Practical reproducibility in bioinformatics

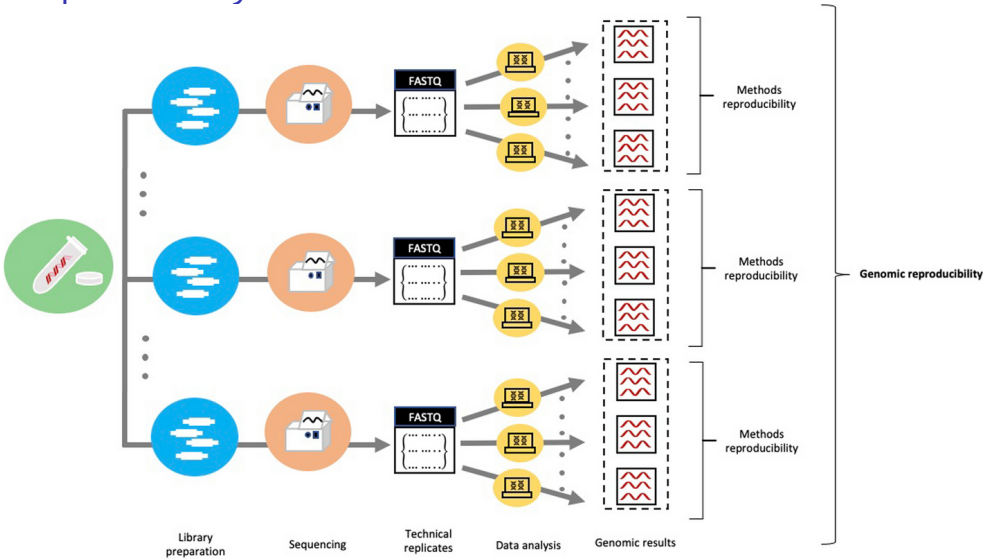
Guillaume Louvel

CAGT

December 13th, 2024

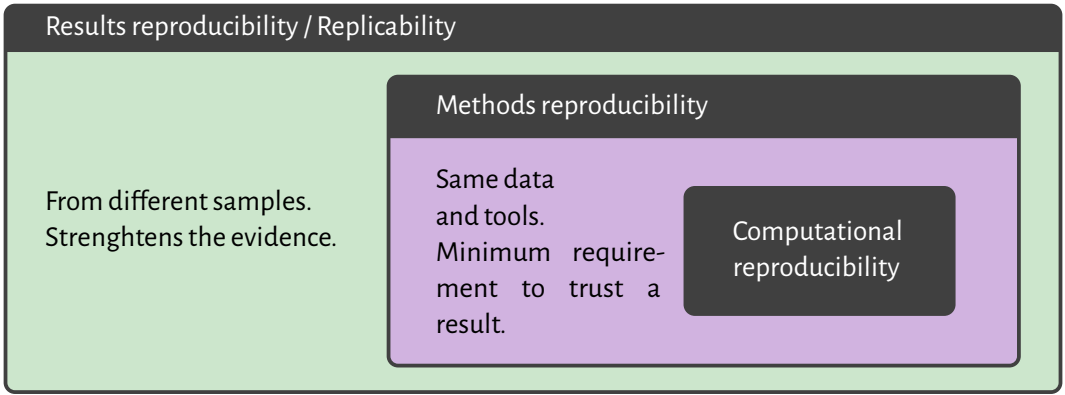
Intro

What is reproducibility?



From Baykal et al. 2024 *"Genomic reproducibility in the bioinformatics era"*

Replicability and Reproducibility



Also see Goodman et al. 2016 "*What does research reproducibility mean?*"

Why?

- Morally:

Why?

- Morally:
  - Validation & trust

Why?

- Morally:
  - Validation & trust
- Scientifically:

Why?

- Morally:
  - Validation & trust
- Scientifically:
  - Incremental progress: lower redundancy of protocols, time-saving, easier generalization to new data.

Why?

- Morally:
  - Validation & trust
- Scientifically:
  - Incremental progress: lower redundancy of protocols, time-saving, easier generalization to new data.
- Personally:

Why?

- Morally:
  - Validation & trust
- Scientifically:
  - Incremental progress: lower redundancy of protocols, time-saving, easier generalization to new data.
- Personally:
  - Quality and speed of the reviewing of your article

Why?

- Morally:
  - Validation & trust
- Scientifically:
  - Incremental progress: lower redundancy of protocols, time-saving, easier generalization to new data.
- Personally:
  - Quality and speed of the reviewing of your article
  - Increased citations

Why?

- Morally:
  - Validation & trust
- Scientifically:
  - Incremental progress: lower redundancy of protocols, time-saving, easier generalization to new data.
- Personally:
  - Quality and speed of the reviewing of your article
  - Increased citations
  - time-saving on the long run

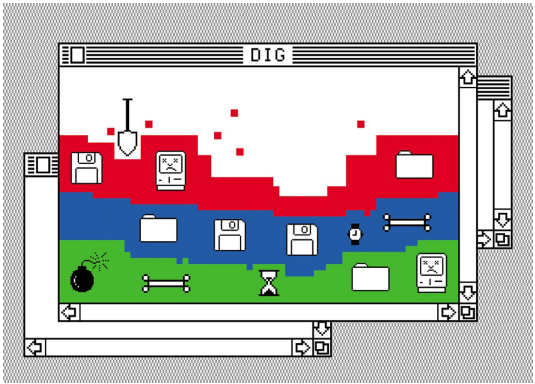
Real world stats

- Ioannidis et al. 2009, *Repeatability of published microarray gene expression analyses*: 2 of 18 articles could be reproduced (**11%**).
- Samuel et al. 2022, *Computational reproducibility of Jupyter notebooks from biomedical applications*: **5.9%** of 4169 notebooks could be reproduced.
- Trisovic et al. 2022, *A large-scale study on research code quality and execution*: **26%** of R scripts from Harvard Dataverse completed without errors.

Challenge to scientists: does your ten-year-old code still run?

Missing documentation and obsolete environments force participants in the Ten Years Reproducibility Challenge to get creative.

