Scientific Data Management with Git and Git-Annex

Arnaud Legrand

Journée GitLab, GT "Données" de la MITI du CNRS
June 2023
Scientific Consensus

No Transparency
No Consensus

Can I see your raw data?
No way!
So how did you get these results?
No idea!
I lost my workings.
How did he do that?
Want to peer review my paper?
Sure as long as I don't have to read it.
How do I know it's true?
You will just have to trust me.
Reproducible Research = Rigor and Transparency

Findable Accessible Interoperable Reusable

MOOC: Reproducible Research: Methodological principles for a transparent science, Inria Learning Lab
• Konrad Hinsen, Christophe Pouzat

Markdown, CSV, GitLab
• Notebooks: Jupyter/Rstudio/Org-Mode
• 3rd Edition: March 2020 – December 2023 (15,000+)
• MOOC “Advanced RR” planned for Nov. 2023

Managing data (FITS/HDF5, Zenodo, SWH git annex)
• Software environment control (docker, singularity, guix)
• Scientific workflow (make, snakemake)

2/20
Reproducible Research = Rigor and Transparency

Good research requires time and resources

Train yourself and your students: RR, statistics, experiments

MOOC Reproducible Research: Methodological principles for a transparent science, Inria Learning Lab

- Konrad Hinsen, Christophe Pouzat
- Markdown, CSV, GitLab
- Notbooks: Jupyter / Rstudio / Org-Mode
- 3rd Edition: March 2020 – December 2023 (15,000+)
Reproducible Research = Rigor and Transparency

Good research requires time and resources

Train yourself and your students: RR, statistics, experiments

MOOC Reproducible Research: Methodological principles for a transparent science, Inria Learning Lab

- Konrad Hinsen, Christophe Pouzat
- Markdown, CSV, GitLab
- Notbooks: Jupyter / Rstudio / Org-Mode
- 3rd Edition: March 2020 – December 2023 (15,000+)

MOOC “Advanced RR” planned for Nov. 2023

- Managing data (FITS/HDF5, Zenodo, SWH git annex)
- Software environment control (docker, singularity, guix)
- Scientific workflow (make, snakemake)
VERSION CONTROL AND LARGE FILES
• Allows to track versions (i.e., to manage a history) in a **distributed** way (MOOC RR1: Introduction to Git without the command line (1/3), (2/3), (3/3))

• Designed by Linus Torvald in 2005 (BitKeeper licensing issues)

• Although many common git workflows are centralized (e.g., through GitHub and GitLab), git is **distributed**

**Main drawback**: git has been designed and optimized for source code, not for large binary files
Option 1  Let’s commit large files anyway

- Files are stored in the ”block chain” of git and cannot be removed
- The directory `.git` becomes large (+ duplication) → git becomes slow for you (`checkout`, `diff`, `push`, ...) and others (`clone`, `pull`)
**POSSIBLE STRATEGIES**

**Option 1** Let’s commit large files anyway
- Files are stored in the “block chain” of git and cannot be removed
- The directory `.git` becomes large (+ duplication) ⇒ git becomes slow for you (`checkout`, `diff`, `push`, …) and others (`clone`, `pull`)

**Option 2** Let’s not commit large binary files and put them in a shared directory instead
- When and who did what, and why?
  - Indicate *when* (and *who*) in external metadata
- Backup? How to make sure files are not altered?
  - Store a checksum (MD5, SHA1, SHA256, …) of your files!
  - Files are lost or corrupted? Recompute and check the signature
**Possible strategies**

**Option 1** Let’s commit large files anyway
- Files are stored in the ”block chain” of git and cannot be removed
- The directory `.git` becomes large (+ duplication) $\leadsto$ git becomes slow for you (checkout, diff, push, ...) and others (clone, pull)

**Option 2** Let’s **not** commit large binary files and put them in a shared directory instead
- When and who did what, and why?
  - Indicate when (and who) in **external** metadata
- Backup? How to make sure files are not altered?
  - Store a checksum (MD5, SHA1, SHA256, ...) of your files!
  - Files are lost or corrupted? Recompute and check the signature

**Option 3** Use a git on steroids
1. A lightweight `git clone`
   - Do not download all large files
   - More than git tricks (`git clone --depth`, `git subtree/submodule`)
1. A lightweight `git clone`
   - Do not download all large files
   - More than git tricks (`git clone --depth`, `git subtree/submodule`)

2. Get large files on demand
1. A lightweight `git clone`
   - Do not download all large files
   - More than git tricks (`git clone --depth`, `git subtree/submodule`)

