A problem shared is a problem solved!

managing data the Open Source way

Robert Forkel

Department for Cultural and Linguistic Evolution
Max Planck Institute for the Science of Human History

Sharing problems

- Generally, having the same problems as others is good, insofar as this broadens the pool for potential problem solvers.
- But to profit from solutions others have found, you need to know it was the same problem to begin with.
- And even then, there are multiple ways to adopt solutions.

What to do with others' solutions?

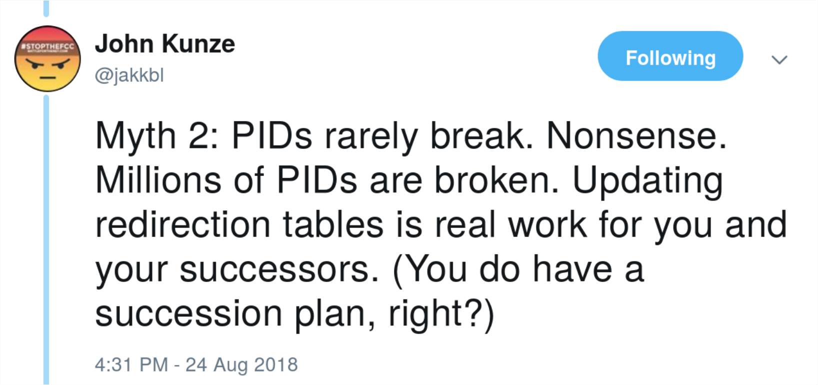
- Often, having analogous problems does not necessarily lead to adopting others' solutions,
- but rather to adoption of the idea with modifications: ”We want the same thing, just with X”
- Thus incurring maintenance costs by owning the solution instead of just stealing it.

An Example: DOI

Digital Object Identifiers

- Problem: Make persistent identification of the scholarly record possible.
- Solution: A managed redirection layer on top of HTTP URLs.
- Arguably the problem solved by DOI can be solved with ”cool” URIs or HTTP redirection alone.
- We know about the problems with the ”simple” solution. But fact is that the ”better” solution comes at a price (and may not be that much better).

Example: DOI


John Kunze
@jakkbl

Following

Myth 2: PIDs rarely break. Nonsense. Millions of PIDs are broken. Updating redirection tables is real work for you and your successors. (You do have a succession plan, right?)

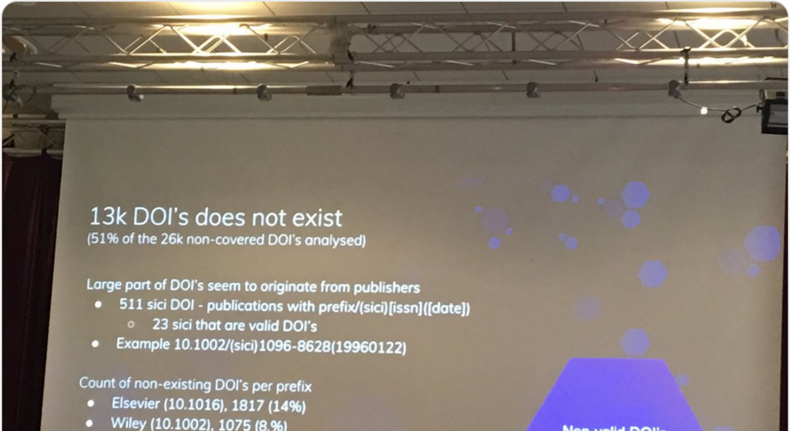
4:31 PM - 24 Aug 2018

Example: DOI

Alice Meadows
@alicejmeadows

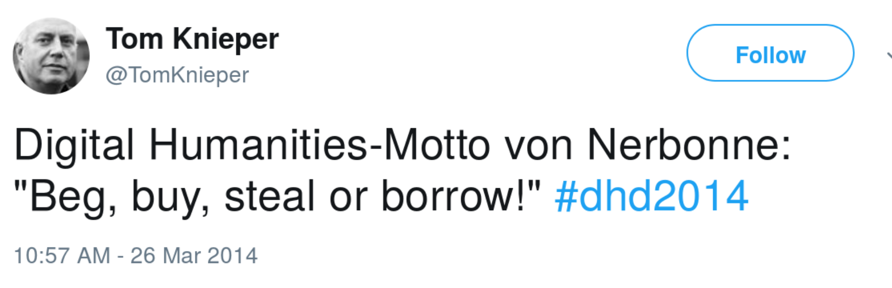
Follow

Overview of 13k DOIs that don't exist - mostly publisher owned #PIDapalooza19



Steal

So instead on adopting ideas for solutions this talk will focus on "stealing" – as in



Don't beg for programmer resources or buy custom solutions – steal what works for others.

