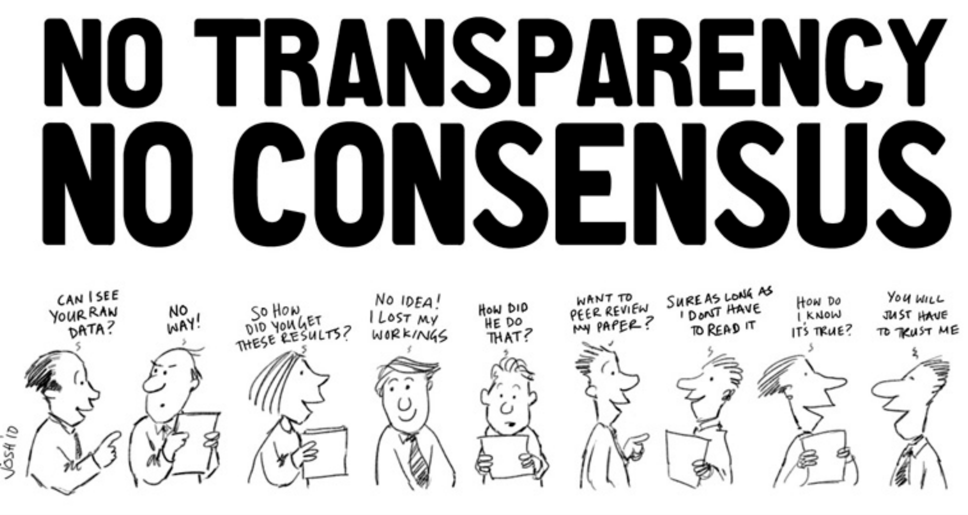
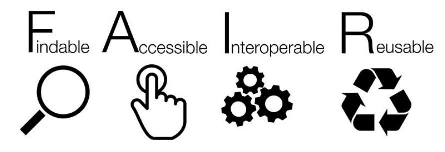
Scientific Data Management with Git and Git-Annex

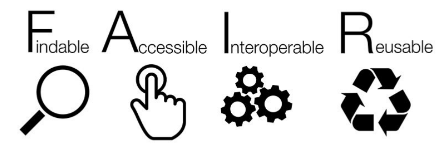
Arnaud Legrand

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


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



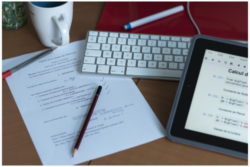


Good research requires time and resources

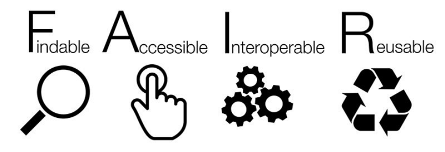
Train yourself and your students: RR, statistics, experiments

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

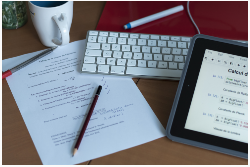


Good research requires time and resources

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

Git

- 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

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`) ☹

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 <u>not</u> commit large binary files and put them in a shared directory instead

- When and who did what, and why?
  - Indicate *when* (and *who*) in <u>external</u> 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) ⇝ `git` becomes slow for you (`checkout`, `diff`, `push`, …) and others (`clone`, `pull`) ☹

Option 2 Let's <u>not</u> commit large binary files and put them in a shared directory instead

- When and who did what, and why?
  - Indicate *when* (and *who*) in <u>external</u> 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* 😎

Christmas list for data management

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

Christmas list for data management

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

Christmas list for data management

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

Christmas list for data management

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
4. Manage different (possibly unreliable) storage media
   - While ensuring data integrity

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**

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/`)

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

Git Annex principles (1/2)

```
tree
```

```
├── data.csv
└── big_file.pdf
```

Git Annex principles (1/2)

```
git add data.csv ;    git annex add big_file.pdf
```

```
├── data.csv
└── big_file.pdf -> .git/annex/objects/KJ/QF/SHA256E-s776715--4b2aef98a8a706be4eedbf390ba
                         SHA256E-s776715--4b2aef98a8a706be4eedbf390ba724a64d75bdf295d603951773230a37
```

- 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 Annex principles (1/2)

```
tree
```

.

