File synchronization between Linux systems in Python with YARsync

Yaroslav Nikitenko [1]

RWTH Aachen University

PyHEP.dev 2024
29 August, 2024

[1] metst13 (at) gmail.com

Data analysis

Software	Data
	?
git	rsync,Back In Time, Grsync,LuckyBackup,rsnapshot, casync,Duplicity, Git-annex,Rclone, *Continuous synchronisation:* gut-sync,Syncthing,Unison, Dropbox,Google Drive, Yandex Disk,... (closed-source)

Data analysis

Software	Data
git	Yet Another Rsync
Changing (text) files	Unchanged (data) files

Data synchronization software

Goals of having several copies of data:

- Flexibility/independence. Work offline.
- Safety. Data won't be lost.
- Unification. Same data on different machines. *Repository - but not backup*.

Operations on files/directories in a repository

- add
- remove
- rename

Rsync. First steps

On server:

```
[server]$ mkdir my_data
[server]$ mv file1 .. fileN my_data
```

On local machine:

```
[local]$ rsync -av myserver:~/my_data/ my_data
```

Rsync vs renames

On server:

```
[server]$ cd my_data
[server]$ mkdir version1
[server]$ mv file* version1/
```

Sometimes *rsync* can not recognize moves/renames, that is it does not do efficiently renames.
But let us use *rsync* as our backend.

Introduce YARsync

```
$ # alias ys=yarsync
$ cd my_data
$ # add files
$ yarsync init
$ # Creates a configuration directory .ys/
$ # add/remove files
$ yarsync commit -m "Initial commit"
```

Introduce YARsync

```
$ # alias ys=yarsync
$ cd my_data
$ # add files
$ yarsync init
$ # Creates a configuration directory .ys/
$ # add/remove files
$ yarsync commit -m "Initial commit"
```

We have a *configuration directory* .ys and a commit in .ys/commits.
A commit is a hard-link copy of the *working directory*.

- Does not take additional space.
- Allows efficient tracking of renames for *rsync*.

Local workflow

```
$ # alias ys=yarsync
$ # work in the repository
$ ys status  # see the changes
$
$ ys commit
$ # print last three logs
$ ys log -n 3
$
$ yarsync remote add my_server myserver:~/my_repo
# "remote" can also be an external drive
$
$ # get help for a specific command
$ yarsync remote add --help
$
$ # read the manual for more commands
$ man yarsync
```

Synchronizing workflow

```
$ # alias ys=yarsync
$ ys commit -m 'Local commit'
$
$ ys remote -v
my_server     myserver:~/my_repo
$
$ ys push my_server
$
$ # work and commit on server
$
$ # locally:
$ ys pull my_server
```

Conflict resolution

```
# work on local and remote, commit
#    -> different repository states
```

1. Discard changes

```
$ # discard local changes
$ ys pull --force my_server
$
$ # discard remote changes
$ ys push -f my_server
```

Conflict resolution

1. Discard changes, or
2. Merge changes.

```
$ ys pull --new my_server
# pull remote commits, working directories are merged
$ # optional:
$ ys checkout <most appropriate commit>
$ # or use the merged working directory
$ # rearrange files manually into the desired state
$
$ ys commit
$ # a commit "resolves" the conflict
$
$ ys push my_server
```

Overview

Good interface and *simple implementation* allows our program to be:

- distributed. No central repository. Supports manual conflict resolution.
- efficient. Does not re-transmit transmitted files (handles renames).
- transparent. Working directory and commits are filesystem directories, where standard UNIX tools can be run: *find*,... .
- non-intrusive. Does nothing to the working directory. YARsync repository can be removed by *rm -rf .ys*. C.f. *git-annex or backup software*.
- safe (on top of *rsync*). Use *--dry-run* to see what is going to happen.
- simple (for Linux users). Familiar commands from *git*, can be used with *myrepos*. Simple configuration and storage. Can be repaired.

Limitations

- relies on *rsync* and *Python*. Remote can have only *rsync*.
- *rsync* gets slow with **millions of files**.
- Filesystem must support **hard links**. ext2-4, HFS+, NTFS,.... Not supported on FAT. `https://yarsync.readthedocs.io/en/latest/details.html#hard-links`
- Oblivious to different filesystems (transfers between FS with different file attributes).

Code

Python wrapper around *rsync* (uses *subprocess*, *argparse*).

- **Packaged** for PyPI, Debian/Ubuntu and Arch Linux.
- **Used** `https://github.com/ynikitenko/yarsync`, popular with 33 stars, listed on Arch Wiki.
- **Documented** (readthedocs and *man*).
- **Tested**, $\sim 80\%$ coverage.

Code

Python wrapper around *rsync* (uses *subprocess*, *argparse*).

- **Packaged** for PyPI, Debian/Ubuntu and Arch Linux.
- **Used** `https://github.com/ynikitenko/yarsync`, popular with 33 stars, listed on Arch Wiki.
- **Documented** (readthedocs and *man*).
- **Tested**, $\sim 80\%$ coverage.

Thank you for your attention!

Changing files

1. Overwrite changes.

```
$ ys pull
$ # or
$ ys push
```

2. Create versions manually.

```
$ cp my_file my_file_v1
$ # now we have two independent versions
$ ys commit
$
$ ys pull
$ # my_file gets updated, my_file_v1 stays.
```