# Computing Activity Protocol

April 13, 2018

**Problem 1.** Only using standard `git` isn't elegant for managing and versioning binary files in addition to text-based files. For binary files, `diff`s most often don't make sense and there's no general behaviour or rules for how they would be merged. If there's no reasonable way to merge, the files have to be stored whole. Since binary files are usually way larger than pure source code files, saving them as whole blobs for each commit enourmously increases the size of the core git repository.
The solution must involve the storage of a set of binary files (whole chunks, no diffs) for each commit that produced new ones, outside the core git repository, but still associatable with each commit.
What I specifically want at this point is to export a pdf (e.g. current version from latex file), and misc. exported figures (pdf or png) for different versions of a program.
This would be great for collaboration, since if the binary files aren't integral to the git repository itself, a supervisor doesn't have to deal with an unprofessionally cluttered up and ever growing git repo containing binary files, but rather just `get`s or `drop`s only the necessary binary files (demo-files, figures, pdfs) to review at will.

**Solution** There exist several approaches to do this:

1. I tried `git-media` first, which turns out to not be developed actively any longer and therefore I abandoned it.
2. `git-annex` seems to be promising, and that's what I've implemented.

`git-annex` is a decentralized system, where binary files can be added to a commit. Those binaries are then stored in a special subfolder somewhere in `.git/`, and symlinks are automatically put at the right positions in the working tree. Core `git` only tracks and checks out the symlinks (those are usually text files with a hash of the associated binary file).
I set it up with one client (non-bare repo) and one server (bare repo), where the client pushes changes to the server and syncs the binary files (`rsync` by default, and over ssh, in this case is `rsync` not a special remote) with a call to `git annex sync --content`.
Details can be found in the walkthrough on the official `git-annex` webpage `http://git-annex.branchable.com/walkthrough/`.

**Problem 2.** There are differences between a plain ol' `remote` and a `git-annex` special remote. Special remotes are places, where the contents of the binary files are stored (optionally encrypted), while git annex can still operate on them much as it would on any other remote. Such special remotes support also nice integration with a whole variety with widely used web services such as Dropbox, Amazon S3, or your own ssh rsync server like. Using a special remote is sensible to do if your core git repo is stored on GitHub, which doesn't yet support `git-annex`. ("Once you've initialized a special remote in one repository, you can enable use of the same special remote in other clones of the repository."[1])

[1] `http://git-annex.branchable.com/walkthrough/#index12h2`