- 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 Annex principles (1/2)

```
git clone https://gitlab.com/alegrand/myrepos.git
```

```
├── data.csv
└── big_file.pdf -> .git/annex/objects/KJ/QF/SHA256E-s776715--4b2aef98a8a706be4eedbf390ba
            SHA256E-s776715--4b2aef98a8a706be4eedbf390ba724a64d75bdf295d603951773230a37
```

- 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

Git Annex principles (1/2)

```
git annex get big_file.pdf
```

```
├── data.csv
└── big_file.pdf -> .git/annex/objects/KJ/QF/SHA256E-s776715--4b2aef98a8a706be4eedbf390ba
              SHA256E-s776715--4b2aef98a8a706be4eedbf390ba724a64d75bdf295d603951773230a37
```

- 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`

Git Annex principles (1/2)

```
git annex drop big_file.pdf
```

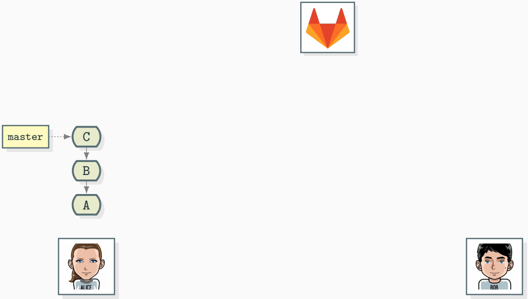
```
├── data.csv
└── big_file.pdf -> .git/annex/objects/KJ/QF/SHA256E-s776715--4b2aef98a8a706be4eedbf390ba
              SHA256E-s776715--4b2aef98a8a706be4eedbf390ba724a64d75bdf295d603951773230a37
```

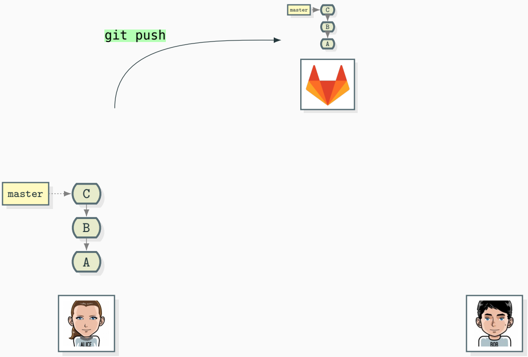
- The project is populated with **symbolic links** to the large files which end up in `.git/annex/objects` (`git annex add`)
  - ⇝ No wasted space with file duplication
  - Large files are identified by their content (SHA256 by default)
- `git clone` will retrieve only symbolic links for annexed files
  - ⇝ Get (and check) content with `git annex get`
- Files may be `git annex drop`ed (from the annex)

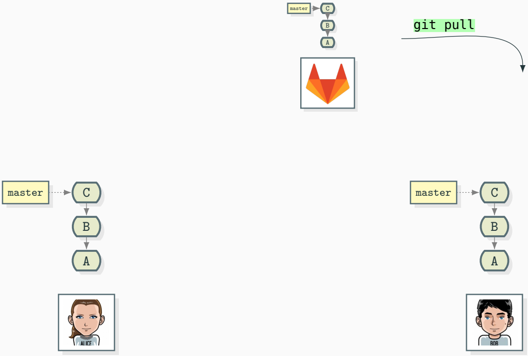
Git Annex principles (2/2)

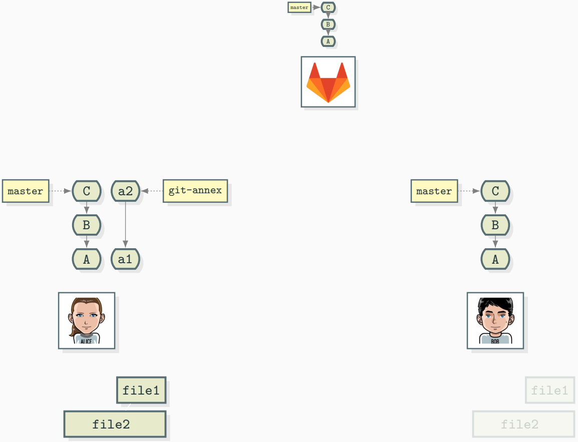
- Special remotes are ways to access files
  - A USB key, a server through SSH or webdav, a web server, Amazon S3, etc.
  - They have their own structure and do not comprise the git history
  - Files may be migrated/duplicated between (special) remotes
  - Information on the remotes is stored in a special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