Who to steal solutions from?

- Software developers are tool-makers and problem solvers
- *Data is Code!* – or at least often similar enough
- So let’s see whether we can **curate linguistic research data** using Open Source tools.
- Often it just needs a bit of translation to discover shared problems!

So in the following we will look at the available tools.

First things first

Version control is essential for traceable data curation, thus allowing incremental progress.

It let’s us answer the questions

- who?
- changed what?
- when?

”backup, backup, backup” – but also use version control, because it allows to batch related changes into **commits**.

git as seen through GitHub

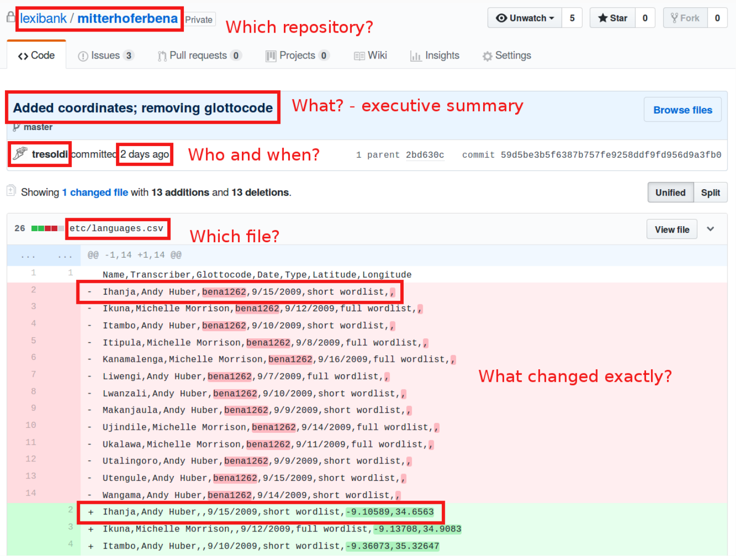
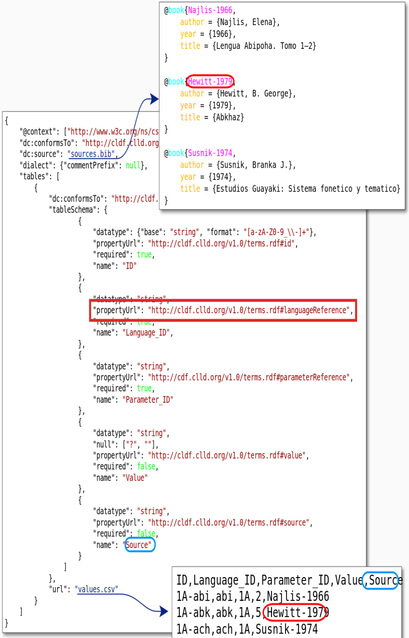


Figure 1: A git commit as displayed on GitHub

Digression: Line-based text formats



To make the most of version control, data should be in line-based text formats.

- **CLDF** – the Cross-Linguistic Data Format – tries to do just that.
- Toolbox' SFM format is quite ok in that respect, too!
- But OLAC metadata records could fit as well.
- Digital Notebooks.

Distributed version control

In **Distributed version control** systems each copy of the repository stores the complete history.

This may sound like un-neccessary complication, but it allows using version control without a server (or network).

working offline = using git locally

synching = merge from/push to other clones of the repository

GitHub

- collaboration platforms like GitHub add support for online collaboration on git repositories
- GitHub can be said to be all about ”Connecting Communities, Languages & Technology” :)
- Language resources are already on GitHub:
    - https://github.com/LowResourceLanguages/endangered-languages
    - Glottolog
    - Mother Tongues dictionaries

Data curation workflows the Open Source way

So we have our data versioned and on GitHub – how do we collaborate?

Submission	= Pull Request (a GitHub thing)
Review	= Pull Request review
Acceptance	= "merge into master"
Publication	= release

Note: Using this workflow you can also contribute to Glottolog!

Archiving and Persistence

- GitHub is not an archiving or publication platform, though.
- ZENODO fills that gap:
    - Can pick up releases of GitHub repositories automatically
    - Provides longterm archiving ...
    - ...and access via a DOI
    - You can get a DOI for your Mother Tongues dictionary!

