Environment management for scientific computing

A computer used for scientific analysis must be set up for the work, much as a bench is set up for an experiment. An environment manager creates an isolated, fully specified set of software dependencies for each project that can be shared with collaborators or restored years or more later without manual archaeology.

The one-time install takes about ten minutes; creating an environment for a new project adds roughly two minutes. This is the full overhead. Skipping it works until an analysis needs to be revisited after a software update, at which point reconstructing a working environment from scratch typically takes longer than the setup would have.

This protocol uses Conda, a fast, lightweight environment manager widely used in scientific computing, installed via Miniforge using Mamba as the solver. It handles Python and R packages as well as most command-line tools used in bioinformatics and data science in a unified manner.

Risk assessment

- Installing software from unverified sources can introduce malware or compromise data integrity.
- Mixing system-wide and user-level installs can also leave the system in a state that is hard to recover.

▷ Install package managers and packages only from reputable sources: official conda-forge and bioconda channels, the official miniforge installer, your operating system's vendor repositories.

▷ Avoid running unfamiliar shell installers piped from the internet without inspecting them first. Conda and mamba operate at the user level and should not require `sudo`; if a command asks for it, treat that as a signal to stop and check.

Reviewed: May 10, 2026

Procedures

›› *One-time install on a new machine*

(1.) Download the miniforge installer for your OS and CPU architecture from Miniforge, then run it.

```
bash Miniforge3-$(uname)-$(uname -m).sh
```

Resource: Miniforge (https://github.com/conda-forge/miniforge) is a Conda edition that uses free and openly-licensed packages from the conda-forge project by default which avoids accidental violations of the Anaconda terms of service. It takes up significantly less space and installs much faster than Anaconda.

Hint: Pick `Miniforge3-MacOSX-arm64.sh` for Apple Silicon, `…-x86_64.sh` for Intel Mac or Linux. Accept the license, install into the default `~/miniforge3` path.

Critical: Do not run the installer with `sudo`. Conda installs under your user account and never needs administrator rights.

(2.) Open a new terminal. Verify the install and set the channel configuration.

```
mamba init
mamba info
mamba config --add channels bioconda
mamba config --add channels conda-forge
mamba config --set channel_priority strict
mamba install -n base conda-lock
```

This is why: Strict priority prevents the solver from mixing packages across channels (a common source of broken environments). `--add` prepends to the channel list, so the last call wins; adding conda-forge second places it above bioconda.

This is why: `conda-lock` is a tool rather than a project dependency and belongs in base alongside mamba, where it is available to all projects without cluttering individual environments.

›› *Per-project environment workflow*

(1.) Create one environment per project repository, named after the repo. Create it from an existing environment specification file `environment.yml`, or from scratch. Specify the Python version explicitly to avoid silent upgrades when conda-forge advances.

```
mamba env create -n my-project -f environment.yml
mamba activate my-project
```

If no environment file exists yet:

```
mamba create -n my-project python=3.12
mamba activate my-project
```

> *Critical:* Keep the `base` environment empty except for mamba itself. Installing project packages into `base` is the most common path to a fragile installation.

> *This is why:* Per-project environments are disk-cheap because mamba stores each package once in a shared cache (`~/miniforge3/pkgs/`) and hardlinks it into every environment that uses it. So, ten environments using "numpy" occupy the same disk space as one. The exception is packages installed via `pip`, which copy rather than hardlink. Prefer the conda-forge or bioconda version when one exists.

> *Hint:* To create the environment in a project-local directory instead of the default `~/miniforge3/envs/`, use `--prefix ./env` and activate with `mamba activate ./env`; useful for projects that ship their environment with the repository.

(2.) Install all packages the project needs (Python libraries, R packages, and command-line tools), then export and lock the environment immediately.

```
mamba install numpy pandas scipy matplotlib samtools bwa multiqc
mamba env export --from-history > environment.yml
conda-lock lock -f environment.yml
```

> *Critical:* Install command-line bioinformatics tools (`samtools`, `bwa`, `bcftools`, `nextflow`, etc.) via mamba from bioconda, not through Homebrew or apt. System-installed tools are invisible to `environment.yml` and will not travel with the project.

> *This is why:* `--from-history` exports only the packages you explicitly requested, producing a portable file that re-solves on another machine. Without it, the export pins the entire transitive graph including platform-specific build hashes.