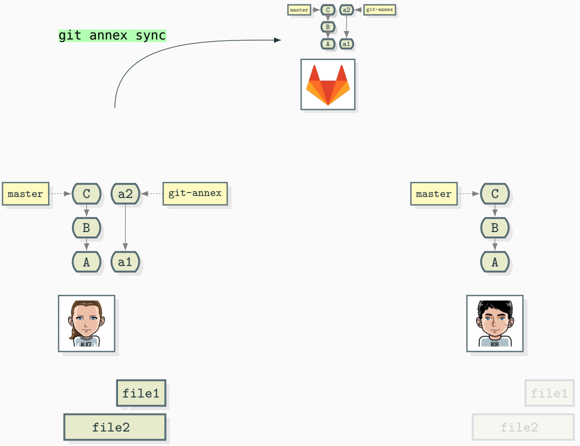
Illustration? Wait for it!

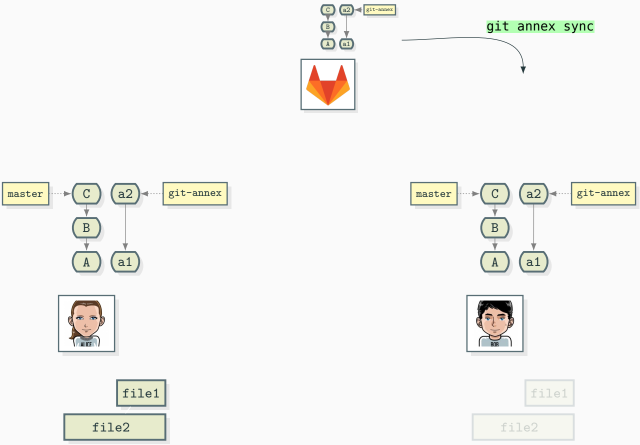

master
C
B
A
ALICE
BOB

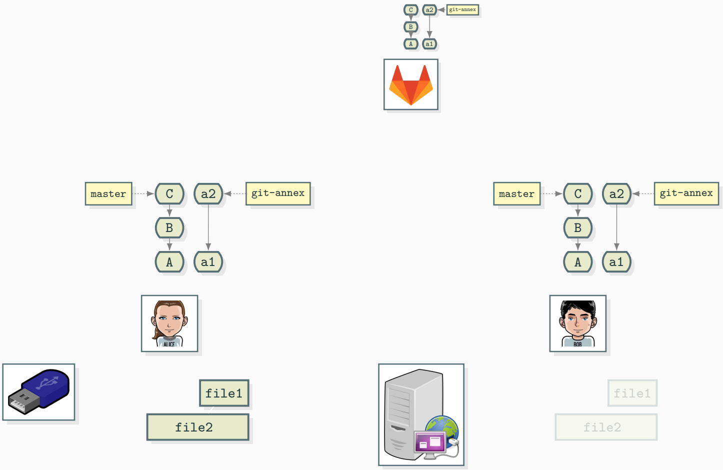

git push
master
C
B
A
master
C
B
A
ALICE
BOB

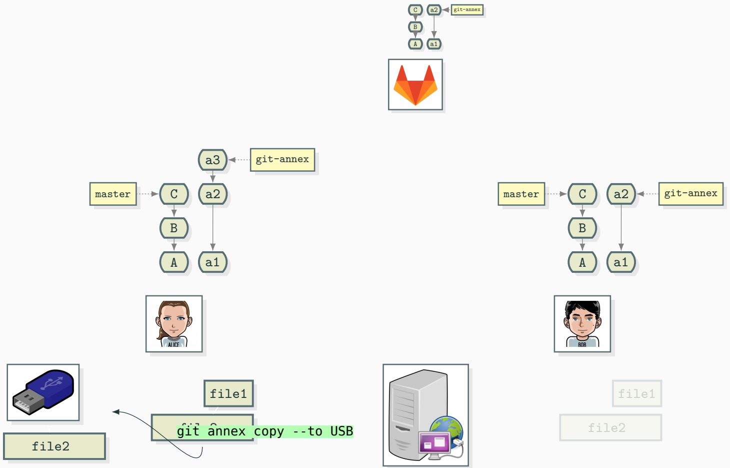

master
C
B
A
git pull
master
C
B
A
ALICE
master
C
B
A
BOB

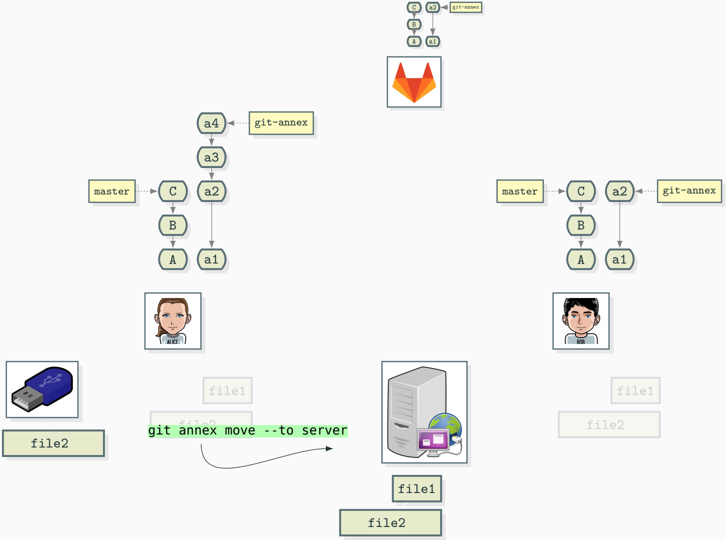

master
C
B
A
master
C
a2