By Jeffrey M. Perkel



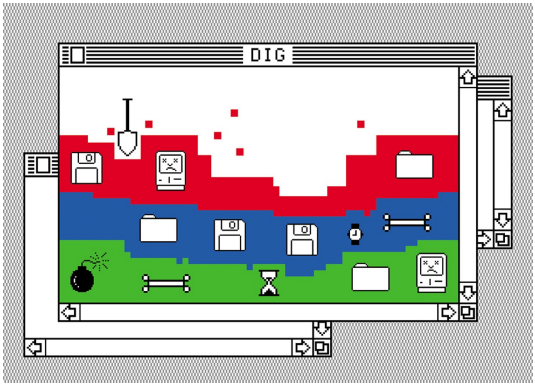
Link: doi.org/10.1038/d41586-020-02462-7

TECHNOLOGY FEATURE | 24 August 2020

Challenge to scientists: does your ten-year-old code still run?

Missing documentation and obsolete environments force participants in the Ten Years Reproducibility Challenge to get creative.

By Jeffrey M. Perkel



Link: doi.org/10.1038/d41586-020-02462-7

Reproducibility targets

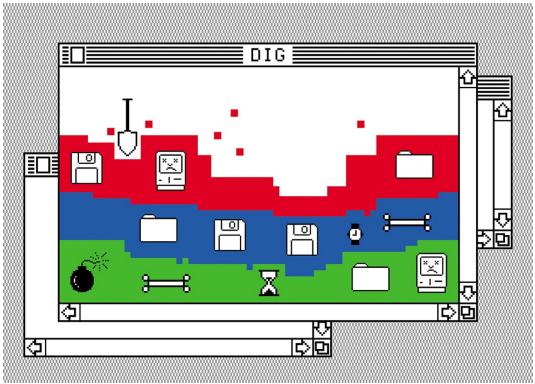
- reproducible by yourself
- reproducible by someone else from the lab
- reproducible by someone else from the field

TECHNOLOGY FEATURE | 24 August 2020

Challenge to scientists: does your ten-year-old code still run?

Missing documentation and obsolete environments force participants in the Ten Years Reproducibility Challenge to get creative.

By Jeffrey M. Perkel



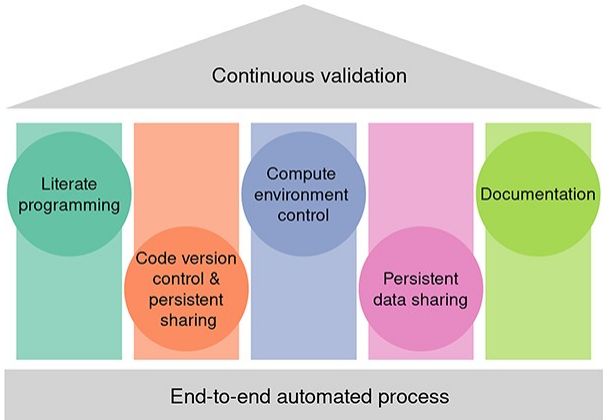
Link: doi.org/10.1038/d41586-020-02462-7

Reproducibility targets

- reproducible by yourself
- reproducible by someone else from the lab
- reproducible by someone else from the field

- reproducible in six months
- reproducible in ten years

Ziemann et al. 2023 "*The five pillars of computational reproducibility*"



Outline

- Data and code accessibility
- Project documentation
  - data descriptions
  - methods high-level summary
- Computational environment
- Recording versions
- Removing manual processing (workflows)
- Coding practices: project structure, continuous validation
- Saving random seeds

Data and code accessibility

Findable, Accessible, Interoperable, Reusable data (FAIR)

nature › scientific data › comment › article

Comment | Open access | Published: 15 March 2016

The FAIR Guiding Principles for scientific data management and stewardship

Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, ... Barend Mons + Show authors

Scientific Data **3**, Article number: 160018 (2016) | Cite this article

Findable, Accessible, Interoperable, Reusable (FAIR)

Findable

- **Metadata**
- Unique identifiers
- Indexing

FAIR principles: www.go-fair.org/fair-principles

Findable, Accessible, Interoperable, Reusable (FAIR)

Findable

- **Metadata**
- Unique identifiers
- Indexing

Accessible

- Through *standard* communication protocols
- Metadata must persist even if the data is missing

FAIR principles: www.go-fair.org/fair-principles

Findable, Accessible, Interoperable, Reusable (FAIR)

Findable

- **Metadata**
- Unique identifiers
- Indexing

Accessible

- Through *standard* communication protocols
- Metadata must persist even if the data is missing

Interoperable

- standard data formats
- standard metadata formats

FAIR principles: www.go-fair.org/fair-principles

Findable, Accessible, Interoperable, Reusable (FAIR)

Findable

- **Metadata**
- Unique identifiers
- Indexing

Interoperable

- standard data formats
- standard metadata formats

Accessible

- Through *standard* communication protocols
- Metadata must persist even if the data is missing

Reusable

- Sufficient description
- usage licenses

FAIR principles: www.go-fair.org/fair-principles

Recommendations for data sharing

(i) Deposit data to a specialized repository if possible, otherwise, a general-purpose repository.

(ii) Avoid commodity cloud storage as these are impermanent and susceptible to link decay [128].

(iii) Avoid large Supplementary Data files accompanying journal articles as these are less findable and accessible [129].

(iv) Preferably archive and share raw data and use existing standards for the discipline.

(v) Use file formats that are machine-readable and compatible with many different types of software. Some examples include comma- and tab-separated values (CSV/TSV) formats, eXtensible Markup Language (XML), JavaScript Object Notation (JSON), Hierarchical Data Format version 5 (HDF5) and Apache Parquet.

(vi) Provide detailed metadata, e.g. sample descriptions that match the article; describe the columns in tabular data (i.e. data dictionary).

From Ziemann et al. 2023

Platforms for long-term storage

1. Search for a *specialized* platform in re3data.org
2. Use a general purpose platform:
   - Zenodo
   - Dryad
   - FigShare
   - SoftwareHeritage

From Ziemann et al. 2023

Project documentation and structure

Project documentation

1. Describe data

Project documentation

1. Describe data
2. Keep a notebook.

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:
   - Materials Design Analysis and Reporting (MDAR) checklist (Mellor et al. 2021)

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:
   - Materials Design Analysis and Reporting (MDAR) checklist (Mellor et al. 2021)
   - Minimum Information about a Bioinformatics Investigation (MIABi) guidelines (Tan et al. 2011)

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:
   - Materials Design Analysis and Reporting (MDAR) checklist (Mellor et al. 2021)
   - Minimum Information about a Bioinformatics Investigation (MIABi) guidelines (Tan et al. 2011)
5. Share the exhaustive data and code to: `protocols.io`, `Zenodo.org`, etc.

