The Not-forking Software Reproducibility Tool

Not-forking lets you integrate non-diffable codebases, like patch/sed/diff/cp/mv rolled into one. Not-forking is a standard, machine-readable way of answering this question:

> What is the minimum difference between multiple source trees, and how can this difference be applied as versions change over time?

Here is a picture of the simplest use case. External software, here called *Upstream*, forms a part of a new project called *Combined Project*. *Upstream* is not a library provided on your system, because then you could simply link to *libupstream*. Instead, *Upstream* is source code that you copy into the *Combined Project* directory tree like this:

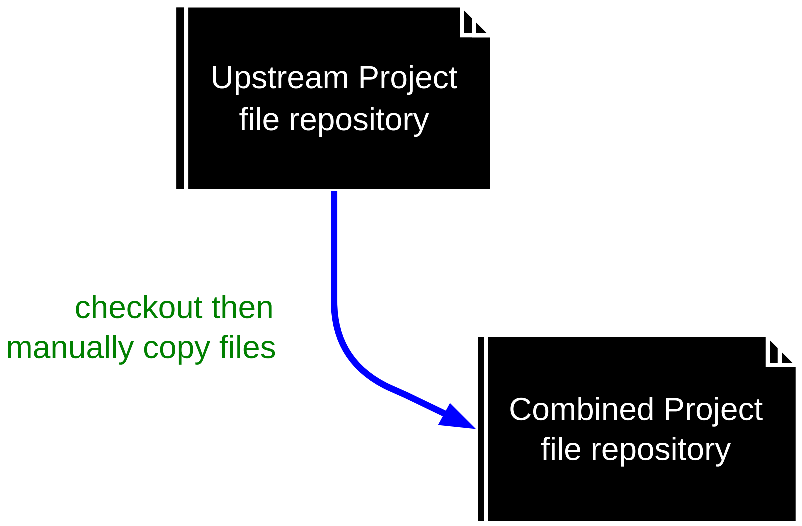
Upstream Project file repository
checkout then manually copy files
Combined Project file repository

Diagram 1: Not-forking Problem Overview

There are some obvious questions in this example:

- Should you import *Upstream* into your source code management system? All source code should be under version management, but having a checkout of an external repository within your local repository feels wrong... and do we want to lose upstream project history?
- If *Upstream* makes modifications, how can you pull those modifications into *Combined Project* safely?
- If *Combined Project* has changed files in *Upstream*, how can you then merge the changes and any new changes made in *Upstream*?

If you are maintaining a codebase or configuration files that are mostly *also* maintained elsewhere, Not-forking could be the answer for you. Not-forking was designed to be part of a build tool, and can remove a lot of build system complexity.

Not-forking **avoids project-level forking** by largely automating change management in ways that version control systems such as Fossil, Git, or GitHub cannot. The full documentation goes into much more detail than this overview.

Referring to Diagram 1, the developer now has good reasons to separate *Upstream* project code from its repository and maintain it within the *Combined Project* tree, because in the short term it is just simpler. But that brings the very big problem of the Reluctant Project Fork. A Reluctant Project Fork, or "vendoring" as the Debian Project calls it, is where *Combined Project's* version of *Upstream* starts to drift from the original *Upstream*.

Nobody wants to maintain code that is currently being maintained by its original authors, but it can become complicated to avoid that. Not-forking makes this a much easier problem to solve.

Not-forking produces a buildable tree from inputs that would otherwise need manual merging, or an algorithm so specific that it would become its own project. Rather than adding intelligence to a diff tool, Not-forking gets trees in a condition where diff will work. To do that it needs some guidance from a config file. Of course, at times there will be a merge conflict that requires human intervention, and since Not-forking uses all the ordinary VCS and diff tools, that is a normal merge resolution process. Not-forking understands and can compare many different human-readable styles of version numbering, and is able to monitor and pull from all kinds of upstreams.