git-annex
B
A
a1
ALICE
file1
file2
master
C
B
A
BOB
file1
file2

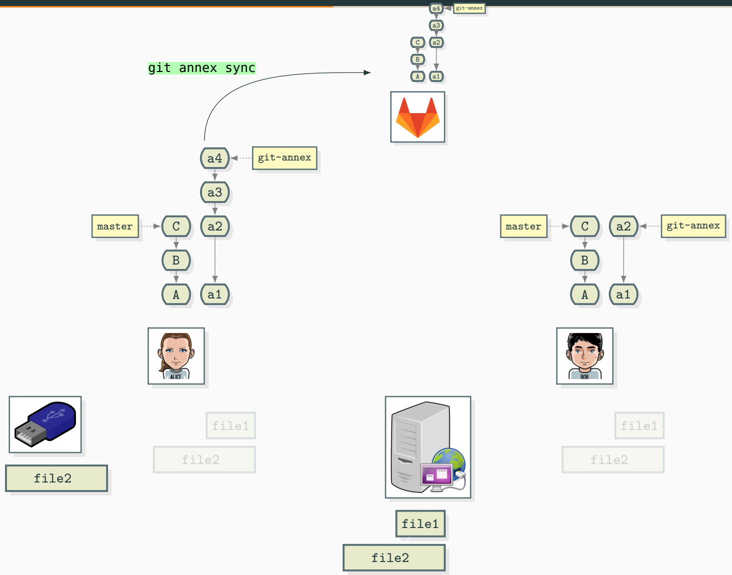

git annex sync
master
C
a2
git-annex
B
A
a1
master
C
a2
git-annex
B
A
a1
ALICE
file1
file2
master
C
B
A
BOB
file1
file2

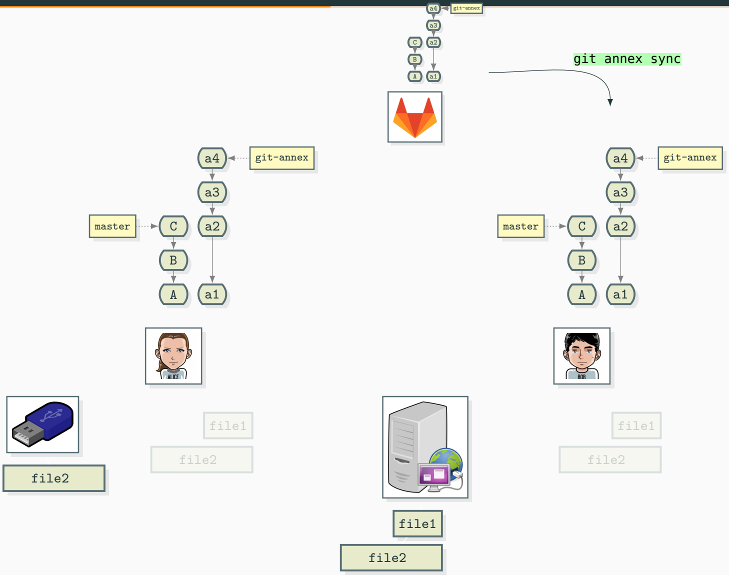

git annex sync
C
a2
git-annex
B
A
a1
master
C
a2
git-annex
B
A
a1
file1
file2
master
C
a2
git-annex
B
A
a1
file1
file2

