nature methods

SUPPLEMENTARY INFORMATION

https://doi.org/10.1038/s41592-018-0046-7

In the format provided by the authors and unedited.

Bioconda: sustainable and comprehensive software distribution for the life sciences

Björn Grüning[1,12], Ryan Dale[2,12], Andreas Sjödin[3,4], Brad A. Chapman[5], Jillian Rowe[6], Christopher H. Tomkins-Tinch[7,8], Renan Valieris[9], Johannes Köster[10,11]* and The Bioconda Team[13]

[1]Bioinformatics Group, Department of Computer Science, University of Freiburg, Freiburg, Germany. [2]Laboratory of Cellular and Developmental Biology, National Institute of Diabetes and Digestive and Kidney Diseases, US National Institutes of Health, Bethesda, MD, USA. [3]Division of CBRN Security and Defence, FOI–Swedish Defence Research Agency, Umeå, Sweden. [4]Department of Chemistry, Computational Life Science Cluster (CLiC), Umeå University, Umeå, Sweden. [5]Harvard T.H. Chan School of Public Health, Boston, MA, USA. [6]Center for Genomics and Systems Biology, Genomics Core,, NYU Abu Dhabi,, Abu Dhabi,, United Arab Emirates. [7]Department of Organismic and Evolutionary Biology, Harvard University, Cambridge, MA, USA. [8]Broad Institute of MIT and Harvard, Cambridge, MA, USA. [9]Laboratory of Bioinformatics and Computational Biology, A. C. Camargo Cancer Center, São Paulo, Brazil. [10]Algorithms for Reproducible Bioinformatics, Genome Informatics, Institute of Human Genetics, University Hospital Essen, University of Duisburg–Essen, Essen, Germany. [11]Medical Oncology, Dana Farber Cancer Institute, Harvard Medical School, Boston, MA, USA. [12]These authors contributed equally: Björn Grüning and Ryan Dale. [13]A full list of authors and affiliations is available as Supplementary Table 1. *e-mail: johannes.koester@uni-due.de

Supplement to Bioconda: A sustainable and comprehensive software distribution for the life sciences

S1 Software management with Conda

Via the Conda package manager, installing software from Bioconda becomes very simple. In the following, we describe the basic functionality (see Fig. S1f) assuming that the user has access to a Linux or macOS terminal. After installing Conda (https://conda.io/miniconda.html), the first step is to set up the Bioconda channel via:

```
$ conda config --add channels conda-forge
$ conda config --add channels bioconda
```

Now all Bioconda packages are visible to the Conda package manager. For example, the software CNVkit[1], can be searched for with

```
$ conda search cnvkit
```

in order to check if and in which versions it is available. Alternatively, packages can be searched interactively at http://bioconda.github.io. CNVkit can be installed with:

```
$ conda install cnvkit
```

CNVkit needs various dependencies from Python and R, which would otherwise have to be installed in separate manual steps (Supplementary Fig. S1c). The above command installs all dependencies needed by CNVkit, including Python and R themselves. Conda enables updating and removing all these dependencies via one unified interface.

A key benefit of Conda is the ability to define isolated, shareable software environments. This can happen ad-hoc, or via YAML (https://yaml.org) files. For example, the following file defines an environment consisting of Salmon[5] and DESeq2[6]. Here we specify the exact versions to install, though this is not strictly necessary as the latest available version will be installed by default:

```
channels:
  - bioconda
  - conda-forge
  - defaults
dependencies:
  - bioconductor-deseq2 =1.16.1
  - salmon =0.8.2
  - r-base =3.4.1
```

Assuming that the above environment specification is stored in the file `env.yaml`, an environment `my-env` containing the specified requirements and all of their dependencies, including R itself, can be created via the command:

```
$ conda env create --name my-env --file env.yaml
```

Creating an environment does in general not mean that Conda will install new copies of all required packages. Instead, whenever possible, it will use *hard links* to a single central installation of each package, such that environments are lightweight even if hundreds of packages are involved. To use the commands installed in above new environment, it must first be *activated* by issuing the following command:

```
$ source activate my-env
```

Multiple environments can exist on the same system and only the packages in the currently-activated environment will be accessible.

