Distributed file management using git-annex

Christoph Piechula – December 11, 2014

Hochschule Augsburg – University of Applied Sciences,
`christoph.piechula@hs-augsburg.de`

Abstract. Today cloud storage services are all over the place. There is dropbox, box.com, google cloud and many other storage services. Important files also often gets backuped to a local hard drive or network attached device, to ensure access if network connection is not available. It is often hard for the user to keep track of changes and update backups. Sensible personal data also often gets just saved to untrusted cloud storage services and users often dont even know about possible issues or that there is the possibility to encrypt data. A tool named git-annex is a possible solution to this problem. It also addresses other issues with cloud security and data integrity in general. This paper intends to give a overview about git-annex in general. It also intends to demonstrate the command line usage and explaining git-annex' concepts and features.

Keywords: git, annex, distributed filemanagement, repository, remote, backup, storage, encryption

1 Introduction

The demand on data access is growing day by day. Today a lot of users have multiple devices like a PC, a laptop, a tablet pc or a smart phone. Usually synchronisation with cloud storage services is preformed to keep data up-to-date on all devices. As devices are not inevitably connected directly with each other, a cloud storage service is used as a proxy. Cloud services are also a comfortable way to share e.g. photos with friends or sensible documents with co-workers. Storage services are also often used as a backup location for private or commercial data. Widely used cloud storage service providers are Dropbox, Box.com, Google Drive, Apple iCloud and Microsoft OneDrive, see [Reference 1].

But there are known issues with cloud storage services which may cause data loss [Reference 2] or unauthorized access [Reference 3] to private data. Data also might be unaccessible when a provider gets taken down by the FBI because of piracy allegations, see [Reference 4]. Another point is that the user doesn't know if his private or business data gets encrypted properly, even if this is often claimed by cloud storage service providers. Also there might be a privacy issues because of PRISM, see [Reference 5]. In general, there is a problem with most of the cloud storage service providers, that used tools and methods are not transparent at all.

To keep data nevertheless accessible and secure the use of backups and encryption is advisable. When backing up data, one should always bear in mind that hardware is also very error prone, see [Reference 6] or [Reference 7].

Just to sum up, doing a backup is not enough to keep data safe and secured. It is also necessary to check the data's integrity (checking if data is unchanged and valid). Therefore a backup and restore strategy, to validate the integrity of data, is needed.

There are numerous ways to check or ensure data integrity. One possibility would be to use file systems like `ZFS`[1] or `BTRFS`[2] which by itself ensure that data doesn't get corrupted. These file systems have the ability to detect and avoid data corruption, see [Reference 8]. Another possibility to identify data corruption is by checksumming the data itself, see [Reference 9].

To keep the data secured and avoid unauthorized access a proven encryption standard should be used, see [Reference 10].

To implement above backup strategy usually different tools are needed. Another concept that also intends to deal with the issues above is `git-annex`. It is a tool for file synchronisation and archival purposes. This paper aims to give a introduction to the `git-annex` basics and its core features.

2 Git-annex overview

2.1 Project details

`Git-annex` is a tool that aims to synchronize one's data while addressing all the issues that automatically come with cloud storage services and hardware.

`Git-annex` was started as a crowd funding project by Joey Hess, a former Debian GNU/Linux developer. Targeting the project at $3.000 it reached over $20.000 within a few hours. `Git-annex` is written in Haskell, a functional programming language. It has been written in Haskell and not as a *bash script*-tool as Joey Hess wanted to have a decent and stable tool with a decent test suite.

`Git-annex` is a free and open source software tool on top of git. Git is a *distributed* version control system used by developers to mange source code, for more details see [Reference 11].

However, git is not suitable for big binary data as it is calculating differences for all file changes and storing the complete history of a file that changes. This allows the developer to „rollback" the source code state to a previous version. Doing this with big binary data, the managed files and the created file differences would becomes very large and unusable.

[1] A combined file system and logical volume manager designed by Sun Microsystems
[2] „B-tree FS" A copy-on-write linux file system with similar features to zfs

`Git-annex` addresses this problem by only managing the file's metadata within `git`. While this might seem paradoxical for the first moment, is has a lot of advantages with files which cannot be handled by `git` very well. Primary `git-annex` is developed as a command line tool. In the meantime there is also a user-friendly web-based GUI that allows the non-technical user to use `git-annex` for file synchronization in a comfortable way like using e.g. the Dropbox client.