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:
   - Materials Design Analysis and Reporting (MDAR) checklist (Mellor et al. 2021)
   - Minimum Information about a Bioinformatics Investigation (MIABi) guidelines (Tan et al. 2011)
5. Share the exhaustive data and code to: `protocols.io`, `Zenodo.org`, etc.
6. Cross-link all published resources: eg.

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:
   - Materials Design Analysis and Reporting (MDAR) checklist (Mellor et al. 2021)
   - Minimum Information about a Bioinformatics Investigation (MIABi) guidelines (Tan et al. 2011)
5. Share the exhaustive data and code to: `protocols.io`, `Zenodo.org`, etc.
6. Cross-link all published resources: eg.
   - Zenodo dataset $\rightarrow$ article

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:
   - Materials Design Analysis and Reporting (MDAR) checklist (Mellor et al. 2021)
   - Minimum Information about a Bioinformatics Investigation (MIABi) guidelines (Tan et al. 2011)
5. Share the exhaustive data and code to: `protocols.io`, `Zenodo.org`, etc.
6. Cross-link all published resources: eg.
   - Zenodo dataset → article
   - Container image → article

Project documentation

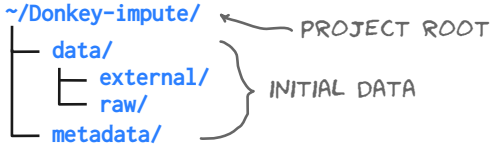
1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:
   - Materials Design Analysis and Reporting (MDAR) checklist (Mellor et al. 2021)
   - Minimum Information about a Bioinformatics Investigation (MIABi) guidelines (Tan et al. 2011)
5. Share the exhaustive data and code to: `protocols.io`, `Zenodo.org`, etc.
6. Cross-link all published resources: eg.
   - Zenodo dataset $\rightarrow$ article
   - Container image $\rightarrow$ article
   - article $\rightarrow$ Zenodo dataset

Project documentation

1. Describe data
2. Keep a notebook.
3. Track versions.
4. Publish a detailed Methods section:
   - Materials Design Analysis and Reporting (MDAR) checklist (Mellor et al. 2021)
   - Minimum Information about a Bioinformatics Investigation (MIABi) guidelines (Tan et al. 2011)
5. Share the exhaustive data and code to: `protocols.io`, `Zenodo.org`, etc.
6. Cross-link all published resources: eg.
   - Zenodo dataset → article
   - Container image → article
   - article → Zenodo dataset
   - article → container image

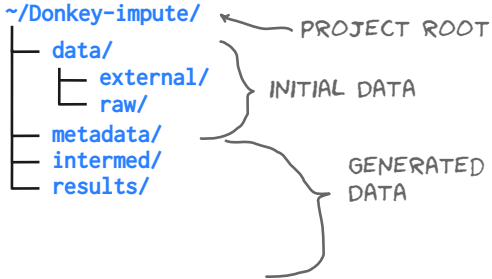
The project structure should be self-contained

Example:



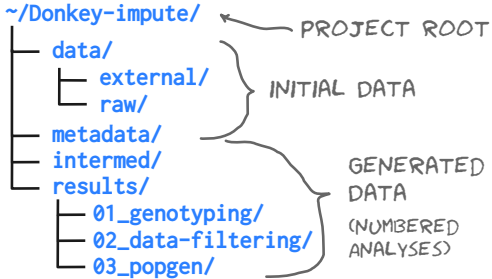
The project structure should be self-contained

Example:



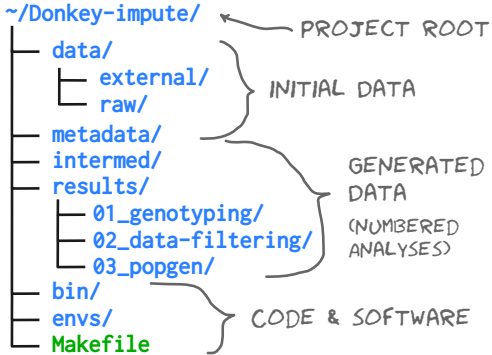
The project structure should be self-contained

Example:



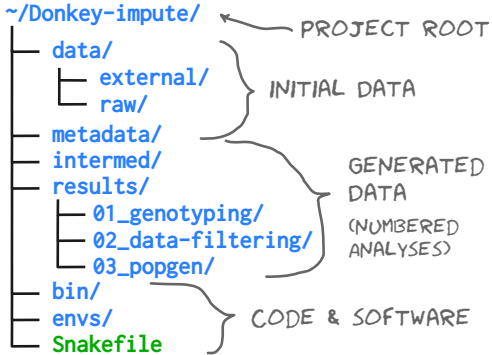
The project structure should be self-contained

Example:



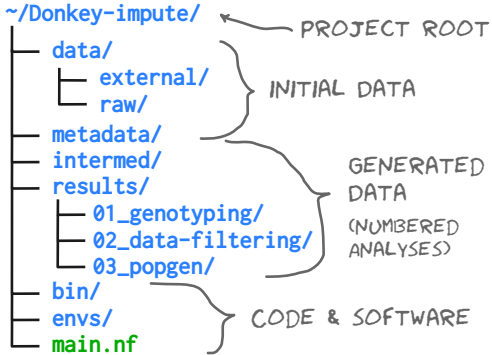
The project structure should be self-contained

Example:



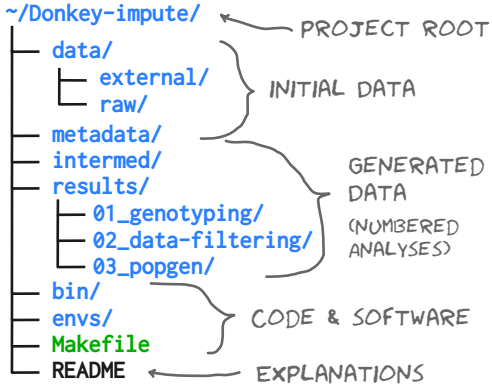
The project structure should be self-contained

Example:



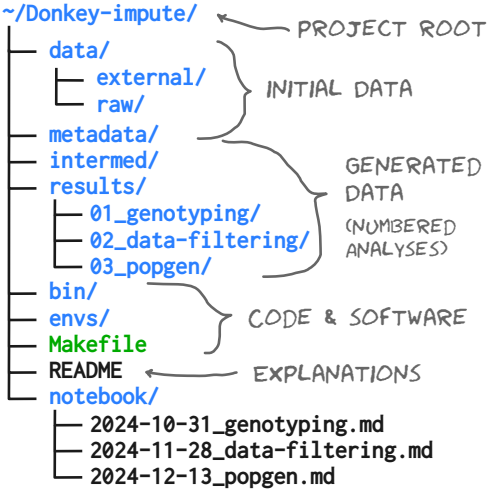
The project structure should be self-contained

Example:



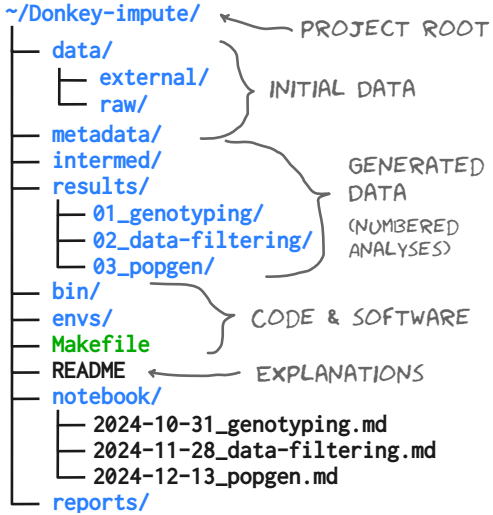
The project structure should be self-contained

Example:



The project structure should be self-contained

Example:



To use existing templates:

Documentation: `cookiecutter.readthedocs.io`


```
pipx install cookiecutter
```

Search and install bioinformatics templates (e.g. from github):

`cookiecutter gh:paleobiotechnology/analysis-project-structure`

`cookiecutter gh:maxplanck-ie/cookiecutter-bioinformatics-project`


```
pipx install cookiecutter
```

Search and install bioinformatics templates (e.g. from github):

`cookiecutter gh:paleobiotechnology/analysis-project-structure`

`cookiecutter gh:maxplanck-ie/cookiecutter-bioinformatics-project`

TODO

Develop our own template?

Recording the software environment

What are software made of?

- executables
- shared libraries

Where are these components stored?

Locations of `bin` and `lib` folders in a typical Linux system

What are software made of?

- executables
- shared libraries

Where are these components stored?

Locations of `bin` and `lib` folders in a typical Linux system

What are software made of?

- executables
- shared libraries

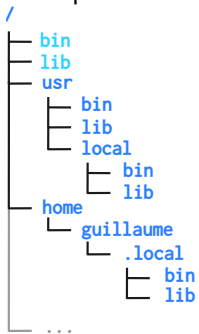
Where are these components stored?

Locations of `bin` and `lib` folders in a typical Linux system

What are software made of?

- executables
- shared libraries

Where are these components stored?

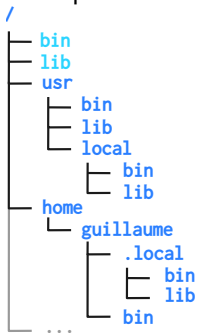


Locations of `bin` and `lib` folders in a typical Linux system

What are software made of?

- executables
- shared libraries

Where are these components stored?

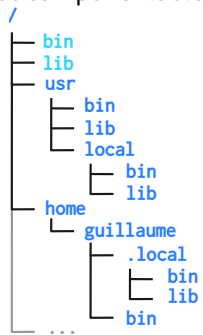


Locations of `bin` and `lib` folders in a typical Linux system

What are software made of?

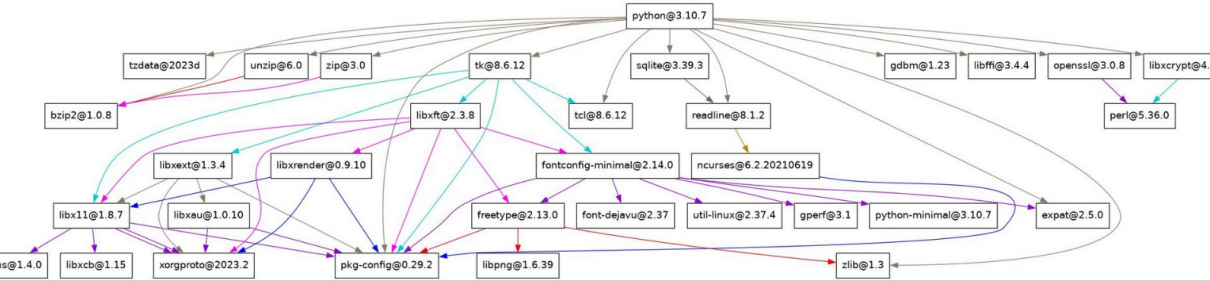
- executables
- shared libraries

Where are these components stored?



Locations of `bin` and `lib` folders in a typical Linux system

Your environment (where to look for programs) is made of several layers.



Python dependency graph, by Simon Tournier. Source: 2024.jres.org/programme#modal-135

Introductory problem: cross-platform code

An example: GNU sed vs BSD sed (includes MacOS)

```
sed -r 's/^.* ([0-9]+) .*$/\1/'
```

Introductory problem: cross-platform code

An example: GNU `sed` vs BSD `sed` (includes MacOS)

```
sed -r 's/^.* ([0-9]+) .*$/\1/'
```

`-r` is a GNU sed option (for extended regex). Use `-E` for portability.

Precisely recording the computational environment

- "shallow": Conda/Mamba, Renv, Nix/Guix
- containers: Docker, Singularity, Podman, lxc.

  Emulate a full operating system, enforcing process and filesystem isolation.

Solving the sed ambiguity

```
micromamba install sed  # In conda-forge, this is GNU sed.
```

Solving the sed ambiguity

```
micromamba install sed  # In conda-forge, this is GNU sed.
```

Similarly:

```
micromamba install gawk  # GNU awk

micromamba install coreutils  # cut, sort, uniq, join, ...
```

How many environments?