2. Get large files on demand

3. Garbage collection
   - Allow to delete large files (even in `.git/`)...
   - ... without messing up the history
1. A lightweight *git clone*
   - Do not download all large files
   - More than *git tricks* (\texttt{git clone --depth}, \texttt{git subtree/submodule})

2. Get large files on demand

3. Garbage collection
   - Allow to delete large files (even in \texttt{.git/})...
   - ... without messing up the history

4. Manage different (possibly unreliable) storage media
   - While ensuring data integrity
Proposed extensions for handling large files:

**Git LFS**
- Centralized, supported by GitHub, GitLab, BitBucket
- Easy to use (git lfs track "*.hdf5")
- Get large files on demand: set lfs.fetchexclude
- Double disk space (. and .git/lfs/objects/)
- No file removal without rewriting the whole history
- Data is locked up in an opaque storage

**Git Annex**
- Steeper learning curve but incredibly powerful
- Supported by GitLab (2015-2017)
- Not specifically designed for scientific data management but works well
Proposed extensions for handling large files:

**Git LFS**

- Centralized, supported by GitHub, GitLab, BitBucket
- Easy to use (git lfs track "*.hdf5") but
  - Get large files on demand: set lfs.fetchexclude to *
    git lfs pull --exclude= --include "filename"
  - Double disk space (./ and .git/lfs/objects/)

- Get large files on demand: set lfs.fetchexclude to *
  git lfs pull --exclude= --include "filename"
- Double disk space (./ and .git/lfs/objects/)
Git extensions for large files

Proposed extensions for handling large files:

**Git LFS**
- Centralized, supported by GitHub, GitLab, BitBucket
- Easy to use (git lfs track "*.hdf5") but
  - Get large files on demand: set lfs.fetchexclude to *
    git lfs pull --exclude= --include "filename"
  - Double disk space (./ and .git/lfs/objects/)
- No file removal without rewriting the whole history
- Data is locked up in an opaque storage
Git extensions for large files

Proposed extensions for handling large files:

**Git LFS**
- Centralized, supported by GitHub, GitLab, BitBucket
- Easy to use (`git lfs track "*.hdf5"`) but
  - Get large files on demand: set `lfs.fetchexclude` to `*`  
    ```
    git lfs pull --exclude= --include "filename"
    ```
  - Double disk space (`./` and `.git/lfs/objects/`)
  - No file removal without rewriting the whole history
  - Data is locked up in an opaque storage

**Git Annex** by Joey Hess (Debian, Haskell)
- Steeper learning curve but incredibly powerful
- Supported by GitLab (2015-2017)
- Not specifically designed for scientific data management but works well
Introduction to Git-Annex
The project is populated with symbolic links to the large files which end up in `.git/annex/objects` (git annex add)

• No wasted space with file duplication
• Large files are identified by their content (SHA256 by default)

• git clone will retrieve only symbolic links for annexed files

• ⇛ Get (and check) content with `git annex get`

• Files may be `git annex dropped` (from the annex)
• The project is populated with **symbolic links** to the large files which end up in `.git/annex/objects` (git annex add)
  
  • ⇝ No wasted space with file duplication
  • Large files are identified by their content (SHA256 by default)
The project is populated with **symbolic links** to the large files which end up in `.git/annex/objects` (**git annex add**)

- No wasted space with file duplication
- Large files are identified by their content (SHA256 by default)
The project is populated with **symbolic links** to the large files which end up in `.git/annex/objects` (**git annex add**)

- ~~No wasted space with file duplication~~
- Large files are identified by their content (SHA256 by default)

**git clone** will retrieve only symbolic links for annexed files
The project is populated with **symbolic links** to the large files which end up in `.git/annex/objects` (git annex add)

- → No wasted space with file duplication
- Large files are identified by their content (SHA256 by default)

**git clone** will retrieve only symbolic links for annexed files

- → Get (and check) content with **git annex get**
The project is populated with **symbolic links** to the large files which end up in `.git/annex/objects` (**git annex add**)

- ⇔ No wasted space with file duplication
- Large files are identified by their content (SHA256 by default)

**git clone** will retrieve only symbolic links for annexed files

- ⇔ Get (and check) content with **git annex get**

Files may be **git annex dropped** (from the annex)
• **Special remotes** are ways to access files
  - A USB key, a server through SSH or webdav, a web server, Amazon S3, etc.
  - They have their own structure and do **not** comprise the git history
  - Files may be migrated/duplicated between (special) remotes
  - Information on the remotes is stored in a special **git-annex** branch which will be synchronized between git repositories

