda.recipe #1251
- create build environment before looking for VCS in it. #1253
- fix a deadlock with recursive environment creation on encountering packages with short prefixes #1257
- with conda commands #1259

- only compile pyc files if python is in the build prefix # 1261
- remove exception catch-all in build CLI, to show actual errors more #1262
- specify full paths to activate scripts to avoid PATH conflicts with virtualenv #1266
- clean up remnants of pipbuild #1267
- remove pyc files from any source_files arguments to test in meta.yaml (avoid __file__ errors) #1272
- copy files and folders when breaking hardlinks (rather than renaming) to avoid cross-filesystem errors #1273
- add Scripts folder to prepended binary paths searched on Windows #1276
- update MACOSX_DEPLOYMENT_TARGET hard-coded value to 10.7 (better fix soon) #1278
- disallow backslashes in meta.yaml fields describing paths (e.g. always_include_files) #1280
- prevent GIT_* env vars from containing newlines #1282
- restore prefix-lengths inspect command (lost in merging) #1283

5.157.3 Restructuring

- CLI scripts have been gutted and moved to conda_build/cli. Content from them is in conda_build, in scripts without `main_` prefix. #953
- Skeleton generators have been broken out of main_skeleton.py, and consolidated into conda_build/skeletons. The contents of this folder are examined at runtime for pluggability. #953

5.157.4 Contributors

- @bladwig1
- @brentp
- @heather999
- @jakirkham
- @mingwandroid
- @msarahan
- @melund
- @pigmej

5.157.5 Testers

- @dsludwig
- @ericdill
- @jakirkham
- @mingwandroid
- @pitrou
- @srossross

5.158 2.0.0beta4 (2016-08-26)

5.158.1 Bug fixes

- improve backwards compatibility with conda commands #1259
- only compile pyc files if python is in the build prefix # 1261
- remove exception catch-all in build CLI, to show actual errors more #1262
- specify full paths to activate scripts to avoid PATH conflicts with virtualenv #1266
- clean up remnants of pipbuild #1267
- remove pyc files from any source_files arguments to test in meta.yaml (avoid __file__ errors) #1272
- copy files and folders when breaking hardlinks (rather than renaming) to avoid cross-filesystem errors #1273
- add Scripts folder to prepended binary paths searched on Windows #1276
- update MACOSX_DEPLOYMENT_TARGET hard-coded value to 10.7 (better fix soon) #1278

5.158.2 Contributors

- @dsludwig (testing)
- @ericdill (testing)
- @jakirkham (testing)
- @mingwandroid (testing)
- @msarahan
- @pitrou (testing)
- @srossross (testing)

5.159 2.0.0beta3 (2016-08-23)

5.159.1 Enhancements

- add a numpy.distutils patch to jinja templating, so that fortran projects using numpy can also use jinja2 (thanks @bladwig1) #1252

5.159.2 Bug fixes

- prefix build and test envs with _, so that conda can be installed in them #1242
- fix bdist_conda; add smoke test for it to Travis #1243
- fix windows entry points (duplicate bad logic) #1246
- fix inspect entry point in embedded conda.recipe #1251
- create build environment before looking for VCS in it. #1253
- fix a deadlock with recursive environment creation on encountering packages with short prefixes #1257

5.159.3 Contributors

- @bladwig1
- @ericdill (testing)
- @jakirkham
- @mingwandroid (testing)
- @msarahan

5.160 2.0.0beta2 (2016-08-22)

This release includes all current (1.21.14) changes made to the 1.21.x series.

5.160.1 Enhancements

- Configuration is local, passed via config argument. No more global config. #953
- Created Python API in conda_build/api.py #953
- Separate build folders per-build to allow parallelism #953
- Add locking to allow safe parallelism #953
- Add build flag to turn off separate build folders (--no-build-id) #953
- Much greater test coverage across all modules #953
- Add CONDA_BUILD_STATE variable that reflects RENDER, BUILD, or TEST state of build #1232
- Add support for `HG_` variables in meta.yaml templates (for hg repos) #207 #1234
- Add source_files test argument in meta.yaml to copy files from source into test #1237

5.160.2 Bug fixes

- Fix error converting linux to win packages due to python version mismatch #481
- Fix infinite loop that occurred with circular dependencies being built #953
- Improve test data structures to allow profiling with pytest-profiling #953
- Fix version sorting in pypi skeleton generator #1238

5.160.3 Restructuring

- CLI scripts have been gutted and moved to conda_build/cli. Content from them is in conda_build, in scripts without `main_` prefix. #953
- Skeleton generators have been broken out of main_skeleton.py, and consolidated into conda_build/skeletons. The contents of this folder are examined at runtime for pluggability. #953

5.160.4 Contributors

- @melund
- @msarahan
- @pigmej

5.161 1.21.14 (2016-08-18)

5.161.1 Bug fixes

- fix pyc compilation when egg files/folders are present #1225

5.161.2 Contributors

- @msarahan

5.162 1.21.13 (2016-08-18)

5.162.1 Enhancements

- use git -am when applying git patches, so that patches better retain git history #1222
- allow relatively pathed git submodules #1222
- add guess_license_family to pypi skeleton generator #1222