The environment should be *project-specific*

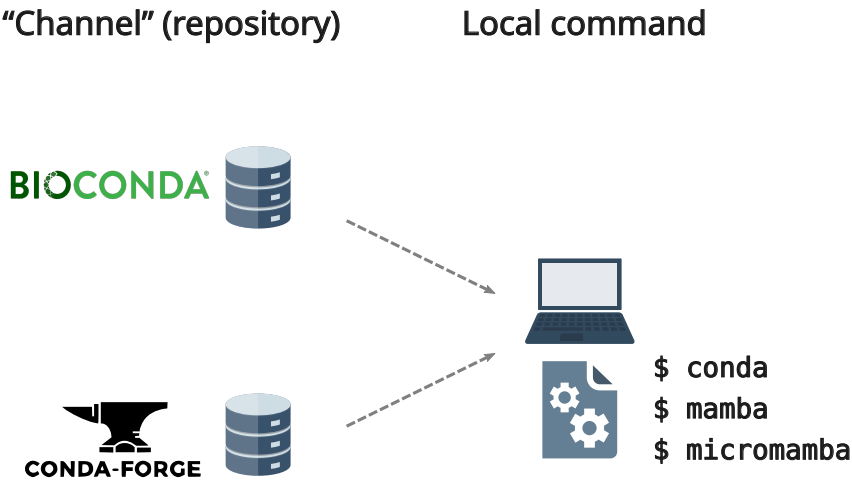
This ensures it won't have changed if you work on something else in between, or if the operating system is updated.

and potentially task-specific

More reproducible because simpler (less dependency relations).
Requires automation to set the environment of each task.

Conda/Mamba

Overview


“Channel” (repository)
Local command
BIOCONDA
CONDA-FORGE
$ conda
$ mamba
$ micromamba

Installing

See intructions from `bioconda.github.io`.

1. Install `conda`/`mamba`/`micromamba`:
   - Download the installer `miniforge`[1]: ships `conda` and `mamba`.
   - Alternatively, install `micromamba` (this modifies your `.bashrc` file).
2. Configure (in `~/.condarc`): *Disable Anaconda™ repositories.*

```
channels:
  - conda-forge
  - bioconda
channel_priority: strict  # No fallback to Anaconda's 'default'
auto_activate_base: false  # Using the 'base' environment is officially discouraged.
```

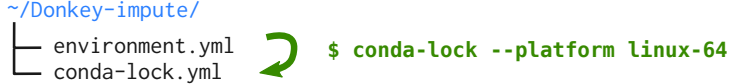
[1] Please also download the corresponding `.sha256` file and verify it.

Setting up your project environments

```
~/Donkey-impute/
└── environment.yml
```

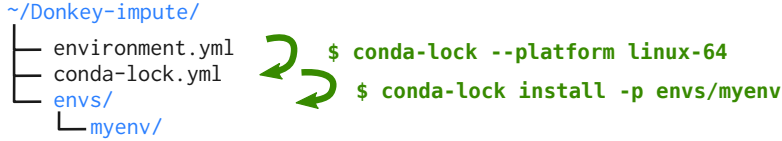
```yaml
# File: environment.yml
# Provide the target requirements (minimal, no version needed)
name: myenv
channels:
  - conda-forge
  - bioconda
dependencies:
  - bcftools
  - samtools
  - pip
  - pip:
    - ↴
      git+https://github.com/popgenmethods/ldpop.git@d786968b346c16e1415c2
```

Setting up your project environments



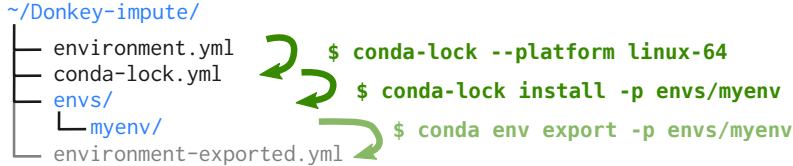

```yaml
# File: environment.yml
# Provide the target requirements (minimal, no version needed)
name: myenv
channels:
  - conda-forge
  - bioconda
dependencies:
  - bcftools
  - samtools
  - pip
  - pip:
    - ↵
      git+https://github.com/popgenmethods/ldpop.git@d786968b346c16e1415c2
```

Setting up your project environments




```yaml
# File: environment.yml
# Provide the target requirements (minimal, no version needed)
name: myenv
channels:
  - conda-forge
  - bioconda
dependencies:
  - bcftools
  - samtools
  - pip
  - pip:
    - ↴
      git+https://github.com/popgenmethods/ldpop.git@d786968b346c16e1415c2
```

Setting up your project environments




```yaml
# File: environment.yml
# Provide the target requirements (minimal, no version needed)
name: myenv
channels:
  - conda-forge
  - bioconda
dependencies:
  - bcftools
  - samtools
  - pip
  - pip:
    - ↴
      git+https://github.com/popgenmethods/ldpop.git@d786968b346c16e1415c2
```

What to do when a software is not in conda?

Provide exact versions (e.g. commit hash) and commands to install them.

or… build a container.

But there are still system dependencies not recorded.

Other environment managers

Complete and reproducible dependency graphs with Guix

- Bit-for-bit environment reproducibility, thanks to:

Complete and reproducible dependency graphs with Guix

- Bit-for-bit environment reproducibility, thanks to:
  - deterministic dependency graph building

Complete and reproducible dependency graphs with Guix

- Bit-for-bit environment reproducibility, thanks to:
  - deterministic dependency graph building
  - recording hashes of the compiled binaries

Complete and reproducible dependency graphs with Guix

- Bit-for-bit environment reproducibility, thanks to:
  - deterministic dependency graph building
  - recording hashes of the compiled binaries
  - “time-machine”: calling a previous guix version when necessary.

Complete and reproducible dependency graphs with Guix

- Bit-for-bit environment reproducibility, thanks to:
  - deterministic dependency graph building
  - recording hashes of the compiled binaries
  - “time-machine”: calling a previous guix version when necessary.
- shell containers: ignore everything installed on your system.