Illustration? Wait for it!
git annex sync

master → C → a2 → git-annex

master → C

file1

file2

file1

file2
git-annex

master → C → a2

B → a1

file1

file2

git annex move --to server

file1

file2
git annex sync

file1
file2

file1
file2

very big file

master

a1
a2
a3
a4

file1
file2

file1
file2

git annex
git annex copy --from server

very big file

file1

file2

file1

file2

file1

file2

file1

file2

file1

file2

file1

file2
- Hash (SHA1, SHA256, SHA512, ...) for integrity
- Robust internal naming convention compatible with every file-system
- Minimal number of copies per suffix, directory, ...
- All remotes and special remotes can be verified
  - `git fsck` and `git annex fsck`
  - standard remotes: local verification, transmit the result
  - special remotes: may require to transfer all data to verify
Scientific Data Management With Git-Annex
Situation #1: External data
Data are produced and made available read-only
(directory, web server, hard drive)

What could possibly go wrong?

1. New data
2. Data is moved around
3. Data behind a filename is altered
4. Data silently disappears

Let's assume data had been imported in `git annex`
Situation #1: External data
Data are produced and made available read-only
(directory, web server, hard drive)

What could possibly go wrong?

Let’s assume data had been imported in git annex

1. New data
   Just reimport, duplicates will stored only once!

2. Data is moved around

3. Data behind a filename is altered

4. Data silently disappears
**Situation #1: External data**
Data are produced and made available read-only (directory, web server, hard drive)

What could possibly go wrong?

*Let’s assume data had been imported in* **git annex**

1. **New data**
   *Just reimport, duplicates will stored only once!*

2. **Data is moved around**
   *So what?*

3. **Data behind a filename is altered**
   **git annex** *will warn you right away.*

4. **Data silently disappears**
Situation #1: External data
Data are produced and made available read-only (directory, web server, hard drive)

What could possibly go wrong?

Let’s assume data had been imported in git annex

1. New data
   Just reimport, duplicates will stored only once!

2. Data is moved around
   So what?

3. Data behind a filename is altered
   git annex will warn you right away.

4. Data silently disappears
   Is there a copy in another remote? Otherwise, if you ever get this file back, your old symlink will work.
**GIT-ANNEX** can pull files down from the web and bittorrent.

```bash
cd data/
git annex addurl --preserve-filename --pathdepth=2 \  
  https://www.sidc.be/DATA/uset/Wlight/2014/06/UPH20140601105039.FTS
```

```
addurl https://www.sidc.be/DATA/uset/Wlight/2014/06/UPH20140601105039.FTS  
(to uset/Wlight/2014/06/UPH20140601105039.FTS) ok  
(recording state in git...)
```

This is a *(special)* url remote from which data can only be pulled

- only **git annex get** *(no git annex copy nor git annex move)*
Situation #2: Collaborative data production/analysis

- Members of a team are both data producers and consumers
- Read-Write permissions on a server to share data

What will ultimately happen?

1. No more space on your laptop

2. No more space on the server

3. You inadvertently change the content of a file

4. Is this data reproducible?
Situation #2: Collaborative data production/analysis

- Members of a team are both data producers and consumers
- Read-Write permissions on a server to share data

What will ultimately happen?

1. No more space on your laptop
   Just `git annex drop` or `git annex move --to=my-usb-drive`

2. No more space on the server

3. You inadvertently change the content of a file

4. Is this data reproducible?
Situation #2: Collaborative data production/analysis

- Members of a team are both data producers and consumers
- Read-Write permissions on a server to share data

What will ultimately happen?

1. No more space on your laptop
   
   *Just* `git annex drop` or `git annex move --to=my-usb-drive`

2. No more space on the server
   
   `git annex drop --from=server` checks how many copies are available
   
   *No miracle, if the only copy was on your colleague’s stolen laptop...*

3. You inadvertently change the content of a file

4. Is this data reproducible?
Situation #2: Collaborative data production/analysis

- Members of a team are both data producers and consumers
- Read-Write permissions on a server to share data

What will ultimately happen?

1. No more space on your laptop
   - Just `git annex drop` or `git annex move --to=my-usb-drive`

2. No more space on the server
   - `git annex drop --from=server` checks how many copies are available
   - *No miracle, if the only copy was on your colleague’s stolen laptop...*

3. You inadvertently change the content of a file
   - *Permission denied. You should `git annex unlock` it first*

4. Is this data reproducible?
Situation #2: Collaborative data production/analysis

- Members of a team are both data producers and consumers
- Read-Write permissions on a server to share data

What will ultimately happen?