Not-forking also addresses more complicated scenarios, such as when two unrelated projects are upstream of *Combined Project*:

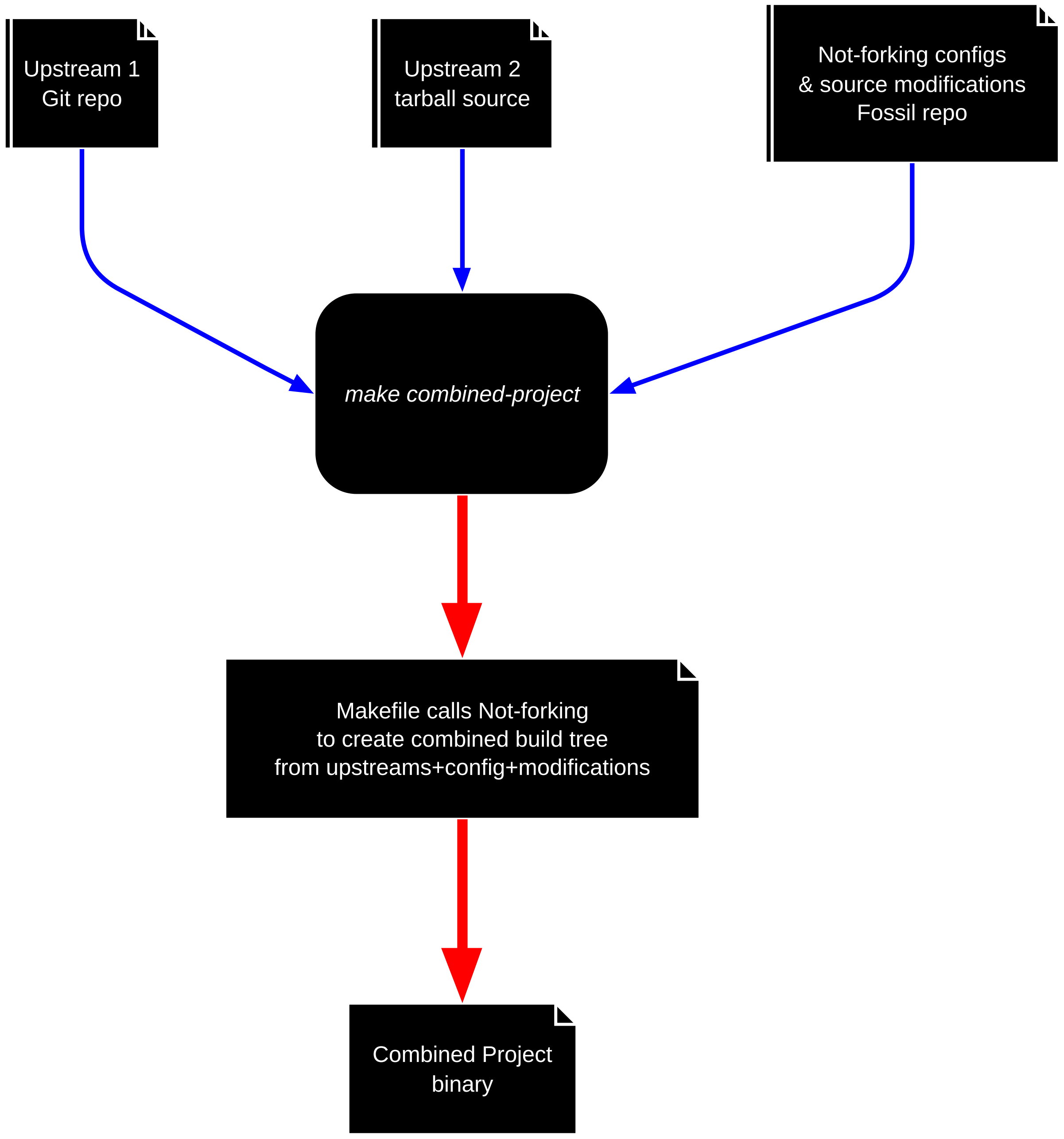

Diagram 2: Not-forking With Two Upstreams

In more detail, the problem of project forking includes these cases:

- Tracking multiple upstreams, each with a different release schedule and version control system. Manual merging is difficult, but failing to merge or only occasionally merging will often result in a hard fork. LumoSQL tracks three upstreams that differ in all these ways

- Tracking an upstream to which you wish to make changes that are not mergable. Without Not-forking a manual merge is the only option even if there is only one upstream and even if the patch set is not complicated. An obvious case of this is replacing, deleting or creating whole files
- Vendoring, where a package copies a library or module into its own tree, avoiding the versioning problems that arise when using system-provided libraries. This then becomes a standalone fork until the next copy is done, which often involves a manual porting task. Not-forking can stop this problem arising at all
- Vendoring with version control, for example some of the 132 forks of LibVNC on GitHub are for maintained, shipping products which are up to hundreds of commits behind the original. Seemingly they are manually synced with the original every year or two, but potentially Not-forking could remove most of this manual work

The following diagram indicates how even more complex scenarios are managed with Not-forking. Any of the version control systems could be swapped with any other, and production use of Not-forking today handles up to 50 versions of three upstreams with ease.

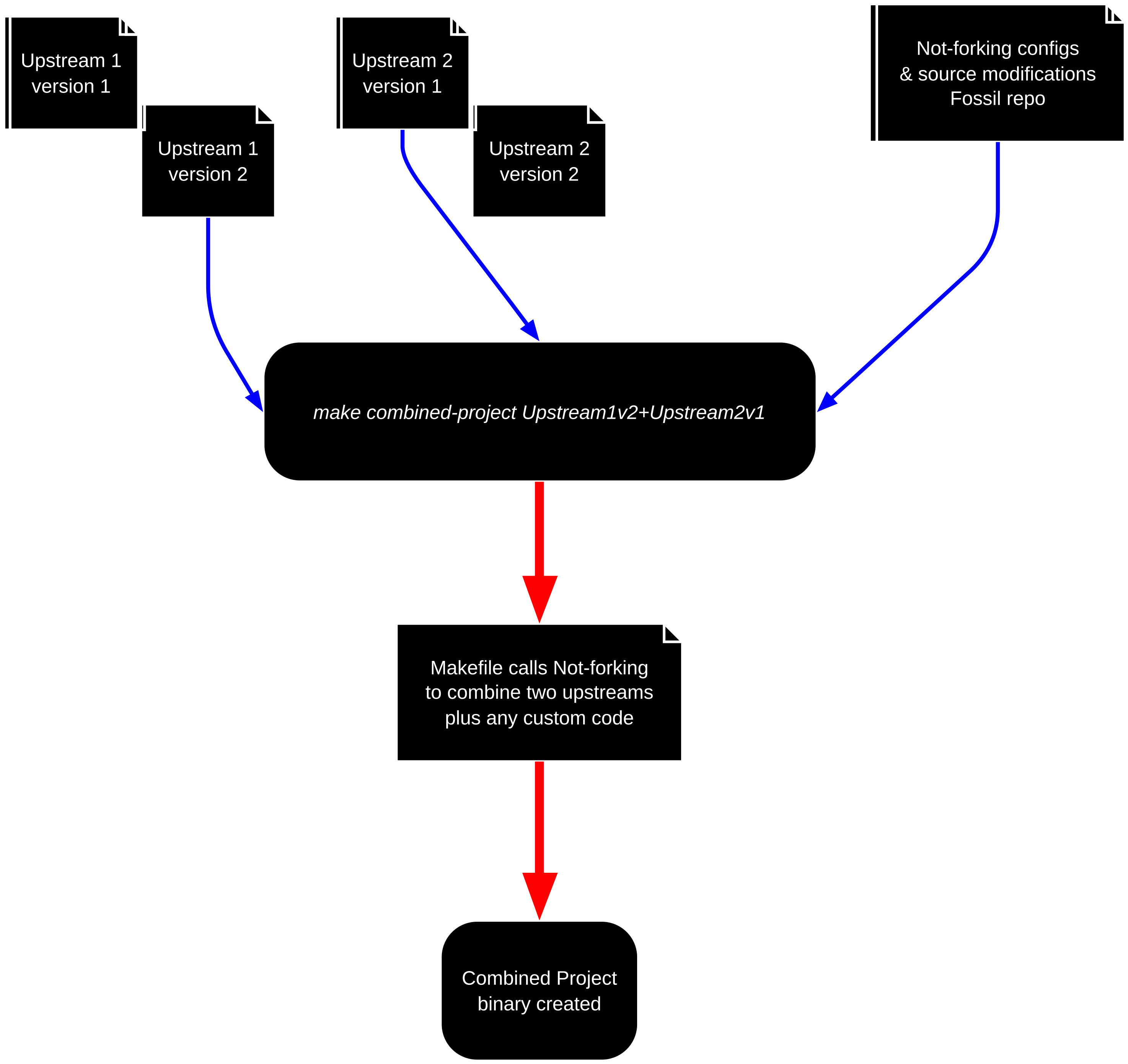
Upstream 1 version 1
Upstream 1 version 2
Upstream 2 version 1
Upstream 2 version 2
Not-forking configs & source modifications Fossil repo
make combined-project Upstream1v2+Upstream2v1
Makefile calls Not-forking to combine two upstreams plus any custom code
Combined Project binary created