Within the activated environment, R, Salmon, and DESeq2 are available in exactly the defined versions. For example, Salmon can be executed with:

```
$ salmon --help
```

It is possible to modify an existing environment by using `conda update`, `conda install` and `conda remove`. For example, we could add a particular version of Kallisto[7] and update Salmon to the latest available version with:

```
$ conda install kallisto=0.43.1
$ conda update salmon
```

Finally, the environment can be *deactivated* again with the following command:

```
source deactivate
```

S1.1 How isolated software environments enable reproducible research

With isolated software environments as shown above, it is possible to define an exact version for each package. This increases reproducibility by eliminating differences due to implementation changes. Note that above we also pin an R version, although the latest compatible one would also be automatically installed without mentioning it. To further increase reproducibility, this pattern can be extended to all dependencies of DESeq2 and Salmon and recursively down to basic system libraries like zlib and Boost (https://www.boost.org). Environments are isolated from the rest of the system, while still allowing interaction with it: e.g., tools inside the environment are preferred over system tools, while system tools that are not available from within the environment can still be used. Conda also supports the automatic creation of environment definitions from already existing environments. This allows to rapidly explore the needed combination of packages before it is finalized into an environment definition:

```
$ conda env export --name my-env > full-env.yaml
```

This file can then be shared with other researchers, who, with the following two commands, can exactly reproduce the same environment on their system and then activate that environment to use it:

```
$ conda env create --name from-collaborator --file full-env.yaml
$ source activate from-collaborator
```

In combination with workflow management systems like Galaxy[8], bcbio-nextgen (https://github.com/chapmanb/bcbio-nextgen), and Snakemake[9] that interact directly with Conda, a data analysis can be shipped and deployed in a fully reproducible way, from description and automatic execution of every analysis step down to the description and automatic installation of any required software. While this ensures that exactly the same software is used, the underlying operating system and used hardware can still have influence on the obtained results. While in theory robust results should not be affected by such aspects, it is important to note that a Conda based approach can always be combined with virtual machines and container images. The hosting of Bioconda packages is guaranteed by Anaconda Inc. without time limit. Nevertheless, sustainability can be further improved by archiving a workflow together with all used packages via Zenodo (https://zenodo.org), for example when using Snakemake (http://snakemake.readthedocs.io/en/v4.

3.0/snakefiles/deployment.html#sustainable-and-reproducible-archiving). Naturally, above notion of reproducibility only involves the ability to rerun *in silico* experiments under the same technical circumstances and with the same data. It does not free the researcher from properly designing experiments.

S2 Relation to container and module infrastructure

Reproducible software management and distribution is enhanced by other current technologies. Conda integrates well with environment modules (http://modules.sourceforge.net/), a technology used nearly universally across HPC systems. An administrator can use Conda to easily define environment modules or use Conda itself as an environment manager in order to maintain software stacks for multiple labs and project-specific configurations. Another approach to reproducibility is to use containers, popularized by Docker, which provide a way to publish an entire software stack down to the operating system. They provide greater isolation and control over the environment that software is executed in, but at the expense of some customizability. Conda complements container-based approaches. Where flexibility is needed, Conda packages can be installed directly on the system. Where the uniformity of containers is required, Conda can be used to build images, avoiding the nuanced installation steps that would ordinarily be required to build and install software within an image. In fact, for each Bioconda package, our build system automatically builds a minimal Docker image containing that package and its dependencies, which is subsequently uploaded and made available via the Biocontainers project[2]. As a consequence, every built Bioconda package is available not only for installation via Conda, but also as a container via Docker, Rkt (https://coreos.com/rkt), and Singularity[3], such that the desired level of reproducibility can be chosen freely[4].

S3 New recipes, maintenance and quality assurance in the Bioconda project

To ensure reliable maintenance of thousands of packages, we use a semi-automatic, agent-assisted development workflow (Supplementary Fig. S1d), orchestrated by a suite of tools we authored and maintain (*bioconda-utils*, https://github.com/bioconda/bioconda-utils). All Bioconda recipes are hosted in a GitHub repository (https://github.com/bioconda/bioconda-recipes). Both the addition of new recipes and the update of existing recipes in Bioconda is handled via *pull requests*. A contributor opens a pull request on the GitHub repository with a modified version of one or more recipes, and these changes are automatically compared against the current state of Bioconda. Once a pull request arrives, our infrastructure performs several automatic checks. Problems discovered in any step are reported to the contributor and further progress is blocked until they are resolved. First, the modified recipes are checked for syntactic anti-patterns, i.e., formulations that are syntactically correct but bad style (termed *linting*). This process ensures consistency across all submitted recipes, serving as an initial quality-control step and easing the automated maintenance of recipes. Second, the modified recipes are built on Linux and macOS, via a cloud based, free-of-charge service (https://travis-ci.org). Successfully built recipes are tested (e.g., by running the generated executable). Since Bioconda packages must be able to run on any supported system, it is important to check that the built packages do not rely on particular elements from the build environment. Therefore, testing happens in two stages: (a) test cases are executed in the full build environment and (b) test cases are executed in a minimal Docker (https://docker.com) container which purposefully lacks all non-common system libraries. Hence, a dependency that is not explicitly defined will lead to a failure in the latter, more stringent test. Once the *build* and *test* steps have succeeded, a member of the Bioconda team reviews the proposed changes and, if acceptable, merges the modifications into the official repository. Upon merging, packages are uploaded to the hosted Bioconda channel (https://anaconda.org/bioconda), where they become available via the Conda package manager. When a Bioconda package is updated to a new version, older builds are generally preserved, and recipes for multiple older versions may be maintained in the Bioconda repository.