1. No more space on your laptop
   
   *Just* `git annex drop` or `git annex move --to=my-usb-drive`

2. No more space on the server
   
   `git annex drop --from=server` checks how many copies are available
   
   *No miracle, if the only copy was on your colleague’s stolen laptop…*

3. You inadvertently change the content of a file
   
   *Permission denied. You should* `git annex unlock` *it first*

4. Is this data reproducible?
   
   *Just* `rm`, `rerun`, `git annex add`, and `git status`
Set up a shared data store

**git-annex** can store files in Amazon S3, Glacier, WebDAV, or on a rsync server through ssh:

```
1  git annex initremote g5k-rsync type=rsync \
2    rsyncurl=grenoble.g5k:/home/alegrand/git-annex-rsync/
3  git annex describe g5k-rsync "Rsync server on Grid5000"
```
Setting up a shared data store

**`git-annex`** can store files in Amazon S3, Glacier, WebDAV, or on a rsync server through ssh:

```bash
1  git annex initremote g5k-rsync type=rsync \
2       rsyncurl=grenoble.g5k:/home/alegrand/git-annex-rsync/
3  git annex describe g5k-rsync "Rsync server on Grid5000"
```

This is a **special remote**, i.e.,:

- the file hierarchy is not on the server
  - files are stored with the annex structure (SHA256 names)
- the git history is not on the server
  - only the annexed files

Information on this remote (in the `git-annex` branch) will need to be regularly synchronized between team members:

- `git annex sync --only-annex to GitLab or GitHub`

Bonus: Files stored on special remotes can easily be encrypted!
Setting up a shared data store

`git-annex` can store files in Amazon S3, Glacier, WebDAV, or on a rsync server through ssh:

```
git annex initremote g5k-rsync type=rsync \
  rsyncurl=grenoble.g5k:/home/alegrand/git-annex-rsync/
git annex describe g5k-rsync "Rsync server on Grid5000"
```

This is a special remote, i.e.,:

- the file hierarchy is not on the server
  - files are stored with the annex structure (SHA256 names)
- the git history is not on the server
  - only the annexed files

Information on this remote (in the `git-annex` branch) will need to be regularly synchronized between team members

- `git annex sync --only-annex` to GitLab or GitHub

**Bonus:** Files stored on special remotes can easily be encrypted!
Situation #3: Publication to the community

- You want to publish part of your data for a publication
- Others should not have to know nor to use ‘git-annex’

Many possible options

1. Make your git repository and your data server public

   *Wait, making the data server public? How?*
Situation #3: Publication to the community

- You want to publish part of your data for a publication
- Others should not have to know nor to use ‘git-annex’

Many possible options

1. Make your git repository and your data server public
   *Wait, making the data server public? How?*

2. Clean up in a specific branch and publish its head
   *Just git rm before git annex exporting*
   *Large. History remains hidden*

3. Same as above but publish the content of a few files
   *git annex unannex file; git add file*
   *then clone with a --single-branch --depth=1*
   *History is hidden. SHA256 are visible, anyone can check!*
Situation #3: Publication to the community

- You want to publish part of your data for a publication
- Others should not have to know nor to use ‘git-annex’

Many possible options

1. Make your git repository and your data server public
   
   *Wait, making the data server public? How?*

2. Clean up in a specific branch and publish its head
   
   *Just* `git rm` *before* `git annex export`
   
   Large. History remains hidden

3. Same as above but publish the content of a few files
   
   `git annex unannex file; git add file`
   
   *then* `clone with` *a* `--single-branch --depth=1`
   
   History is hidden. SHA256 are visible, anyone can check!

Make it easy for others to import your work
There is even a prototype to use Zenodo as a special remote

- Smooth storing and archiving of file 😊
- Files are identified by their SHA256
- Archiving then amounts to push a tar.gz of the content of your git repository (which points to the SHA256 files)
- Sensitive files could be stored on an encrypted remote and be made available to only a few persons
Conclusion
Clearly define:

- Data stores: servers, USB drives, ...
  - USB drives used to share data or only to extend your laptop?
Clearly define:

- Data stores: servers, USB drives, ...
  - USB drives used to share data or only to extend your laptop?
- Access rights (read/write, privacy/encryption) of both:
  - Git repositories (normal remotes)
  - Data stores (special remotes)
Clearly define:

- Data stores: servers, USB drives, ...
  - USB drives used to share data or only to extend your laptop?
- Access rights (read/write, privacy/encryption) of both:
  - Git repositories (normal remotes)
  - Data stores (special remotes)
- Backup policy
  - Who is allowed to drop files on the server?
  - How much can you trust remotes?
  - Minimal number of copies?
  - Favorite remotes (for bandwidth)
What makes git-annex relevant in our context?