Complete and reproducible dependency graphs with Guix

- Bit-for-bit environment reproducibility, thanks to:
  - deterministic dependency graph building
  - recording hashes of the compiled binaries
  - “time-machine”: calling a previous guix version when necessary.
- shell containers: ignore everything installed on your system.
- “Full-source bootstrap”

Complete and reproducible dependency graphs with Guix

- Bit-for-bit environment reproducibility, thanks to:
  - deterministic dependency graph building
  - recording hashes of the compiled binaries
  - “time-machine”: calling a previous guix version when necessary.
- shell containers: ignore everything installed on your system.
- “Full-source bootstrap”
- export environments for systems that don’t have Guix (e.g. as Docker image)

Complete and reproducible dependency graphs with Guix

- Bit-for-bit environment reproducibility, thanks to:
  - deterministic dependency graph building
  - recording hashes of the compiled binaries
  - “time-machine”: calling a previous guix version when necessary.
- shell containers: ignore everything installed on your system.
- “Full-source bootstrap”
- export environments for systems that don’t have Guix (e.g. as Docker image)
- fallback to Software Heritage

Guix example

Recording your environment, and replicating it exactly somewhere else requires two files:

`manifest.scm` the packages from the environment
`channels.scm` Guix current state (version, repositories)

Guix example

Recording your environment, and replicating it exactly somewhere else requires two files:

`manifest.scm` the packages from the environment
`channels.scm` Guix current state (version, repositories)

```
host1:~$ guix package -p $GUIX_PROFILE --export-manifest > manifest.scm
host1:~$ guix describe --format=channels -p $GUIX_PROFILE > channels.scm
```

```
host2:~$ guix time-machine --channels=channels.scm -- package -p ↵
    $GUIX_PROFILE -m manifest.scm
```

See `infosec.press/csantosb/guix-crash-course`

Containers

Containers

- Docker
- Podman
- Singularity/Apptainer
- lxc

Repositories of existing container images:

- DockerHub
- BioContainers

Tracking code versions

Why?

- Code changes
- Results depend on the exact code being used
- You might want to revert back to a previous version

A very frequent backup as a minimal solution?

```
# Incremental backup with rsync (optimizes space)
mkdir backup2024_Dec8
rsync -ra --link-dest=../backup2024_Dec7 projectdir/ backup2024_Dec8
```

Limitations

- Navigating through the history manually is tricky
- Taking a snapshot of a subset of files is tricky
- Using multiple computers is tricky
- You can only have a linear history
- You cannot easily collaborate (e.g. merge changes from someone else)

First: Git $\neq$ GitHub

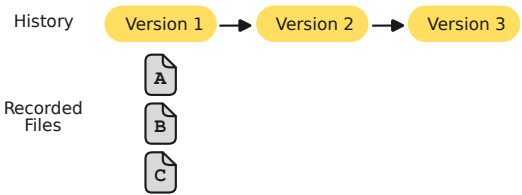
Git is a *distributed version control system*, created by Linus Torvalds around 2005 for the Linux kernel development.

```
$ man git
```

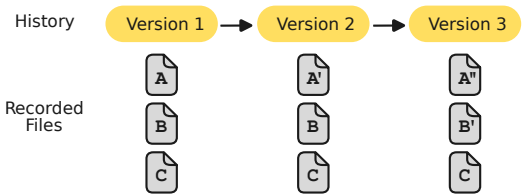
See Git book: 1.3. Getting started - What is Git?