5.162.2 Bug fixes

- fix typo in convert.py

5.162.3 Contributors

- @mingwandroid
- @msarahan

5.163 1.21.12 (2016-08-17)

5.163.1 Enhancements

- Whitelist the CPU_COUNT environment variable. #1149
- Add tool for examining prefix length in existing packages #1195
- Add a conda interface layer for better compatibility with conda 4.2 #1200 #1203 #1206
- Document how to run tests #1205

- Update default versions for R (3.3.1) and Perl (5.20.3) builds #1220

5.163.2 Bug fixes

- Don't compile .py files in executable locations. Compile one at a time. #1186
- Don't force download if vcs is used as a source #1212
- Break hardlinks as a post-install step. Hard links can cause problems at package install time. #1215
- Make environment variables used by conda in environment creation always be bytestrings #1216 #1219

5.163.3 Contributors

- @jakirkham
- @kalefranz
- @msarahan

5.164 1.21.11 (2016-08-06)

5.164.1 Bug fixes

- Correct logic for printout of meta.dist determination #1174
- Attempt to use src_dir instead of WORK_DIR for directory creation #1175
- Fix escaping problem with PY_VCRUNTIME_REDIST setting #1172
- Set build prefix for win by path, not name #1172
- Quote INCLUDE and LIB env var settings for win better #1172
- Fix pypi skeleton package search #1181

5.164.2 Contributors

- @msarahan
- @pelson

5.165 1.21.10 (2016-08-02)

5.165.1 Bug fixes

- Compile files ending with .py, not py. #1163
- Move root logger to entry points, to not interfere with conda #1164 #1166
- Use setuptools entry points, rather than pre-defined scripts #1165
- Always use the long build prefix to avoid confusion #1168

5.165.2 Contributors

- @mingwandroid
- @msarahan

5.166 1.21.9 (2016-08-01)

5.166.1 Bug fixes

- Add debug option that shows conda output during build. Hide output otherwise. #1159
- Add regression test for conda metapackage command, fix missing token and user args. #1160
- Create croot (conda-bld) folder if missing before locking in render and skeleton. #1161

5.166.2 Contributors

- @msarahan

5.167 1.21.8 (2016-07-31)

5.167.1 Bug fixes

- Fix --source argument to build - was building when should only download source. #1152
- Don't try to create work folder when it exists (but is empty) #1153
- Fix a logic error with need_source_download not existing #1148

5.167.2 New Things

- Don't exit on compileall failure #1146
- Add CONDA_BUILD_RENDERING environment variable that is set during recipe rendering #1154
- Change pyc compilation to only affect files that would be packaged (not all of site-packages). Compile pyc files on py3. #1155
- Rename load_setuptools to load_setup_py_data (keep load_setuptools for compat; but show warning) #1156
- Test that condarc channels are respected in build #1157

5.167.3 Contributors

- @daler
- @minrk
- @msarahan

5.168 1.21.7 (2016-07-22)

5.168.1 Bug fixes

- Add test of requirements.txt parsing for runtime requirements #1127
- Set PY_VCRUNTIME_REDIST for VS 2015+, so that DLL linkage is used #1129
- Use os.path.normpath in find_lib #1132
- Fix path prepending in test (use only PATH, and use consolidated code) #1135
- Add dist split for channel names #1136
- Provide fallback path to render recipe when build environment is necessary for rendering #1140
- Sort package versions coming from PyPI for skeleton #1141

5.168.2 Contributors

- @mingwandroid
- @msarahan

5.169 1.21.6 (2016-07-14)

5.169.1 New Things

- Allow pass-through of setup.py options in conda skeleton pypi #680
- Allow specification of pinning numpy in conda skeleton pypi #680
- Support PEP420 namespace packages (don't barf on existing folders.) Do barf on existing files. #1074

5.169.2 Bug fixes

- Fix handling of quotes in selectors #1104
- Fix load_setuptools in jinja context. Problem was incorrect cwd in function. #1106
- Make Win activate script file extensions explicit #1107
- Warn users on failed git repo info failure, rather than crash #1108
- Remove killing MSBuild.exe at end of win build. Remove psutil dependency. #1109
- Prepend PATH before creating env, to ensure post-link script success. #1115, #1118
- Make Python tests drop out on failure appropiately on win #1122

- Make hyphenation consistent with include_recipe in meta.yaml #1124
- Use full path of test env when activating #1125

5.169.3 Contributors

- @ikalev
- @msarahan
- @mwcraig

5.170 1.21.5 (2016-07-07)

5.170.1 Bug fixes

- Make --skip-existing respect remote channels (s3, file, anaconda.org) #1102
- Reduce always_include_files glob fail exit to a warning #1101
- Fail more gracefully when finding a vcs executable fails #1100
- Add better error when PyPI fails with XMLRPC. Add tests for published examples. #1098
- Fix lack of 'call' in windows test activate script that was terminating tests early #1097
- Take newest version from PyPI when creating skeleton #1092
- Fix unicode encoding error in conda skeleton pypi #1092
- Support PEP420 namespace packages (write into existing folders, but raise error rather than overwrite existing files. #1090
- Fix an error where an intermediate None value broke jinja2 rendering #1088
- Add missing support for include_recipe in meta.yaml #1085