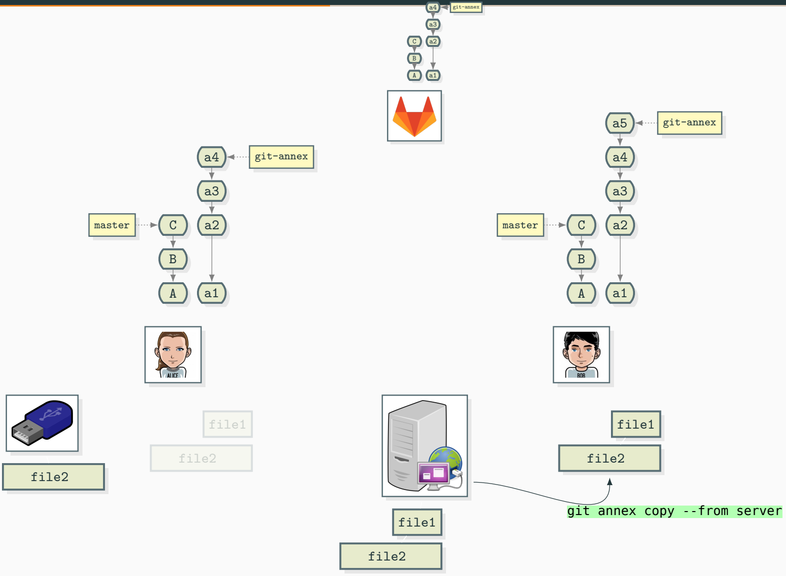

C
a2
git-annex
B
A
a1
master
C
a2
git-annex
B
A
a1
ALICE
file1
file2
master
C
a2
git-annex
B
A
a1
BOB
file1
file2

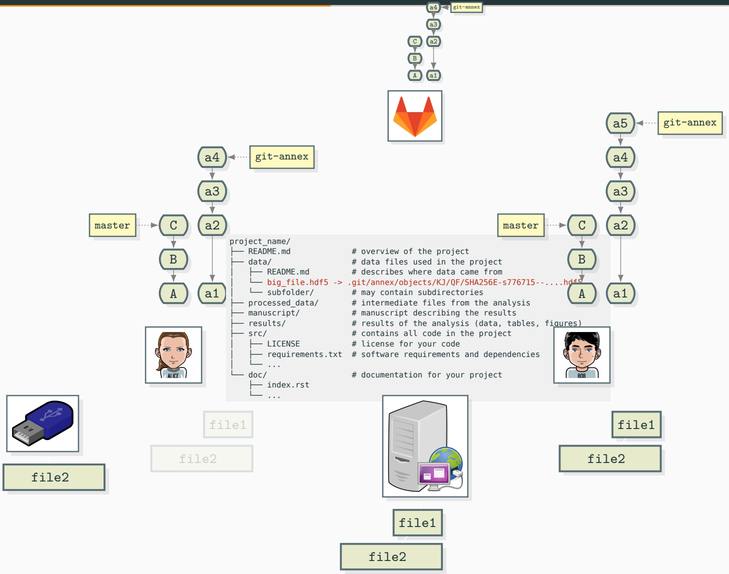

C
a2
git-annex
B
A
a1
a3
git-annex
master
C
a2
B
A
a1
master
C
a2
git-annex
B
A
a1
ALICE
BOB
file1
file2
git annex copy --to USB
file1
file2


C
a2
git-annex
B
A
a1
a4
git-annex
a3
master
C
a2
B
A
a1
master
C
a2
git-annex
B
A
a1
ALICE
BOB
file2
file1
git annex move --to server
file1
file2
file1
file2