Git in a nutshell

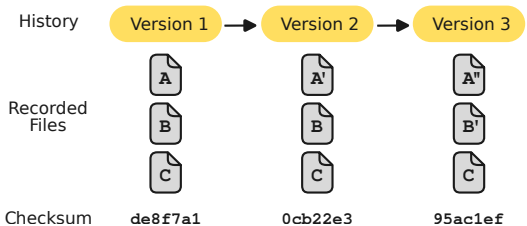
Git in a nutshell



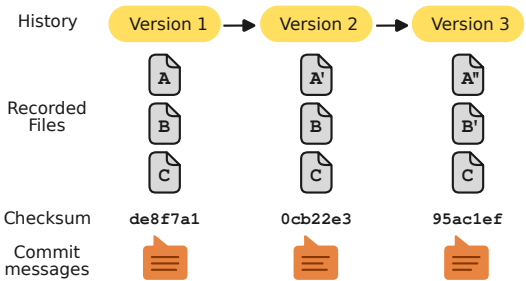
Git in a nutshell



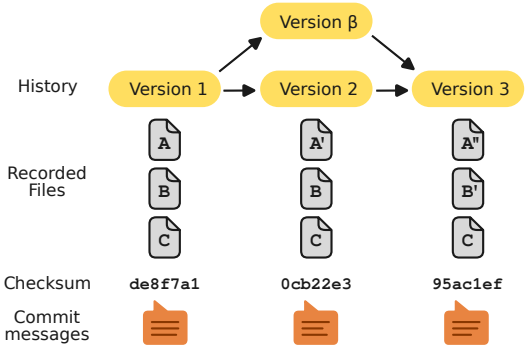
Git in a nutshell



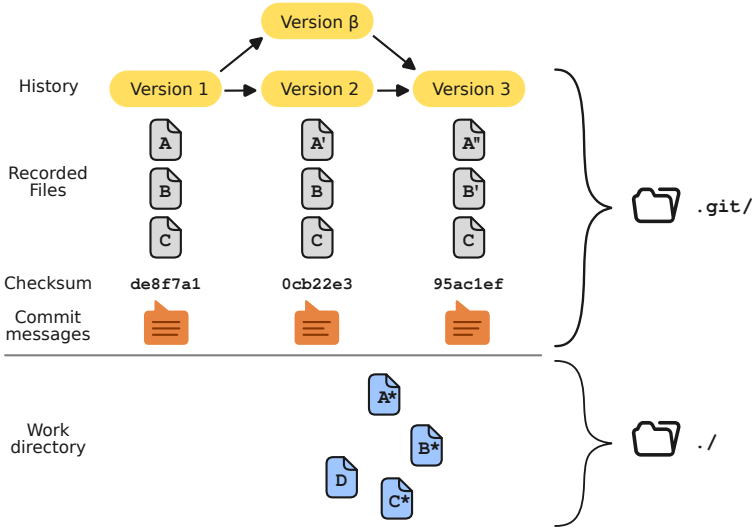
Git in a nutshell



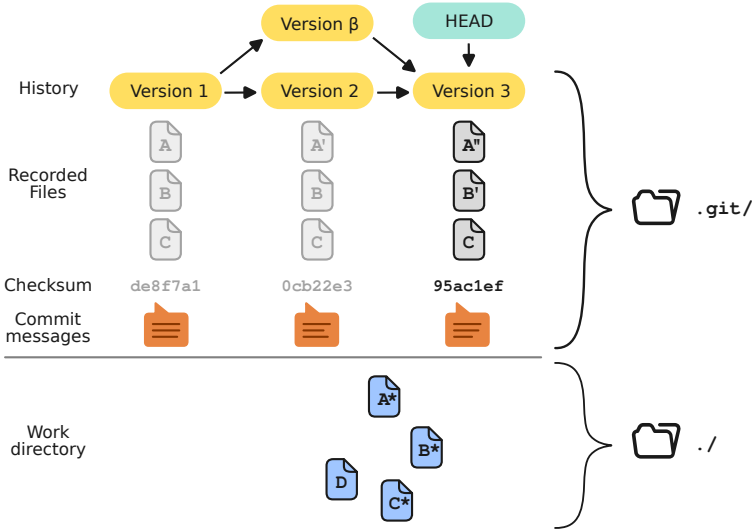
Git in a nutshell



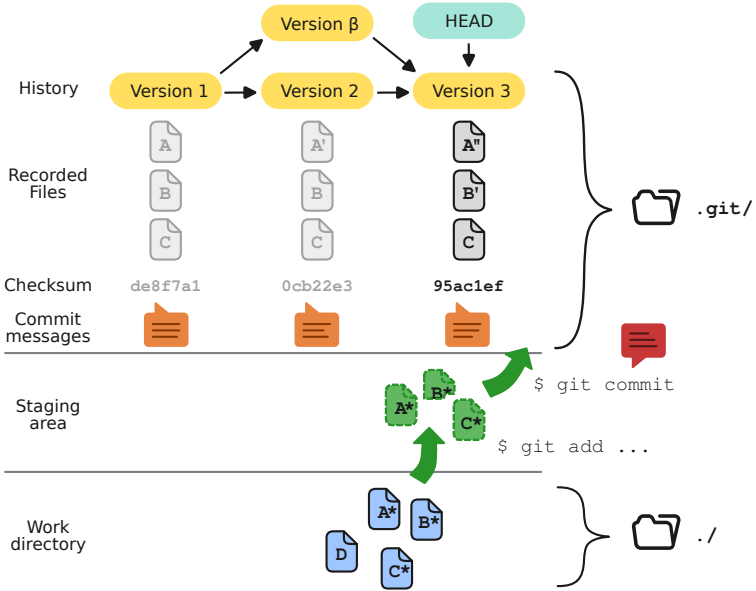
Git in a nutshell



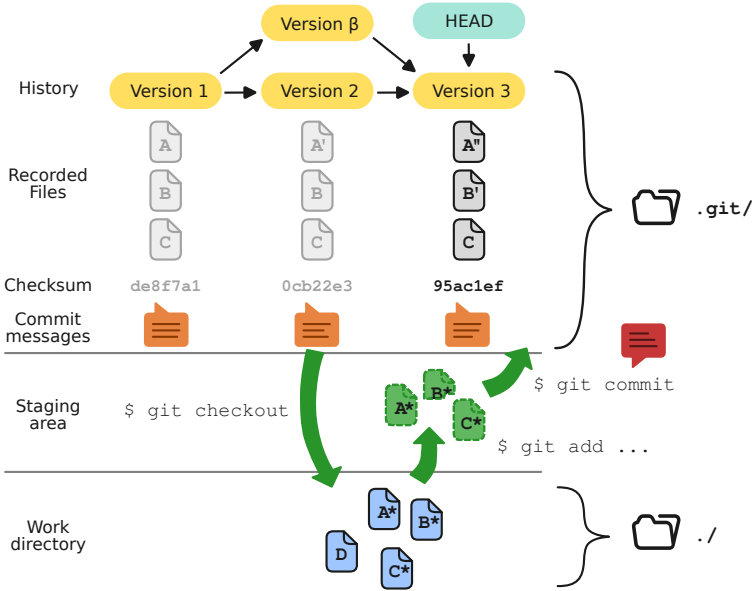
Git in a nutshell



Git in a nutshell

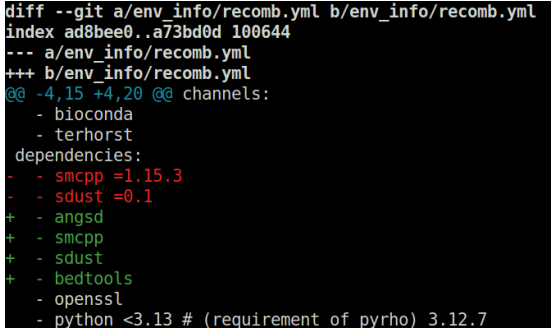


Git in a nutshell



Put colors in your Git

```
git config --global color.ui 'auto'
# default since v1.8.4
```



Colored `git diff` output

Also select an editor with syntax highlighting (here `Vim`):

```
git config --global core.editor 'vim'
```

Minimalist command usage (initialization)

Once for each project:

```
# Go to the root of your project directory
cd ~/Donkey-impute

# Initialize this directory as a Git repository
git init --initial-branch main

# Configure a remote repository (optional).
# Create it before on e.g. Gitlab.com
git remote add origin https://gitlab.com/guillaumelouvel/Donkey-impute.git
```

Minimalist command usage (initialization)

Once for each project:

```
# Go to the root of your project directory
cd ~/Donkey-impute

# Initialize this directory as a Git repository
git init --initial-branch main

# Configure a remote repository (optional).
# Create it before on e.g. Gitlab.com
git remote add origin https://gitlab.com/guillaumelouvel/Donkey-impute.git
```

Summary of the state of your repository:

```
git status
# Shows untracked files, staged files, modified files, and more.
```

Minimalist command usage

Every time you want to record a version (aka commit):

```
# Optional: visualize your
# modifications
git diff

# Stage modifications
git add fileA.txt

# Commit:
git commit -m 'An informative message'
```

Minimalist command usage

Every time you want to record a version (aka commit):

```
# Optional: visualize your
# modifications
git diff

# Stage modifications
git add fileA.txt

# Commit:
git commit -m 'An informative message'
```

Shortcuts:

```
# Stage a file and commit
# simultaneously:
git commit \
    -m 'An informative message' \
    fileA.txt
```

```
# Stage all tracked files and commit:
git commit -am 'An informative message'
```

Synchronizing with the remote repository:

```
# Send your changes
# The first time, configure 'main' to always follow the 'origin' remote:
git push --set-upstream origin main

# Next times:
git push

# Retrieve remote updates
git pull
```