5.170.2 Contributors

- @ikalev
- @msarahan

5.171 1.21.4 (2016-07-05)

5.171.1 Bug fixes

- Choose newest Pypi skeleton version; fix unicode encoding in pypi metadata #1092
- Add Numpy 1.11 to all_versions dict for autocompletion #1078
- Fix MSVC 3.3/3.4 builds when Win7SDK not installed #1072
- Fix an error with build number, when build number is None or otherwise invalid #1088

5.171.2 Known issues

- Environment activation requires conda >=4.1.6. The activate.bat script does not look in the right place for the activate.d folder.
- The test suite on Linux and Mac fails the python-build, python-run, and python-build-run tests, because an errant __conda-version__.txt file is somehow present. It is not clear where it comes from, and each of these tests pass when run individually. If you have mysterious issues, and you use __conda-version__.txt or files like it, please file an issue.

5.171.3 Contributors

- @adament
- @aleksey
- @ikalev
- @msarahan

5.172 1.21.3 (2016-06-27)

5.172.1 Bug fixes

- Fix a regression in Windows, where a compiler was a hard requirement, and was not always showing up, anyway. #1049

Contributors:

- @msarahan

5.173 1.21.2 (2016-06-24)

5.173.1 Bug fixes / Improvements

- revert some MSVC activation logic to still call vcvarsall directly in build script
- fix Windows testing for binary prefix replacement (not done on win)
- Add a warning message when conda-build can't create an environment due to unsatisfiable dependencies
- Improve notion of whether a recipe uses a VCS in its metadata, or in its build

5.173.2 Known issues

- Environment activation on Windows will not work until Conda 4.1.4 is released. The activate.bat script does not look in the right place for the activate.d folder.
- The test suite on Linux and Mac fails the python-build, python-run, and python-build-run tests, because an errant __conda-version__.txt file is somehow present. It is not clear where it comes from, and each of these tests pass when run individually. If you have mysterious issues, and you use __conda-version__.txt or files like it, please file an issue.

5.173.3 Contributors

- @msarahan
- @patricksnape

5.174 1.21.1 (2016-06-22)

5.174.1 Bug fixes / Improvements

- Simplify MSVC activation, using distutil's existing logic #1036
- Correctly interpret paths returned from git on windows, trying cygpath, falling back to conda regex #1037
- Fix ability to disable anaconda upload in condarc #1043
- Change environment activation to call activation in scripts, rather than having Python store variables #1044

5.174.2 Contributors

- @msarahan
- @mwcraig
- @patricksnape

5.175 1.21.0 (2016-06-15)

5.175.1 New stuff

- Add `FEATURE_` environment variables for MKL, opt and debugging #978
- add info/about.json file that contains the "about" section of meta.yaml #941
- allow `` `--dirty` `` flag to be passed to `` `conda build` `` command. Skips download, and provides DIRTY environment variable in build scripts. #973
- Add msys2 paths to build and test environments #979
- add new x86 and x86_64 selectors for Intel platforms #986
- keep original meta.yaml in recipe folder of package; create meta.yaml.rendered in recipe folder. Neither exist when recipe not included. #1004
- add ignore_prefix_files key to build in meta.yaml. Can ignore list of files, or True to ignore all prefix files. #1008 #1009
- Automatically determine patch strip level #1011

5.175.2 Bug fixes/Improvements

- Lightened requirement that x.x be defined in both build and runtime sections. #650
- Remove info/recipe.json from build conda packages. Superseded by info/recipe/meta.yaml.rendered. #781
- Search for single and double backslashes when finding files that need prefix replacement #962
- Track undefined jinja variables and use them to decide whether to download source #964
- handle patches with p0 or p1 #969, #1011, #1020
- only set os.environ for non-None variables #981
- Don't use long prefixes on windows #985
- Fix missing encoding argument #987
- Respect proxy variables more appropriately #989
- Search packages on PyPI, rather than listing them all. Should avoid some timeout errors there. #991
- Fix unix-style paths returned from git on Windows preventing relative paths from providing Jinja2 metadata #995
- improve logic handling "dirty" downloading. Always download when not dirty. #995
- Fix post-build variables when no build section existed in original meta.yaml #999
- Activate _build and _test environments approriately, rather than manipulating PATH directly #1002
- Don't clone git submodules until after first checkout #1025
- Move check_install over from conda.install #1027

5.175.3 Deprecations

- __conda_version__.txt and other post-build methods of altering the build string are marked as deprecated. Prefer Jinja2 templates where possible. Create issues if this breaks your work.