- Protection: corrupted data will be detected

Let's be honest, the learning curve is a bit steep, but it's worth it!
What makes git-annex relevant in our context?

- Protection: corrupted data will be detected
- Made to last: [https://git-annex.branchable.com/future_proofing/](https://git-annex.branchable.com/future_proofing/)

Let's be honest, the learning curve is a bit steep, but it's worth it!
What makes git-annex relevant in our context?

• Protection: corrupted data will be detected
• Made to last: https://git-annex.branchable.com/future_proofing/
• Backup and storage extendability: your data is not locked in an opaque cloud

Let's be honest, the learning curve is a bit steep, but it's worth it!
What makes git-annex relevant in our context?

- Protection: corrupted data will be detected
- Made to last: [https://git-annex.branchable.com/future_proofing/](https://git-annex.branchable.com/future_proofing/)
- Backup and storage extendability: your data is not locked in an opaque cloud
- Location tracking: `git-annex whereis`, `git-annex list`, and `git-annex enableremote`

Let’s be honest, the learning curve is a bit steep, but it’s worth it!
The Elephant in the Room: Climate Change

The Science Is CLEAR

IPCC, IPBES, https://climate.nasa.gov/

1. Global climate change is **not** a future problem


2023 Alberta wildfires (> 1 Mha)
Global climate change is not a future problem. It is entirely due to human activity. 9 out of 10 IPCC scientists believe overshoot is likely.

Limiting warming to 1.5°C and 2°C involves rapid, deep and in most cases immediate greenhouse gas emission reductions.

Net zero CO₂ and net zero GHG emissions can be achieved through strong reductions across all sectors.

The extent to which current and future generations will experience a hotter and different world depends on choices now and in the near-term.


Latest IPCC report 19/20
The Elephant in the Room: Climate Change

1. Global climate change is not a future problem.
2. It is entirely due to human activity.
3. 9 out of 10 IPCC scientists believe overshoot is likely.

Nature survey, Nov. 2021
Put aside biodiversity loss, pollution, freshwater, land system change...

https://www.nosviesbascarbone.org/
Put aside biodiversity loss, pollution, freshwater, land system change...

**French government response**

- **Verdissement de l’industrie**: « pause » sur les normes environnementales
- **Loi de programmation militaire** (+41%)
- **Nous devons préparer la France à une élévation de la température de 4 °C**
- Academia ? PEPR 5G, Cloud, NUMPEX, Quantique, IA, Agroécologie et numérique

---

**Empreinte carbone moyenne en France**

- 10 tonnes de CO₂/an/pers.
- 0,5
- 0,5
- 0,5
- 0,2
- 0,2

**Objectif d’ici 2050**

- - de 2 t de CO₂/an/pers.

---

**Several scenarios on the table**

- Energy optimization/saving ≠ sobriety and frugality

---

**Note:** L’empreinte carbone porte sur les trois principaux gaz à effet de serre (CO₂, CH₄, N₂O). En 2021, la méthodologie a été ajustée afin de mieux tenir compte de l’évolution des coûts du pétrole brut, du gaz et du charbon ; l’ensemble de la série a ainsi été révisée, l’ensemble des ajustements portant sur les émissions importées de CH₄.

Champ : périmètre Île-de-France métropole et outre-mer appartenant à l’UE.
Sources : Citépa ; ADEME ; PNA ; Douanes ; Eurostat ; Insee ; Trimestriel ; SDES, 2021

---

**Verdissement de l’industrie:** « pause » sur les normes environnementales

**Loi de programmation militaire** (+41%)

**Nous devons préparer la France à une élévation de la température de 4 °C**

**Academia ?** PEPR 5G, Cloud, NUMPEX, Quantique, IA, Agroécologie et numérique

---

**https://www.nosviesbascarbone.org/**
Put aside biodiversity loss, pollution, freshwater, land system change...

### French government response

- **Verdissement de l’industrie**: « pause » sur les normes environnementales
- **Loi de programmation militaire** (+41%)
- **Nous devons préparer la France à une élévation de la température de 4 °C**
- Academia ? PEPR 5G, Cloud, NUMPEX, Quantique, IA, Agroécologie et numérique

### Several scenarios on the table

- What will research/CS look like/be used for in such a world?
- Energy optimization/saving ≠ sobriety and frugality