Above process appears to scale well with the growing number of recipes and contributors (Supplementary Fig. S1a,b). The usual turnaround time of the workflow is short (Supplementary Fig. S1e): 61% of the pull requests are merged within 5 hours. Of those, 36% are even merged within 1 hour. Only 18% of the pull requests need more than a day. Hence, publishing software in Bioconda or updating already existing packages can be accomplished typically within minutes to a few hours.

Using Bioconda as a service to obtain packages for local installation entails trusting that (a) the provided software itself is not harmful and (b) it has not been modified in a harmful way. Ensuring (a) is up to the user. In contrast, (b) is handled by our workflow. First, source code or binary files defined in recipes are checked for integrity via MD5 or SHA256 hash values. Second, all review and testing steps are enforced via the GitHub interface. This guarantees that all packages have been tested automatically and reviewed by a human being. Third, all changes to the repository of recipes are publicly tracked, and all build and test steps are transparently visible to the user. Finally, the automatic parts of the development workflow are implemented in the open-source software *bioconda-utils*. In the future, we will further explore the possibility to sign packages cryptographically.

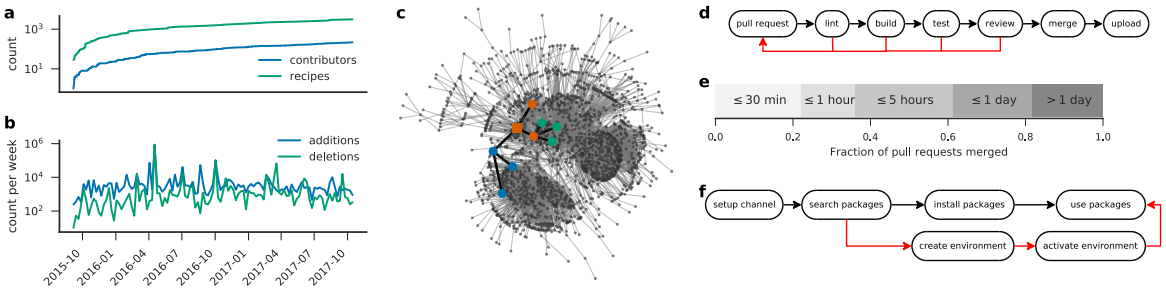

Figure S1: Development, dependency structure, workflow, and turnaround time. (a) contributing authors and (b) added recipes over time. (c) largest connected component of directed acyclic graph of Bioconda packages (nodes) and dependencies (edges). Highlighted is the induced subgraph of the CNVkit[1] package and its dependencies spanning Python, C/C++, and R ecosystems (node coloring as defined in Fig. 1a in main text, squared node represents CNVkit). (d) GitHub based development workflow: a contributor provides a pull request that undergoes several build and test steps, followed by a human review. If any of these checks does not succeed, the contributor can update the pull request accordingly. Once all steps have passed, the changes can be merged. (e) Turnaround time from submission to merge of pull requests in Bioconda. (f) Workflow for using Conda packages. After the Bioconda channel has been set up, packages can be searched and installed directly. Alternatively, isolated software environments can be created.

References

1. Talevich E, Shain AH, Botton T, Bastian BC. CNVkit: Genome-Wide Copy Number Detection and Visualization from Targeted DNA Sequencing. *PLoS Comput Biol* 2016; **12**: e1004873.

2. da Veiga Leprevost F, Grüning BA, Alves AS, Röst HL, Uszkoreit J, Barsnes H *et al.*. BioContainers: an open-source and community-driven framework for software standardization. *Bioinformatics* 2017; **33**: 2580–2582.

3. Kurtzer GM, Sochat V, Bauer MW. Singularity: Scientific containers for mobility of compute. *PLOS ONE* 2017; **12**: e0177459.

4. Grüning B, Chilton J, Köster J, Dale R, Goecks J, Backofen R *et al.*. Practical computational reproducibility in the life sciences. *bioRxiv preprint* 2017. doi:10.1101/200683.

5. Patro R, Duggal G, Love MI, Irizarry RA, Kingsford C. Salmon provides fast and bias-aware quantification of transcript expression. *Nat Methods* 2017; **14**: 417–419.

6. Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol* 2014; **15**: 550.

7. Bray NL, Pimentel H, Melsted P, Pachter L. Near-optimal probabilistic RNA-seq quantification. *Nat Biotechnol* 2016; **34**: 525–7.

8. Afgan E, Baker D, van den BM, Blankenberg D, Bouvier D, Čech M *et al.*. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic Acids Res* 2016; **44**: W3–W10.

9. Köster J, Rahmann S. Snakemake–a scalable bioinformatics workflow engine. *Bioinformatics* 2012; **28**: 2520–2522.