5.175.4 Contributors

- @filmor
- @ilanschnell
- @jschueller
- @mingwandroid
- @msarahan
- @pelson
- @stuarteberg
- @whitequark

5.176 2.0.0beta (2016-06-05)

5.176.1 Compatibility breaks

- Increase placeholder path to 255 bytes (affects only Linux/Mac. Packages need to be rebuilt to support longer embedded paths) #877

5.176.2 Bug fixes/Improvements

- Respect proxy variables more appropriately #989
- Fix post-build variables when no build section existed in original meta.yaml #999
- Fix unix-style paths returned from git on Windows preventing relative paths from providing Jinja2 metadata #995
- improve logic handling "dirty" downloading. Always download when not dirty. #995
- Search packages on PyPI, rather than listing them all. Should avoid some timeout errors there. #991
- Lightened requirement that x.x be defined in both build and runtime sections. #650
- Search for single and double backslashes when finding files that need prefix replacement #962
- Fix missing encoding argument #987
- Don't use long prefixes on windows #985
- only set os.environ for non-None variables #981
- Track undefined jinja variables and use them to decide whether to download source #964
- handle patches with p0 or p1 #969

5.176.3 New stuff

- Add `FEATURE_` environment variables for MKL, opt and debugging #978
- add new x86 and x86_64 selectors for Intel platforms #986
- add info/about.json file that contains the "about" section of meta.yaml #941
- Add msys2 paths to build and test environments #979
- allow `` `--dirty` `` flag to be passed to `` `conda build` `` command. Skips download, and provides DIRTY environment variable in build scripts. #973

5.176.4 Contributors

- @filmor
- @heather999
- @ilanschnell
- @jschueller
- @mingwandroid
- @msarahan
- @pelson

- @stuarteberg
- @whitequark

5.177 1.20.3 (2016-05-13)

5.177.1 Features

- use posix metapackage for cran skeleton packaging (#956)

5.177.2 Bug fixes

- fix output of package paths (extra output was breaking tools). Add tests. (#950)
- change default of no_download_source in build.py (for compatibility with conda-build-all) (#950)
- fix regression in [] being confused for selectors (#957)

5.178 1.20.2 (2016-05-13)

5.178.1 Features

- added --token and --user flags to pass corresponding information to anaconda upload (#921)
- added conda render command that outputs a fully-rendered meta.yaml to either stdout, or to file (with --file) (#908)
- support source checkout tools specified in meta.yaml. If source checkout fails at the rendering phase, source checkout and rendering are re-done after the build environment is created. (#843, #946)
- fn is now optional when a URL specifies a filename. (#942)
- CRAN skeleton generator now uses MSYS2 for Windows support (#942)
- conda build & conda render both recursively look for meta.yaml (support conda-forge feedstock submodules) (#908)
- Whitelist MAKEFLAGS environment variable. Setting this outside conda build should take effect in your build. Parallelize on *nix by adding -j here, instead of -j${CPU_COUNT} in your build.sh. This helps on CI's, where CPU_COUNT is not always well-behaved. (#917)
- Run python_d executable on windows when debug feature is active (#724)
- add conda build flag --keep-old-work that temporarily moves your last build, then moves it back after completion. For debugging, when more than one package is involved. (#833)
- Allow selectors in imported jinja templates (#739)

5.178.2 Bug fixes

- fixed several instances wherein --skip-existing did not work (#897, #945)
- Fully render recipe before outputting build string * fixes empty spots where GIT_* info should have been (#923)
- Add MSYS2 path conversion filters to avoid issues with Win 7.1 SDK (#900)
- Address PyPI's change of URL format (#922,
- Fix invalid gcc "-m 32" flag (#916)
- Fix empty section (due to selectors) handling regression (#919)
- Fix regression in handling of VS2008 Pro (not Express + VC for Python 2.7). It is important to at least try to run vcvarsall.bat. (#913)
- Fix CPAN skeleton generator (handle missing sections better) (#912)
- Make test/requires versions match build/requires without additional pinning (#907)
- Remove hard-coded CYGWIN path from conda-build's custom PATH (#903)
- Source is downloaded before testing, fixing an issue where if build machine and some other test machine had different source, strange things happened. (#946)
- Fix regression with Python 3.x fixing shebangs (#892)
- Fix conda inspect crashes by using conda-meta info rather than filenames or dist names for package info (#947)

5.178.3 Miscellany

- restore AppVeyor testing for Windows builds (#864)
- Build py3.5 on Appveyor (#938)
- PEP8 cleanup; use flake8 rather than pyflakes (#938)
- limited scope of project locking to avoid lock conflicts between build and rendering (#923)
- set up anaconda.org build infrastructure (#924)
- on Windows, environment variables are written to the temporary bld.bat in the source work folder. (#933)

5.179 1.20.1 (2016-04-21)

- fix source/path and GIT_* issues, #801
- fix invalid assertion, #855
- environ.py refactor/clenup, #856
- Better messaging for yaml parsing errors, #862
- fix typo, #863
- make CONDA_PY and CONDA_NPY available in build.sh, #837
- execute source fetchers (e.g., git, hg) in the _build environment, #843
- use memory map rather than read() to reduce memory usage, #866
- fix svn url on Windows in checkout tool test, #867

- fix empty files bug, #869
- improve Visual Studio logic, #861
- add files in order of increasing size to improve access times to tar, #870
- VS_YEAR, VS_VERSION, VS_MAJOR and CMAKE_GENERATOR environment variables, #872

5.180 1.20.0 (2016-03-25)

- support for Lua as a built-in language (Alex Wiltschko), #719
- allow additional keys in "about" section, #831
- fix Examples directory in noarch_python, #838
- revert OS X SIP fix, part of #808, #844
- fixed race condition between removal and creation of tmp_dir on win, #847