Diagram 3: Not-forking With Multiple Versions and Multiple Upstreams

Download and Install Not-forking

You can download Not-forking via wget, Fossil, or git.

If you are reading this on Github, you are looking at a read-only mirror. The official home is the Fossil repository, and that is the best way to contribute and interact with the community. You may raise PRs on Github, but they will end up being pushed through Fossil anyway.

Here you will find the tool and its libraries, and the full documentation. It also contains an example configuration (in directory doc/examples) which can be used for testing. As of version 0.4, directory lib/NotFork/not-fork.d contains example configuration for the tool itself, used to find a specific version even if different from the one installed.

Perl Modules

The default Perl installation on Debian/Ubuntu is perl-base, and on Fedora/Red Hat is perl-core. These have nearly all the Perl modules required except for Text::Glob.

For example, on a Debian or Ubuntu system, as root type:

```
# apt install libtext-glob-perl
```

Or on a Fedora/Red Hat system, as root type:

```
# dnf install perl-text-glob
```

Or on a Gentoo system, as root type:

```
emerge --ask dev-perl/Text-Glob
```

On FreeBSD:

```
pkg install perl5 p5-Text-Glob
# for the complete list of recommended programs to access source repositories:
pkg install fossil perl5 git p5-Git-Wrapper curl p5-Text-Glob patch
```

On NetBSD:

```
pkg_add p5-Text-Glob
# for the complete list of recommended programs to access source repositories:
pkg_add p5-Text-Glob patch fossil git
```

On OpenBSD:

```
pkg_add p5-Text-Glob
# for the complete list of recommended programs to access source repositories:
pkg_add p5-Text-Glob gpatch bzip2 fossil git
```

On minimal operating systems such as often used with Docker there is just a basic Perl package present. You will need to add other modules including ExtUtils::MakeMaker, Digest::SHA, Perl::Git, File::Path and Perl::FindBin .

Not-forking will inform you of any missing Perl modules.