> *This is why:* The `environment.yml` expresses intent ("this project needs numpy and pandas"); the lockfile expresses fact ("on this date the resolution was these exact versions"). Commit both to the project repository.

(3.) When you need a new package mid-project, install it, then re-export and re-lock in the same operation.

```
mamba install seaborn scikit-learn
mamba env export --from-history > environment.yml
conda-lock lock -f environment.yml
```

> *Hint:* This is the same pattern as the initial populate step. Treat export and lock as inseparable from install. The environment file and lockfile should always reflect the current state of the environment.

Finding packages

(1.) Search for a package by name from the command line:

```
mamba search numpy
mamba search r-ggplot2
mamba search samtools
```

> *This is why:* Package names on conda-forge follow predictable conventions: R packages use the prefix `r-` followed by the lowercase CRAN name (`r-mass`, `r-glmnet`, `r-parsnip`); Bioconductor packages use `bioconductor-` (`bioconductor-deseq2`); Python and CLI packages usually match their common name. You can often predict the conda name before searching.

> *Hint:* To restrict the search to a specific channel, add `-c conda-forge` or `-c bioconda`. To search across both: `mamba search -c conda-forge -c bioconda samtools`.

(2.) If you prefer browsing, use the channel web interfaces.

> ***Resource:*** Use the conda-forge package index (https://conda-forge.org/packages/) or the bioconda package index (https://bioconda.github.io/conda-package_index.html) which focuses on bioinformatics tools and Bioconductor packages. The Pixi package index (https://prefix.dev/) offers a unified search engine across conda-forge, bioconda, and other channels in one interface.

›› *Sharing and managing environment files*

(1.) When working with Git to track changes SOP 0101, commit both `environment.yml` and the lockfile to the project repository alongside the source code. When you change one, regenerate and commit the other in the same commit.

> ***Critical:*** Updating `environment.yml` without regenerating the lockfile leaves the two artifacts inconsistent: the next collaborator who installs from the lockfile will not see your changes; the next who installs from `environment.yml` will get a different version set than you tested.

(2.) Pin top-level packages with major and minor versions in `environment.yml`; leave patches floating.

> ***Hint:*** Specify `numpy=1.26` rather than `numpy` alone (too loose) or a fully-specified build hash (too tight). The lockfile captures the patch version separately for full reproducibility.

(3.) To recreate an environment exactly from a lockfile on a new machine, install from the lockfile rather than the environment file.

```
conda-lock install --name my-project conda-lock.yml
```

(4.) When working with an external repository, apply the same discipline. If forking to modify or extend: adopt the upstream `environment.yml` if one exists (create one if not), then immediately regenerate a lockfile in your fork. If cloning without modification: create the environment from the upstream file as-is.

```
mamba env create -f environment.yml
mamba activate my-project
```

Updating and pruning

(1.) Update mamba periodically. Update only in `base`, not in project environments.

```
mamba update -n base mamba
```

> ***Hint:*** Project environments should change only when the project's `environment.yml` and lockfile change. Updating mamba in a project environment risks a silent dependency shift.

(2.) List existing environments and remove ones for projects no longer worked on.

```
mamba env list
mamba env remove -n old-project
```

> ***Hint:*** The environment file in the project repository remains; the environment can be recreated later if work resumes.

(3.) Reclaim disk space from the package cache.

```
mamba clean --all
```

> ***This is why:*** This removes downloaded packages no longer used by any environment plus index caches. Existing environments are unaffected because they reference cached files via hardlinks.

Troubleshooting

Per-project environment workflow

In Step 1:

- An environment with both conda and pip installs becomes inconsistent or breaks on an apparently unrelated update.
  - Install conda packages first; install pip packages after, listed under a `pip:` section in `environment.yml`. Never use `pip install` for a package also available on conda-forge.