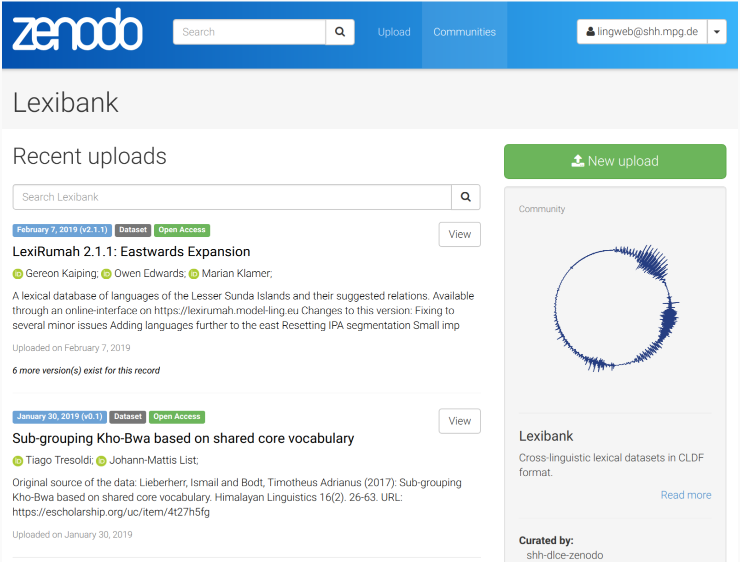


Figure 2: Publication and archiving platforms like ZENODO add persistence.

Adopting Open Source paradigms

Now we know how to publish the data, but what if **it's not finished yet!**

- *release early – release often*
  `https://en.wikipedia.org/wiki/Release_early,_release_often`
- For bonus points, specify maturity in metadata (a la ”trove classifiers”)

releasing often may create headaches for data consumers, though ...

Semantic Versioning for Datasets

snim2 commented on Feb 24, 2015

Collaborator

This is related to Issues #9, #10 and #11.

Datasets may well evolve over time. When reading a paper which describes an experiment on a particular dataset it should be possible to find out which *version* of the dataset was used to produce the documented results. This aids reproducibility.

Semantic versioning is a common technique in software development. The idea is to provide a version number for the data which looks like: `MAJOR.MINOR.PATCH`

- the `MAJOR` version of the data, which should represent significant changes. In the case of datasets, this might mean that an experiment using version `1.0.0` of the dataset could not be run on version `2.0.0` without making some changes to the experiment, or the analysis of the results
- the `MINOR` version of the data which is compatible with other versions of the data which have the same `MAJOR` version. In the case of datasets, this might mean that any experiment or analysis performed on version `1.0.0` of the data should be repeatable with version `1.1.0` of the data.
- the `PATCH` version for bug fixes. For example version `1.0.1` of a dataset may fix a typo in version `1.0.1`.

Semantic Versioning 2

For data creators:

patch release a.k.a. bugfix = fixing ”typos”

minor release = additional data, but nothing changing the ”gist” of it

major release = ”backwards incompatible changes”, different data, different data layout

For data consumers:

- You’d always want to upgrade to the latest patch release of your chosen minor release to avoid working with ”buggy” data
- Upgrading to the next minor version may change your analysis results but shouldn’t break your analysis pipeline
- Upgrading to the next major version may require adapting your analysis code.

Digression: Re-usability

Why would you want to adopt semantic versioning?

To increase **”re-usability”** (the R in FAIR)!

”Usage by others” is a very good proxy for usability of data:

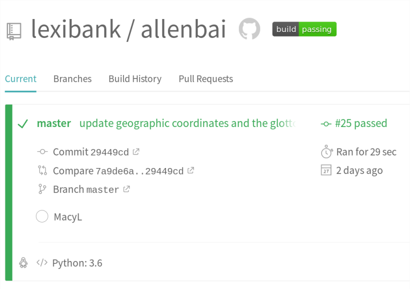
- if no one is using it, it’s probably unusable.
- if others use it, it’s clearly usable.

Thus, to adhere to FAIR data principles, do everything that increases the chances of others using your data.

Continuous Integration

Releasing often is made a lot easier, if you know your data is consistent at all times.

Continuous Integration services allow automatic consistency checks for your data!



Continuous Integration

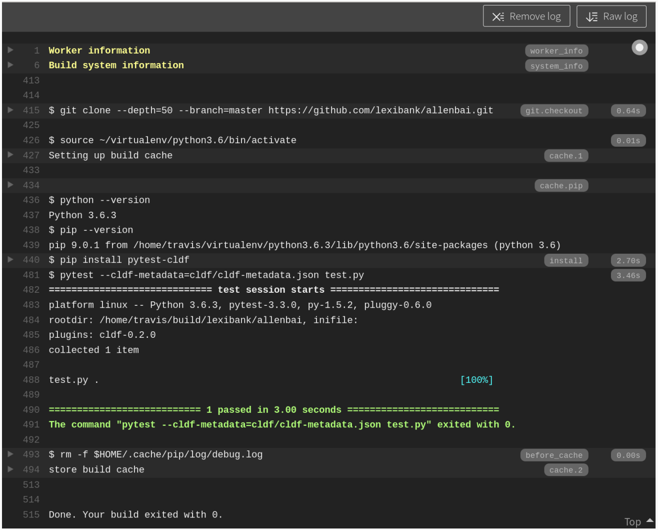


Figure 3: There's a pytest plugin for CLDF datasets!