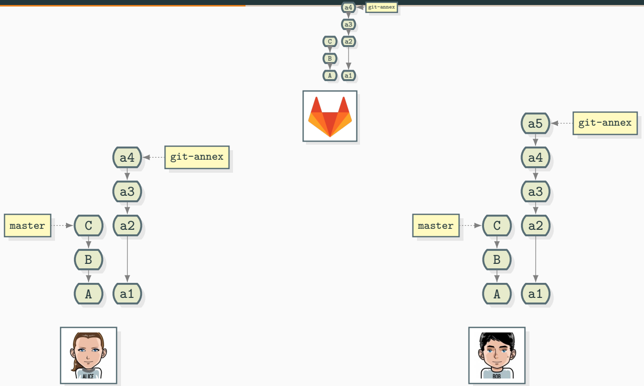

git annex sync
a4
a3
C
a2
B
A
a1
git-annex
master
C
B
A
a2
a1
git-annex
master
C
B
A
a2
a1
git-annex
file1
file2
file2
file1
file2
file1
file2


git annex sync
git-annex
a4
a3
a2
a1
C
B
A
master
git-annex
a4
a3
a2
a1
master
C
B
A
git-annex
a4
a3
a2
a1
master
C
B
A
file2
file1
file2
file1
file2
file1
file2

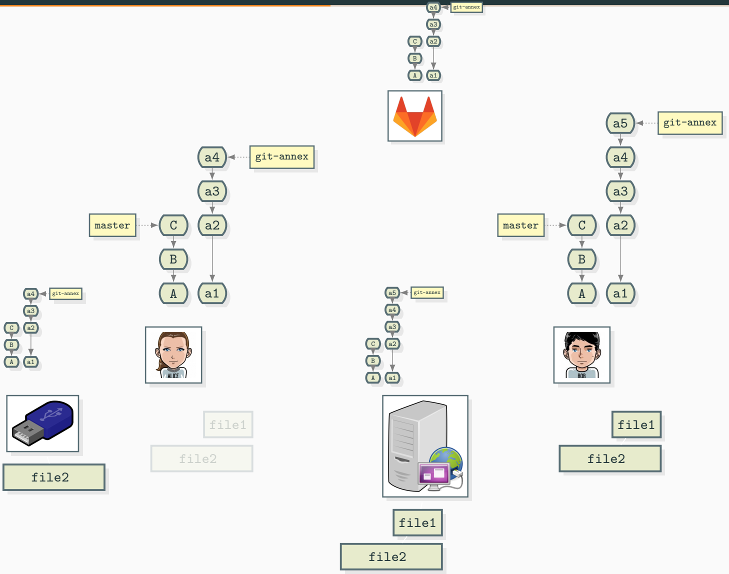

git-annex
a4
a3
C
a2
B
A
a1
a5
git-annex
a4
a3
master
C
a2
B
A
a1
git-annex
a4
a3
master
C
a2
B
A
a1
ALICE
BOB
file2
file1
file2
file1
file2
file1
file2
git annex copy --from server




```
project_name/
├── README.md              # overview of the project
├── data/                  # data files used in the project
│   ├── README.md          # describes where data came from
│   ├── big_file.hdf5 -> .git/annex/objects/KJ/QF/SHA256E-s776715--....hdf5
│   └── subfolder/         # may contain subdirectories
├── processed_data/        # intermediate files from the analysis
├── manuscript/            # manuscript describing the results
├── results/               # results of the analysis (data, tables, figures)
├── src/                   # contains all code in the project
│   ├── LICENSE            # license for your code
│   ├── requirements.txt   # software requirements and dependencies
│   └── ...
└── doc/                   # documentation for your project
    ├── index.rst
    └── ...
```




```
.git/annex/objects/
├── 05
│   └── wJ
│       └── SHA256E-s742--a7f60ced39a5c83adc3152707b6f53b42cec1319223e66869faa761ece3a8b9a.json
│           └── SHA256E-s742--a7f60ced39a5c83adc3152707b6f53b42cec1319223e66869faa761ece3a8b9a.json
├── 1W
│   └── Qq
│       └── SHA256E-s8392320--1c31b7165f51ffb529ce1e068f532bddb2a0188b5576e0c3bc66605de6a17d2b.FTS
│           └── SHA256E-s8392320--1c31b7165f51ffb529ce1e068f532bddb2a0188b5576e0c3bc66605de6a17d2b.FTS
├── 3f
│   └── 2j
│       └── SHA256E-s8392320--666c6a82e73992427d1fcb251c9c854a941cffb435626b899ad4e1e2b155fef3.FTS
│           └── SHA256E-s8392320--666c6a82e73992427d1fcb251c9c854a941cffb435626b899ad4e1e2b155fef3.FTS
├── 4X
│   ├── GJ
│   │   └── SHA256E-s605--b053f4378ec9145613d198c81820edeef7eb0987108b42fe57bdfe461bc46e4f.json
│   │       └── SHA256E-s605--b053f4378ec9145613d198c81820edeef7eb0987108b42fe57bdfe461bc46e4f.json
│   └── mx
│       └── SHA256E-s2102400--297630a3e5fa3030dbdd6e4e14efd87678c778210fdad6fb3ff7030f4f60c0fc.FTS
│           └── SHA256E-s2102400--297630a3e5fa3030dbdd6e4e14efd87678c778210fdad6fb3ff7030f4f60c0fc.FTS
├── 55
│   ├── f7
│   │   └── SHA256E-s2102400--49b875863775ad54d7a5ca0ce678a1f5edf0398875214ffa9083535d8956d7b3.FTS
│   │       └── SHA256E-s2102400--49b875863775ad54d7a5ca0ce678a1f5edf0398875214ffa9083535d8956d7b3.FTS
│   └── QZ
│       └── SHA256E-s3424--dec8ab57c92c3eb4fdbaee905644250b9c138a25ffc44c9676bed9e5a57c1c53.json
```