5.181 1.19.2 (2016-03-10)

- silence some errors when subprocessing git #790
- fixes conda skeleton cran under python3 #817
- fixes some bugs introduced with the #808 otools refactor, #821, #825
- fixes #818 conda-build 1.19.1 breaks C compilation, #825
- actually fix #807 recurisive builds after conda 4.0 release, #826
- fixes #820 crash when building from a git repo on Windows, #824

5.182 1.19.1 (2016-03-09)

- Environment variables defined in the 'script_env' build section of the meta.yaml file were previously assigned the value '<UNDEFINED>' if not found in the environment. Now they are left unset and a warning is raised instead, #763.
- fix printing of NumPy 1.10 in help message, #776
- add -m32 to CFLAGS and CXXFLAGS for multilib gcc, #775
- fixes CYGWIN_PREFIX for drive letters other than C:, #788
- fixes for noarch package building on Windows, #799
- work-arounds for System Integrity Protection on OS X El Capitan, #808
- fix recurisive builds after conda 4.0 release, #813

5.183 1.19.0 (2016-01-29)

- normalize unicode in conda skeleton cran, #681
- use /bin/sh on openbsd, #707
- fail early during patching
- use symlinks=True in copytree() for SVN sources, #665
- support entry points with dots (to suppoer classes), #690
- deprecate conda pipbuild in favor of conda skeleton. #710
- fix Win references to PipBuild scripts, #723
- allow git shallow clones, #604
- remove broken license file detection code, about/license_file expects filename now
- allow pinning dependencies when building a package, #741
- fix to restore building for multiple python versions on Windows, #744
- fix building (git unrelated things) when git is not installed, #745
- enable tab completion for the packages argument of the conda inspect commands, #748

5.184 1.18.2 (2015-11-19)

- move path prepending to function for uniformity, #601
- improve yaml loading, #603
- allow jinja2 templates to be located in current conda environment, #578
- fix NPY_VER for versions >= 1.10 (Should be '1.10', not '1.1.0'), #660
- create jinja2 environment with 'strict' mode for undefined values, #661
- add a method to shell out and execute a command through subprocess, #621

5.185 1.18.1 (2015-10-16)

- allow config system to handle versions with have more than a single digit in the minor version, #626
- fix None encoding bug, #614
- add missing Python version when adding Python to test specs
- add features to build string
- improve yaml loading (you don't have to quote version numbers anymore, eg. if the version is 3.1), #603

5.186 1.18.0 (2015-10-01)

- develop options --build_ext, --clean_build_ext, #512
- fix directory not existing when using --no-include-recipe option
- add support for multiple rpaths on OS X to conda inspect
- don't add 'np' to build string when package depends on numpy, but not a specific version
- be more explicit when numpy version is included in dependency specs, #573
- correctly remove egg directories on Windows, #536
- add new option msvc_compiler to build section for forcing MSVC compiler version
- add new command conda inspect channels --test-installable
- fix a Unicode issue with conda skeleton cpan
- when auto-adding python spec to execute run_test.py, don't require a specific version
- add uninstall option to conda develop
- give a better error message in skeleton pypi for packages with invalid urls
- don't try to test skipped recipes
- don't exit on a skipped recipe
- recursively build packages from unsatisfiable install hints
- make recursive building work better with --skip-existing
- update CONDA_R to 3.2.2
- fix encoding issues with git_info on Windows
- test Python 3.5 in Travis CI
- add support for absolute rpaths on Linux

5.187 1.17.0 (2015-08-24)

- quote set calls in bld.bat
- use the trash on Windows when deleting environments, see #521
- improve documentation in noarch_python source
- rename 'binstar' -> 'anaconda', see #519
- allow blank sections in meta.yaml, see #533
- add --no-include-recipe option to conda-build, see #535
- add ability to add license file in info/license.txt, see #545
- don't recursively build recipes more than once, #538
- .git can be a file, #537

5.188 1.16.0 (2015-07-30)

- handle trailing slashes in package names in conda skeleton cran
- Cygwin git now works correctly.
- the prefix itself is now included in the PATH in the test script on Windows (previously it was just the Scripts directory)
- by default, recipes that runtime depend on numpy will no longer depend on an explicit version of numpy. The old behavior is still available by setting the CONDA_NPY environment variable or using conda build --numpy
- add py35 variable to selector namespace
- improve conda-meta untracked files error message
- fix conda build --help in Python 2
- add conda_build.sub_commands object which is a list of conda sub-commands

5.189 1.15.0 (2015-07-22)

- fix conda skeleton cran --update-outdated --output-dir .
- add argcomplete completers for recipes, --python, --numpy, --R, CRAN packages (with conda skeleton cran), and PyPI packages (with conda skeleton pypi)
- conda develop now relinks object files on OS X (#490)
- allow a glob for always-include-files
- allow an extra section in meta.yaml, with free-form content (#483)
- don't echo environment variables when building on Windows (#274)
- add conda build --skip-existing
- show default in help for conda skeleton --output-dir
- add --update-outdated option to conda skeleton cran
- skeleton: fix noarch_python option when build_comment is "#"
- don't allow to build a package with the conda-meta directory
- automatically remove a package of the recipe itself if it is installed as a build dependency
- allow 'extra' key in meta.yaml, see #483
- move echo command in Windows build, see #274
- add regex to always included files, see #484
- add strings in conda.config.non_x86_linux_machines, e.g. "ppc64le", as selector variables (renames armv6 to armv6l)