In Step 2:

- The solver takes minutes or appears stuck after `mamba install` or `mamba env create`.
  - Confirm you are running mamba and not conda. Confirm channel priority is set to `strict`. For very large environments, split heavy stacks (full Bioconductor) into a separate environment from your lighter analysis stack.
- Running `mamba` in a fresh terminal returns "command not found".
  - The shell init step did not run. Execute `~/miniforge3/bin/mamba init` for your shell and restart the terminal. Confirm the appropriate dotfile (`~/.zshrc`, `~/.bashrc`) was modified.
- Environment activation fails in non-interactive shells. A cron job, systemd unit, or build script cannot find `mamba activate`.
  - Source the conda hook explicitly at the top of the script: `source ~/miniforge3/etc/profile.d/conda.sh`, then `mamba activate my-project`.
- Lockfile generation fails with a solver error.
  - The dependency graph is unsatisfiable. Loosen version pins in `environment.yml` or remove the most recently added package and try again to identify the conflict.
- A needed CRAN or Bioconductor package has no `r-` or `bioconductor-` equivalent.
  - Install it from inside R using `install.packages()` or `BiocManager::install()`. Note in the project README that this package is not captured by `environment.yml`. If the package is critical, consider switching the project to native R with `renv`.
- An R script calls `install.packages()` or `BiocManager::install()` directly, either in your own code or in an external script you are running.
  - Calling `install.packages()` inside a script is widely considered poor practice in the R community since it silently modifies the recipient's library without consent (see Bryan, "Project-oriented workflow", tidyverse.org, 2017). Scripts should only call `library()` to declare what they need, not installing it.
  - For packages in your own scripts: remove the call and add the package to `environment.yml` instead; re-export and re-lock.
  - For external scripts you cannot modify: the call will install into R's user library outside the conda environment and will not travel with the project. Note the dependency in the README and add a conda equivalent to `environment.yml` where one exists.

Sharing and managing environment files

In Step 1:

- An environment recreated on an Apple Silicon Mac fails because some packages have no `osx-arm64` build.
  - Force the architecture for the affected environment: `CONDA_SUBDIR=osx-64; mamba env create -n my-project -f environment.yml`. The environment will run under Rosetta 2; use as a fallback only.

Resources

Per-project environment workflow

```yaml
name: my-project
channels:
  - conda-forge
  - bioconda
  - defaults
dependencies:
  - python=3.12
  - numpy=1.26
  - pandas=2.2
  - scipy
  - scikit-learn
  - matplotlib
  - seaborn
  - jupyterlab
  - r-base=4.3
  - r-essentials
  - r-tidyverse
  - r-irkernel
  - bioconductor-deseq2
  - samtools
  - bwa
```

In Step 1: A default environment file. Adjust dependencies based on your needs.

Purpose	Language	Package	Channel	Notes
Runtime	Python	`python=3.12`	conda-forge	Pin minor version
Runtime	R	`r-base=4.3`	conda-forge	Pin minor version
Runtime	R	`r-essentials`	conda-forge	knitr, rmarkdown, and common R extensions
Notebooks	Python	`jupyterlab`	conda-forge	Interactive notebook interface
Notebooks	R	`r-irkernel`	conda-forge	R kernel for Jupyter; run `R -e "IRkernel::installspec()"` after install
Data analysis	Python	`numpy=1.26`	conda-forge	Arrays and linear algebra; pin minor version
Data analysis	Python	`pandas=2.2`	conda-forge	Tabular data (DataFrames); pin minor version
Data analysis	Python	`scipy`	conda-forge	Statistics, signal processing, optimization
Data analysis	Python	`scikit-learn`	conda-forge	Machine learning models and preprocessing
Data analysis	R	`r-tidyverse`	conda-forge	ggplot2, dplyr, tidyr, purrr, readr, etc.
Visualization	Python	`matplotlib`	conda-forge	General-purpose plotting
Visualization	Python	`seaborn`	conda-forge	Statistical visualization built on matplotlib
Differential expression	R	`bioconductor-deseq2`	bioconda	RNA-seq count data
Differential expression	R	`bioconductor-edger`	bioconda	RNA-seq count data
Sequence alignment	—	`bwa`	bioconda	Short-read alignment (BWA-MEM)
Sequence processing	—	`samtools`	bioconda	SAM/BAM file manipulation and statistics
Sequence processing	—	`bcftools`	bioconda	VCF/BCF variant calling and processing
Quality control	—	`multiqc`	bioconda	Aggregated QC reports across pipeline tools

In Step 1: Common dependencies in scientific computing.

Change log

2026-05-10 Benjamin C. Buchmuller Initial commit.