Download and Install

To download Not-forking, you can use `fossil clone` or `git clone`, or, to download with wget:

```
wget -O- https://lumosql.org/src/not-forking/tarball/trunk/Not-forking-trunk.tar.gz | tar -zxf -
cd Not-forking-trunk
```

Once you have downloaded the Not-forking source, you can install it using:

```
perl Makefile.PL
make
sudo make install       # You need root for this step, via sudo or otherwise
```

If you are on a minimal operating system you may be missing some Perl modules as decsribed above. The command

```
perl Makefile.PL
```

will fail with a helpful message if you are missing modules needed to build Not-forking. Once you have satisfied the Not-forking *build* dependencies, you can check that Not-forking has everything it could possibly need by typing:

```
not-fork --check-recommend
```

and fixing anything reported as missing, or which is too old in cases where that matters.

At which point the `not-fork` command is installed in the system and its required modules are available where your perl installation expects to find them.

You might be wondering about runtime dependencies. That is covered in the full documentation, but in brief, not-fork knows what is needed for each of many different scenarios and it does not need to be addressed now. That means you don't need to worry about what not-fork might be used for when you install it. Currently not-fork can use access methods including Git, Fossil, wget/tar and ftp. Modules will likely be added, for example for Mercurial.

It is also possible to use the tool without installing by using the following commands:

```
perl Makefile.PL
make
perl -Iblib/lib bin/not-fork [options] ...
```

To try the tools using the included example configuration, use:

```
perl -Iblib/lib bin/not-fork -idoc/examples [other_options] ...
```

Why Not Just Use Git/Fossil/Other VCS?

Git `rebase` cannot solve the Not-forking problem space. Neither can Git submodules. Nor Fossil's `merge`, nor the `quilt` approach to combining patches.

A VCS cannot address the Not-forking class of problems because the decisions required are typically made by humans doing a port or reimplementation where multiple upstreams need to be combined. A patch stream can't describe what needs to be done, so automating this requires a tangle of fragile one-off code. Not-forking makes it possible to write a build system without these code tangles.

Examples of the sorts of actions Not-forking can take:

- check for new versions of all upstreams, doing comparisons of the human-readable release numbers/letters rather than repo checkins or tags, where human-readable version numbers vary widely in their construction
- replace foo.c with bar.c in all cases (perhaps because we want to replace a library that has an identical API with a safer implementation)
- apply this patch to main.c of Upstream 0, but only in the case where we are also pulling in upstream1.c, but not if we are also using upstream2.c
- apply these non-patch changes to Upstream 0 main.c in the style of `sed` rather than `patch`, making it possible to merge trees that a VCS says are unmergable
- build with upstream1.c version 2, and upstream3.c version 3, both of which are ported to upstream 0's main.c version 5
- track changes in all upstreams, which may use arbitrary release mechanisms (Git, tarball, Fossil, other)
- cache all versions of all upstreams, so that a build system can step through a large matrix of versions of code quickly, perhaps for test/benchmark

Disambiguation of "Fork"

The term "fork" has several meanings. Not-forking is addressing only one meaning: when source code maintained *by other people elsewhere* is modified *by you locally*. This creates the problem of how to maintain your modifications without also maintaining the entire original codebase.

Not-forking is not intended for permanent whole-project forks. These tend to be large and rare events, such as when LibreOffice split off from OpenOffice.org, or MariaDB from MySQL. These were expected, planned and managed project forks.

Not-forking is not intended for extreme vendoring either, as in the case decided by Debian in January 2021, where the up stream is giant and well-funded and guarantees it will maintain all of its own upstreams.

Not-forking is strictly about unintentional/reluctant whole-project forks, or ordinary-scale vendoring.

Here are some other meanings for the word "fork" that are nothing to do with Not-forking:

- In Fossil, a "fork" can be a point where a linear branch of development splits into two linear branches which have the same name. Fossil has a discussion on forking/branching .
- in Git, a "fork" is just another clone of the repository.
- GitHub uses the same definition as Git. As well as providing tools to identify and re-import changes made in the new clone, GitHub promotes forking repositories. As a result it is common for a project on GitHub to have dozens of forks/clones, and for a popular project there can be hundreds.