5.190 1.14.1 (2015-06-29)

- add --size option to change RSA modulus length when generating RSA key pairs (defaults to 2048 bit)
- make use of Crypto.Signature.PKCS1_PSS module, see #469
- update default for CONDA_R to 3.2.0
- manually install dependencies of recommended R packages in the build.sh, #457
- fix issues when git commits have non-ASCII characters, #458
- catch tarfile.ReadError in conda index, #460

5.191 1.14.0 (2015-06-16)

- add support for signing packages, and indexing them, #430
- removing LIBRARY_PATH and INCLUDE_PATH build environment variables on Unix, they where originally added in #228, but are causing problem for some people and are not really necessary
- don't rename meta.yaml to meta.yaml.orig in the recipe that is copied into built packages
- handle links to libraries that exist in multiple places better on OS X
- add --no-remove option to conda index
- various fixes for --python, --numpy, --perl, and --R
- various cleanups for the command documentation
- fix conda skeleton pypi --pypi-url
- don't add the module name to the import tests in conda skeleton pypi
- add --groupby option to conda inspect linkages
- fix some incorrect "not found" instances from conda inspect linkages on OS X
- don't include versions with restrictions in the build string
- don't fail if conda-build cannot be found for the version check
- remove special logic if the username on Windows is "builder"
- conda skeleton pypi: add --noarch-python
- fix issue with filenames with spaces in conda convert
- place noarch packages in the noarch directory
- handle tests_require in conda skeleton pypi
- pipbuild: don't check if package already exists
- skeleton pypi: remove --no-download option
- add noarch option to pipbuild
- add ability to sign packages

5.192 1.13.0 (2015-05-19)

- skeleton pypi: fail better for packages with bad urls
- fix summary in bdist_conda
- fix compiling pyc files in Python 3
- convert: correctly set the subdir key in the metadata
- add --git-tag to skeleton cran
- include LANG in the build environment
- export proxy environment variables
- fix conda skeleton cran --cran-url
- set CONDA_DEFAULT_ENV in the build environment
- fix conda index -c
- correctly extract .tar.Z files
- avoid infinite loops in conda skeleton pypi --recursive
- add --all to conda inspect linkages and conda inspect objects
- add --manual-url to skeleton pypi
- fix issue where 'conda index' with old packages would create bad metadata
- resolve circular dependencies in conda-skeleton (#409)
- use versioneer 0.14 (#385)
- always_include_files errors out (exits) on files that aren't there (#387)
- automatically lowercase the package name in bdist_conda (see aplpy/aplpy#259)

5.193 1.12.1 (2015-04-28)

- fix regression in always_include_files that causes build failure (#386)

5.194 1.12.0 (2015-04-10)

- correctly fix egg directories that are part of the package
- use the --force-rpath flag to patchelf
- update MACOSX_DEPLOYMENT_TARGET to 10.6
- fix running tests for Python packages whose version differs from the version in conda
- fix some Python 3 issues with pipbuild
- don't allow packages to depend on themselves
- allow to use the r- prefix in conda skeleton cran
- make recommended r packages depend on r-base in skeleton cran

- new post-build logic on OS X. All libraries on OS X now include LC_RPATH, which points to the environment lib directory, and use and install name using @rpath
- don't set DYLD_FALLBACK_LIBRARY_PATH in cran recipes (the new LC_RPATH logic on OS X makes this unnecessary)
- fix conda build --build-only when the long build prefix is used
- make conda inspect linkages work on OS X
- don't hide the traceback for maximum recursion depth exceeded errors
- add conda inspect objects, for inspecting object files in packages (OS X only)
- add --untracked flag to conda inspect linkages
- build R packages against a specific version of R
- decompress .tar.z files
- add support for GitHub urls in conda skeleton cran

5.195 1.11.0 (2015-03-05)

- add 'script_env' key in build section of meta.yaml file, which is a list of environment variable names which are made available in the build script. If a variable is listed here, but is not in the environment, the value '<UNDEFINED>' is assigned.
- Handle OSError in conda index
- Fix how the PATH environment variable is set on Windows
- Remove the work directory earlier in the build
- Give a helpful error message for dependencies like "python >= 2.7"
- Add CYGWIN_PREFIX environment variable on Windows
- Handle list requires in skeleton pypi
- Correctly fail if the Windows bld.bat exits 1
- Give a better error message if no urls can be found for a package
- Add __main__ to allow python -m conda_build
- %R% is now set to R.exe instead of R.bat on Windows
- Write the build script to the source directory for build/script instead of the recipe directory.
- Handle non-directories in copy_into (avoids an OSError, #332)
- Halt the build on YAML error without jinja2
- Clone git sources with the --recursive option
- Add --channel and --override-channels to conda build. -c is changed to mean --channel instead of --check.
- Add --check-md5 flag to conda index
- Look for vcvarsall.bat from the Microsoft Visual C++ Compiler for Python 2.7
- Use PyPi XMLRCP client search in order to ignore case for PyPi package names in pipbuild