git-annex
a5
a4
a3
a2
a1
master
C
B
A
ALICE
BOB
file1
file2

Data integrity

- 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

Git Annex for data integrity (1/3)

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

Git Annex for data integrity (1/3)

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

Git Annex for data integrity (1/3)

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.

```
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`)

Git Annex for data integrity (2/3)

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?

Git Annex for data integrity (2/3)

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?

Git Annex for data integrity (2/3)

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?

Git Annex for data integrity (2/3)

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`

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"
```

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

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 betwen team members

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

Bonus: Files stored on special remotes can easily be encrypted!

Git Annex for data integrity (3/3)

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?

Git Annex for data integrity (3/3)

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`ing
   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!

Git Annex for data integrity (3/3)

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`ing
   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

Preparing the archiving

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

Attention points

Clearly define:

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

Attention points

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)

Attention points

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

What makes git-annex relevant in our context?

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

What makes git-annex relevant in our context?

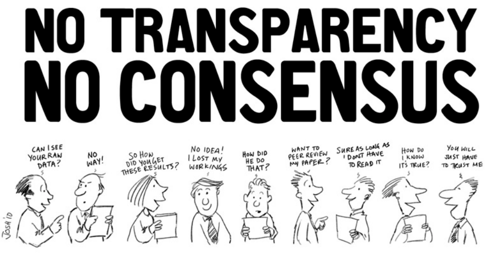
- Protection: corrupted data will be detected
- Made to last: https://git-annex.branchable.com/future_proofing/
- Backup and storage extendability: your data is not locked in an opaque cloud

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
- Location tracking: `git-annex whereis`, `git-annex list`, and `git-annex enableremote`

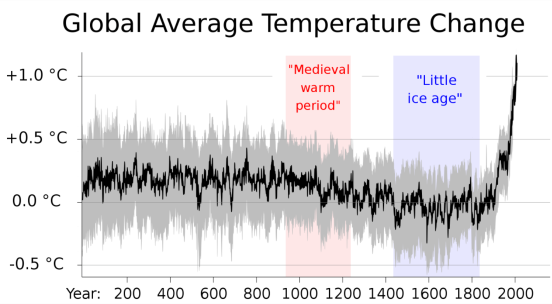
Let's be honest, the learning curve is a bit steep,

but it's worth it!

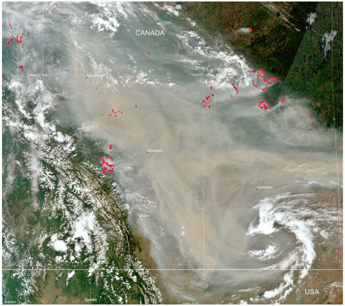


IPCC, IPBES, https://climate.nasa.gov/

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



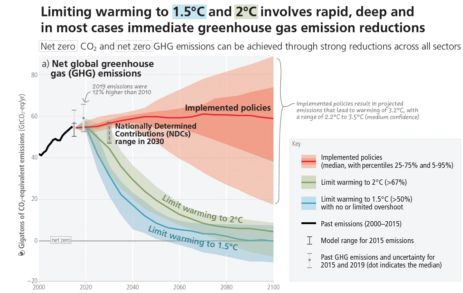
https://en.wikipedia.org/wiki/Global_temperature_record