View your history as commit messages:

```
git log
```

Synchronizing with the remote repository:

```
# Send your changes
# The first time, configure 'main' to always follow the 'origin' remote:
git push --set-upstream origin main

# Next times:
git push

# Retrieve remote updates
git pull
```

View your history as commit messages:

```
git log
```

View your history as a graph:

```
git log --oneline --graph
```

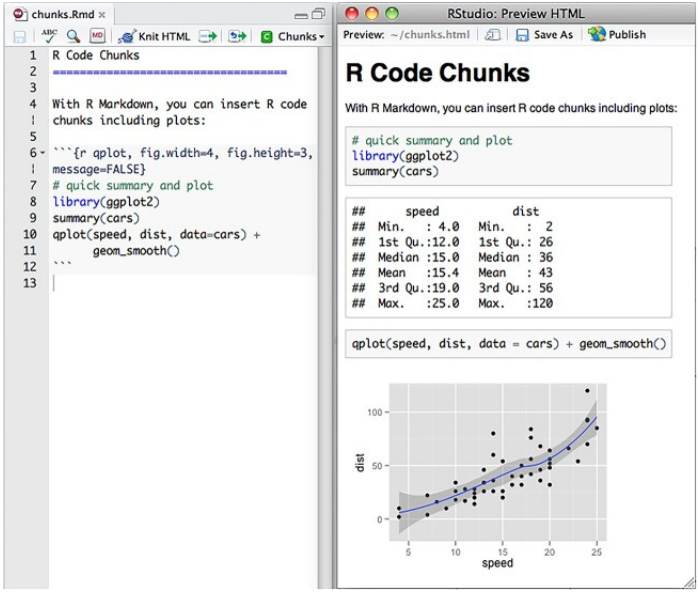
Social git platforms out there

`github.com`

`gitlab.com`

`codeberg.org`

Literate programming (computational notebooks)



Rmarkdown notebook. From Ziemann et al. 2023

Pros and cons

- *Provenance* of the results
- Embedded documentation in versatile formats.
- Matches outputs with code

Caveats:

- Although a notebook can be run interactively, partially, in varying orders, it's best to always run them entirely for the final publication.

Systems

- Quarto
- Jupyter (also see MyST)

Continuous validation with automatic testing

- R `testthat`
- Python `pytest`

More generally, code defensively: make your program fail whenever something looks weird.

Using Git, tests can be automatically executed for each commit/push.

Random seeds

Computer random numbers are pseudorandom

Example with genomics software:

plink:

1. You did not give a seed: save it from `plink2.log`!
2. `plink2 --seed 123456789 ...`

admixture:

The default seed is 43.

Seeds within Python or R code

Your own R code

```
set.seed(123)
```

Also see `help(Random)`.

Your own python code

If using only the `random` module:

```
random.seed(123456789)
```

Seeds when using NumPy

```
from numpy.random import rng_default, SeedSequence

# ask the user for a seed. Set it to None if not given.
ss = SeedSequence(user_seed)
# show it
print('Seed:', ss.entropy)

rng = rng_default(ss)
```

Seeds when using NumPy

```
from numpy.random import rng_default, SeedSequence

# ask the user for a seed. Set it to None if not given.
ss = SeedSequence(user_seed)
# show it
print('Seed:', ss.entropy)

rng = rng_default(ss)
```

If doing parallel computations:

```
child_seedseqs = ss.spawn(12)
generators = [rng_default(s) for s in child_seedseqs]

# Or alternatively, from the generator instead of the seed:

generators = rng.spawn(12)
```

Awk???

Generate a random float in $[0, 1[$ with `awk`:

```
rand()
```

A random integer in $[0, n - 1]$:

```
int(rand()*n)
```

Pseudohaploidize a genotyped dataset 

```
awk '{
    while( sub(/1/, int(rand()*2)+3) ){};
    gsub(/3/, "0");
    gsub(/4/, "2");
    print
    }' input.geno > pseudohaplo.geno
```

Awk???

Generate a random float in $[0, 1[$ with `awk`:

```
rand()
```

A random integer in $[0, n-1]$:

```
int(rand()*n)
```

Pseudohaploidize a genotyped dataset 🙊

```
awk '{
    while( sub(/1/, int(rand()*2)+3) ){};
    gsub(/3/, "0");
    gsub(/4/, "2");
    print
    }' input.geno > pseudohaplo.geno
```

Set the seed:

```
srand(123)  # returns the previously used seed.
```

Awk???

CAUTION: In most `awk` implementations, including `gawk`, `rand()` starts generating numbers from the same starting number, or *seed*, each time you run `awk`.[48] Thus, a program generates the same results each time you run it. The numbers are random within one `awk` run but

Awk???

CAUTION: In most awk implementations, including gawk, `rand()` starts generating numbers from the same starting number, or *seed*, each time you run awk.[48] Thus, a program generates the same results each time you run it. The numbers are random within one awk run but

(48)

mawk uses a different seed each time.

Awk???

CAUTION: In most awk implementations, including gawk, `rand()` starts generating numbers from the same starting number, or *seed*, each time you run awk.[48] Thus, a program generates the same results each time you run it. The numbers are random within one awk run but

(48)

mawk uses a different seed each time.

From version 4.1.4, gawk uses the Bays-Durham shuffle buffer algorithm [...]

To summarize…

Summary

- Data and code accessibility: **FAIR principles**
- Project documentation: **README, notebooks, metadata**
- Computational environment: **conda-lock, Guix**
- Recording versions: **Git**
- Coding practices: **project structure**, continuous validation
- Saving random seeds

For next time

- Removing manual processing (workflows)
  - Snakemake
  - Nextflow
  - CWL
- Containers
- Tracking *data* versions alongside Git: `git-annex`, `DataLad`

References

Baykal, Pelin Icer et al. (2024). “Genomic reproducibility in the bioinformatics era”. In: *Genome Biology* 25.1, p. 213. DOI: 10.1186/s13059-024-03343-2 (cit. on p. 3).

Goodman, Steven N. et al. (2016). “What does research reproducibility mean?” In: *Science Translational Medicine* 8.341, 341ps12–341ps12. DOI: 10.1126/scitranslmed.aaf5027 (cit. on p. 4).

Ziemann, Mark et al. (2023). “The five pillars of computational reproducibility: bioinformatics and beyond”. In: *Briefings in Bioinformatics* 24.6, bbad375. DOI: 10.1093/bib/bbad375 (cit. on pp. 17, 25, 26, 109).