5.196 1.10.2 (2015-02-10)

- don't set the GIT_* environment variables when the source is not a git repo
- skeleton cran: add extra metadata from CRAN to the recipe
- skeleton pypi: fix there not being a fragment in a url
- don't match comment only lines as selectors

5.197 1.10.1 (2015-02-06)

- greatly improved ability to create noarch_python packages, #317
- added 'subdir' key to info/index.json
- allow url to be a list or urls, which are tried until one works
- use quotes instead of !!str for versions from the conda skeleton commands
- add conda skeleton cran to generate recipes for packages from CRAN
- add support for adding a readme to a recipe
- add a --quiet option to conda convert

5.198 1.10.0 (2015-01-15)

- automatically convert absolute symlinks to paths in the build prefix to relative ones.
- error if there are symlinks to the source directory.
- use the placeholder prefix in text files rather than the build prefix
- allow non-Python packages to be converted to other platforms with conda convert
- new command conda inspect for inspecting packages. The only subcommand so far is conda inspect linkages, which shows the dynamic linkages of the shared object files on Linux.
- correctly handle Unix style has_prefixes on Windows
- run the tests in Binstar build
- only modify egg directories that are part of the package being built
- don't exclude .dylib files from prefix replacement
- ability to build noarch packages
- allow specifying files to always include
- fix for building dependencies in some cases
- print the correct thing for binary files detected with a prefix

5.199 1.9.1 (2014-11-18)

- set PYTHONNOUSERSITE=1 while running build scripts
- conda index: add error if they try to mix their packages into Anaconda channel
- fix building recipes with local git urls in Windows
- warn if conda-build is out of date

5.200 1.9.0 (2014-10-22)

- adapt tests for Windows
- don't use the long build prefix if the short build prefix is already long
- support rewriting library load path for libraries that are in subdirectories of lib/ on OS X
- allow 'git_rev' as a valid key in 'source', which is identical in behavior to git_branch and git_tag (all checkout the given revision)
- also grab the full HEAD sha1 and shove it into the environmental variable GIT_FULL_HASH
- automatically detect text prefix files
- add detect_binary_files_with_prefix flag to meta.yaml to automatically detect binary files with the prefix and add them to binary_has_prefix_files
- fix git_info when the author or commit message contains Unicode characters
- allow to pass a url to skeleton pypi
- add NPY_VER environment variable
- fix conda convert --show-imports
- give better error message when encountering a corrupt tarfile in conda index
- print{ some more helpful information about what is going on at the beginning of a build
- allow source/path in the meta.yaml to specify a path to the source (which can be relative to the recipe directory)
- support xz files in Python 3 without requiring unxz
- put spaces after skeleton pypi comments
- correctly detect when to preserve the egg directory and depend on setuptools in skeleton pypi
- set LIBRARY_PATH and INCLUDE_PATH when building on Unix
- allow selectors to have text after them if they are in a comment
- add CPU_COUNT environment variable to the build

5.201 1.8.2 (2014-09-19)

- add substantially more tests
- add ability to set additional rpath directories using build/rpaths
- patch command on windows no-longer takes the --binary option
- fix post processing so that name-space packages can be 'flattened' into a single directory
- don't remove the .svn directory when doing a svn checkout

5.202 1.8.1 (2014-09-03)

- has_prefix paths must always use /, even on Windows
- fix bug in Windows Python 2
- add .travis.yml
- allow recipes to use requirements.txt
- fix building a package that has Mach-O stub files
- fix recursive package building
- handle empty data size in pypi
- allow an explicitly set empty version string

5.203 1.8.0 (2014-08-22)

- add ability to convert Golke's Windows packages into conda packages, use "conda convert <Gohlke>.exe". See also: http://www.lfd.uci.edu/~gohlke/pythonlibs/
- handle spaces on Windows better
- add croot to the module level of conda_build.config for backwards compatibility
- changed extra long prefix for building to _placehold...

5.204 1.7.1 (2014-08-20)

- add --all-extras flag to skeleton pypi
- automatically use the long or short build prefix as needed
- fix to allow specification of full major.micro.minor for numpy, python, and perl (only add the build string if it is an exact major.minor or major.minor.patch version. If it is an inequality, it is not added)

5.205 1.7.0 (2014-08-05)

- better install_requires parsing for skeleton pypi
- the build environment from conda-build is now called _build___... This is so that recipes with binary_has_prefix_files build against a sufficiently long prefix.
- don't overwrite the input file in convert
- fix a bug related to the prefix_files.txt file
- show the download bytes in human form in skeleton pypi
- make patchelf error message clearer
- fix some issues with the git describe environment variables
- improve shebang line modifications with python.app
- show the download bytes in human form in skeleton pypi

5.206 1.6.1 (2014-07-29)

- fix an issue building with a git repository in Python 3

5.207 1.6.0 (2014-07-29)