Package Managers

- Now that our datasets are curated like Software packages, retrieving a dataset could work like **installing** a software package!
- This requires
    - a dataset archive which acts like e.g. a Linux distribution.
    - a package manager to access this archive

pip for data creators

Following the spirit laid out above the solution may just be: Turn your data repository into a python package and use an existing package manager!

- add a `setup.py` file to the repository.
  This allows you to manage your local datasets using python's standard package management tool `pip`:
    - retrieve (particular versions of) the package
    - inspect which datasets you have "installed"
    - "freeze" the state of your local datasets
    - "upgrade" datasets
- host your repository on a hosting platform known to pip
  let's pip also handle the download/clone/update

pip for data creators

```python
setup(
    name='lexibank_allenbai',
    description=metadata['title'],
    license=metadata.get('license', ''),
    url=metadata.get('url', ''),
    py_modules=['lexibank_allenbai'],
    include_package_data=True,
    zip_safe=False,
    entry_points={
        'lexibank.dataset': [
            'allenbai=lexibank_allenbai:Dataset',
        ]
    },
    install_requires=[
        'pylexibank>=0.11',
    ],
    extras_require={
        'test': ['pytest-cldf'],
    }
)
```

pip for data consumers

Now replicating a particular state of your local data is easy:

- specify all your ”dependencies”, i.e. datasets your analysis depends on, in a `requirements.txt` file,
  - this file can be read by `pip` ...
  - ...and written by `pip`, to document the local state!
- run
  `pip install -r requirements.txt`
  and watch the datasets in the correct versions being downloaded to your computer.

pip for data consumers

Branch: **master** ▾ **clics2** / **datasets.txt**

xrotwang fixed dataset stats and upgraded lexibank-ids to v1.2

1 contributor

17 lines (15 sloc) | 1.14 KB

```
-e git+https://github.com/lexibank/allenbai.git@v1.0#egg=lexibank_allenbai
-e git+https://github.com/lexibank/bantubvd.git@v1.0#egg=lexibank_bantubvd
-e git+https://github.com/lexibank/beidasinitic.git@v2.0#egg=lexibank_beidasinitic
-e git+https://github.com/lexibank/bowernpny.git@v1.1.1#egg=lexibank_bowernpny
-e git+https://github.com/lexibank/hubercolumbian.git@v1.0#egg=lexibank_hubercolumbian
-e git+https://github.com/lexibank/ids.git@v1.2#egg=lexibank_ids
-e git+https://github.com/lexibank/kraftchadic.git@v1.0#egg=lexibank_kraftchadic
-e git+https://github.com/lexibank/northeuralex.git@v1.0#egg=lexibank_northeuralex
-e git+https://github.com/lexibank/robinsonap.git@v1.1#egg=lexibank_robinsonap
-e git+https://github.com/lexibank/satterthwaitetb.git@v1.0#egg=lexibank_satterthwaitetb
-e git+https://github.com/lexibank/suntb.git@v1.1#egg=lexibank_suntb
-e git+https://github.com/lexibank/tls.git@v1.1#egg=lexibank_tls
-e git+https://github.com/lexibank/tryonsolomon.git@v1.0.1#egg=lexibank_tryonsolomon
-e git+https://github.com/lexibank/wold.git@v1.1#egg=lexibank_wold
-e git+https://github.com/lexibank/zgraggenmadang.git@v1.1#egg=lexibank_zgraggenmadang

```

Digression: Replicability

For once a concept that goes largely under the same name in software development and research.

In software development the things you want to replicate are

- bugs (if you cannot replicate a bug someone else has encountered, you have a hard time fixing it)
- deployments (if your software does not replicate the same behaviour on a different machine, you cannot distribute it)

The main tool to ensure replicability in software development is controlling/replicating the entire software stack as much as possible. So there should be easy ways to figure out whether two systems are the same, or in which way they differ.

Replicability

The same should be true for datasets: To help with replicabiity of research, it must be easy to figure out the differences between datasets – or if they are the same!

That's exactly what the setup described above is supposed to achieve.

If all you have (stolen) is a hammer ...

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

...don’t try to build a nail gun ...

...look harder for nails!