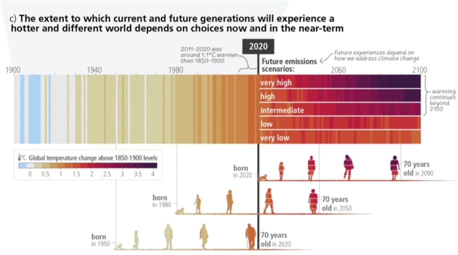


2023 Alberta wildfires (> 1 Mha)

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

1. Global climate change is **not** a future problem
2. It is **entirely** due to human activity



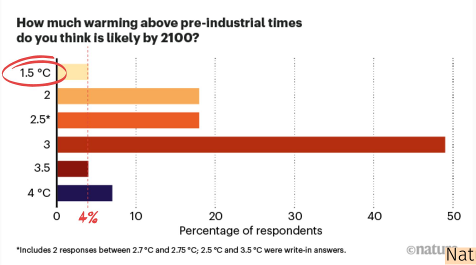


Paris Agreement'15 ∼ Net Zero by 2050

Latest IPCC report

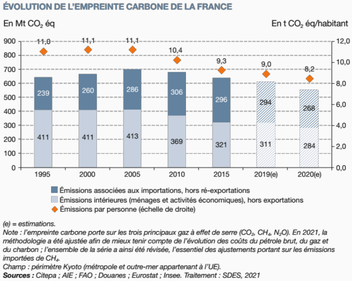
IPCC, IPBES, https://climate.nasa.gov/

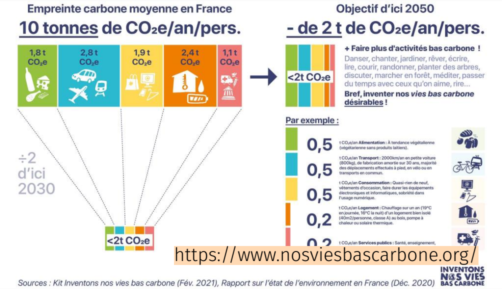
1. Global climate change is **not** a future problem
2. It is **entirely** due to human activity
3. *9 out of 10 IPCC scientists believe overshoot is likely*



Nature survey, Nov. 2021

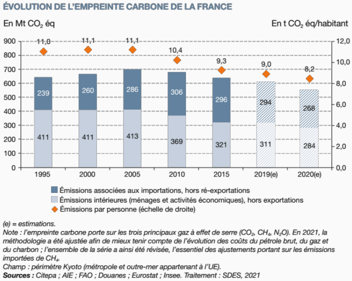
Put aside biodiversity loss, pollution, freshwater, land system change...

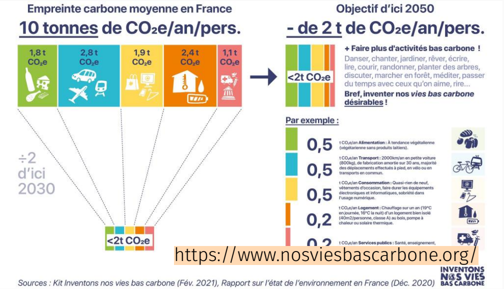




https://www.nosviesbascarbone.org/

Put aside biodiversity loss, pollution, freshwater, land system change…



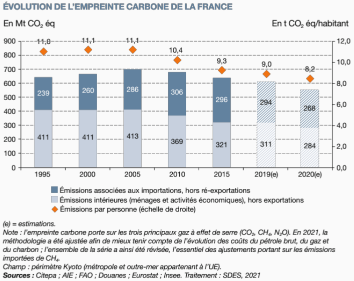


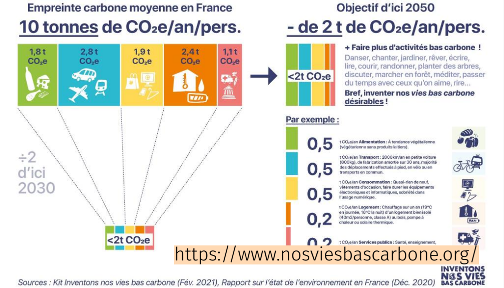
https://www.nosviesbascarbone.org/

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

Put aside biodiversity loss, pollution, freshwater, land system change...





https://www.nosviesbascarbone.org/

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 $\neq$ sobriety and frugality