- don't fail on MachO stub files on OS X
- add some git describe related environment variables when the source is a git repository
- add --python and --numpy flags to conda build, which do the same thing as setting CONDA_PY and CONDA_NPY
- allow a . in CONDA_PY and CONDA_NPY
- correctly catch RuntimeError
- fix an issue building some packages on Windows
- make skeleton pypi --recursive work with versioned dependencies
- some additional type checking for meta.yaml
- always include numpy in skeleton pypi

5.208 1.5.0 (2014-07-03)

- add bdist_conda
- fix features and track_features
- detect files with the build prefix automatically on Windows

5.209 1.4.0 (2014-07-01)

- fix skeleton pypi behind a proxy
- add binary_has_prefix_files, which does a binary prefix replacement
- fix skeleton pypi on Windows
- allow the git_url to be a relative path to the recipe directory

5.210 1.3.5 (2014-06-04)

- the yaml script tag was only written if build.sh already existed, see issue #105
- use tests_require to fill in test requirements in the setuptools template, see issue #107

5.211 1.3.4 (2014-06-02)

- add --build-only and --post flags to conda-build
- add conda convert -p all
- allow to set __conda_buildnum__.txt and __conda_buildstr__.txt, analogous to __conda_version__.txt

5.212 1.3.3 (2014-04-28)

- add SHA256 support for downloaded source
- conda convert now creates platform directories, like win-32/converted_package.tar.bz2
- allow to specify the version of python, numpy, or perl in the meta.yaml (it will ignore the CONDA_PY, CONDA_NPY, or CONDA_PERL, respectively, in this case)

5.213 1.3.2 (2014-04-15)

- allow changing conda-bld directory (which is by default <root>/conda-bld when the conda root is not writable, and ~/conda-bld otherwise), to be changed by CONDA_BLD_PATH environment variable or conda-build: root-dir: in condarc file
- add build/has_prefix_files
- remove broken conda-build/build_dest
- fix build in Windows Python 2
- add --quiet option to conda-build
- add check for characters in package name dependencies
- add .class to the object file extension blacklist
- removed build/no_softlink in favor of build/no_link with list of glob syntax

5.214 1.3.1 (2014-03-25)

- add conda metapackage command
- fix recursive conda building when dependencies are nested more than one level deep.
- fix build in Windows Python 2
- fix skeleton pypi for packages whose setup.py has __future__ imports
- add conda pipbuild command which uses a simple recipe based on pip install to build a conda package
- fix skeleton pypi and pipbuild when package data does not have classifiers
- add a basic conda develop command
- add the --recursive option to conda skeleton pypi
- conda skeleton pypi no longer asks about single line licenses
- conda skeleton pypi now queries pypi case insensitively
- conda skeleton pypi now works in a different conda environment. This avoids anything bad that might happen when trying to get the metadata from the package from messing up the root environment.
- conda skeleton pypi now patches distribute directly. This is more robust than trying to insert a patch into setup.py, as was done previously.
- allow to set the version post-build by writing a __conda_version__.txt file to the source directory.
- add ability to skip binary relocation step by setting "build/binary_relocation: False" in meta.yaml

5.215 1.3.0 (2014-03-14)

- add skeleton for CPAN Perl packages, issue #53. Unlike the PyPI skeleton, it supports a --recursive option to also generate the recipes for all of the dependencies of a given module/distribution
- add support for run_test.pl and Perl import tests when building Perl packages
- add CONDA_PERL environment variable support for determining which version of Perl to build packages for. Unlike CONDA_PY, this must include the full version with periods (e.g., 5.18.2)
- automatically build packages for dependencies if the recipe is present in the current working directory
- fix a number of Python 3 compatibility issues, issue #59
- work with source files with uppercase suffixes
- switch from chrpath to patchelf on Linux, issue #57
- don't use hard-coded msvc path
- sort import tests from skeleton pypi

5.216 1.2.1 (2014-02-25)

- added conda-build/build_dest option to condarc

5.217 1.2.0 (2014-02-13)

- make sure WORK_DIR exists
- use MSVC 2010 for Python 3
- include the summary with conda skeleton pypi
- fix object detection on Python 3
- update default CONDA_NPY to 18

5.218 1.1.0 (2014-02-06)

- add ability to use templates in conda recipes
- remove fallback to <root>/conda-recipes, i.e. conda-build always expects the full path to the recipe
- export PKG Build Number
- add pre-link to package, when it is found in recipe
- allow to add run_test.sh or run_test.bat, which will be run automatically during the test phase.
- Test commands from the test/commands section of meta.yaml are run from bash on Linux and OS X and batch on Windows (previously they were run using Python's subprocess).
- all environment variables from the build process are available during the tests, except with the build prefix replaced with the test prefix.

5.219 1.0.0 (2014-01-24)

- initial release
- includes conda-build, conda-convert, conda-index, conda-skeleton
- depends on new conda version 3
- add license to info/index.json object

INDEX

B

build string
    terminology, 14

C

canonical name
    terminology, 14

F

filename
    terminology, 14

N

name, *see* build string, *see* canonical name, *see* package name, *see* package version

P

package name
    terminology, 14
package spec, *see* package specification
package specification
    terminology, 14
package version
    terminology, 14

T

terminology
    build string, 14
    canonical name, 14
    filename, 14
    package name, 14
    package specification, 14
    package version, 14