The `git annex`-tool is available for all major operating systems like *linux*, *bsd family*, *mac os*, *android* and *windows*.

3 How it works

3.1 Distributed file synchronisation concept

The `git-annex` synchronisation concept differs from the usual centralized synchronisation concept in many ways. The „old-fashioned" concept usually synchronized one or more devices to a cloud service, often there is only a peer (device) to peer (cloud) synchronization possible. Additionally there also might be a local backup if the user does backups.

To get a idea how a `git-annex`-based setup might look like compared to the usual centralized way, see Fig. 1.

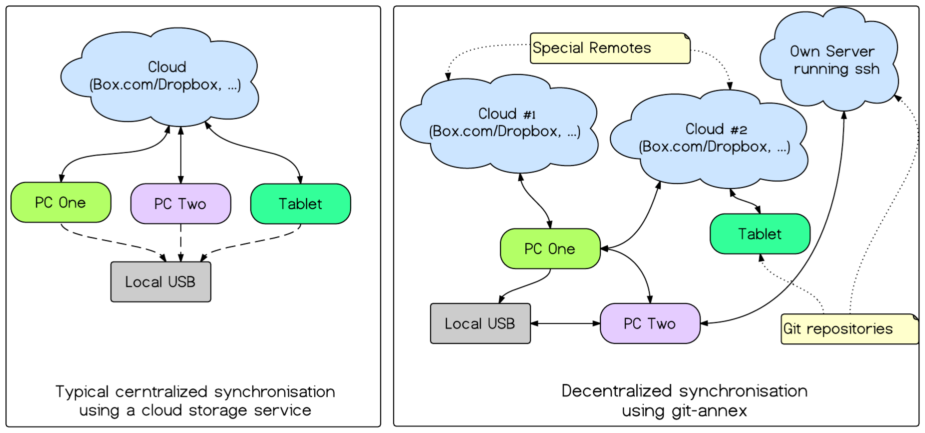


Fig. 1. On the left there is the typical synchronization scenario involving a cloud storage service. On the right you see a typical distributed git-annex synchronization setup.

Alternatives: There are also some other tools like e. g. git-media [Reference 12], git-bigfiles [Reference 13] or git-fat [Reference 14] which extend the capabilities of `git`. Besides those tools extend the `git`-concept of using `git` in conjunction with big files, they follow different approaches to solve different problems. The `git-annex` concept is in its way unique, as the other projects itself. For more details, see *what git-annex is not* [Reference 15].

3.2 Git annex internals

The `git-annex` concept is based on usual `git` repositories. Metadata is synchronized between repositories. A repository is the place where a copy of your meta data is stored. As `git` works in a distributed way, `git-annex` is able to share files across different repositories also in a distributed way, compared to the usual way using cloud storage provider, where usually a centralized approach is used. For more information about `git` see, [Reference 11].

Each `git`-repository is containing a `.git`-folder which contains information about the repository like, available remotes, user identity and so on. For a complete list of the content stored in a `.git`-repository, see [Reference 16]. `Git-annex` extends this folder structure by adding a `annex` folder to it. In this location `git-annex` related information is hold.

When adding a file to `git-annex`, the file gets moved to the `.git/annex/objects/`-folder and a symbolic link to the data is created by default. This is a `git-annex` feature to avoid unwanted manipulation of the data. The data moved to the *objects*-folder is renamed, so that it's name now represents the cryptographic hash sum of the data itself. The files are stored by the *backend* in a key-value manner. In this way the data's integrity might be validated by the `git-annex fsck`-command. For a complete list of supported backends see [Reference 17]. The „symbolic link"-behavior is called `git annex indirect`-mode. To can get more information about the configuration of a repository you can run the `git annex info` command inside the repository.

```
$ git annex info
repository mode: indirect
trusted repositories: 0
semitrusted repositories: 2
        00000000-0000-0000-0000-000000000001 -- web
        d9b74881-8579-48ef-8964-9fc445f2978d -- myrepo [here]
untrusted repositories: 0
transfers in progress: none
available local disk space: 56.52 gigabytes (+1 megabyte reserved)
local annex keys: 1
local annex size: 5 bytes
annexed files in working tree: 1
size of annexed files in working tree: 5 bytes
bloom filter size: 16 mebibytes (0% full)
```

```
    backend usage:
            SHA256E: 2
```

As the symbolic link approach is not always a wanted behavior, there is also the `git annex direct`-mode. This mode is also used when symbolic links are not supported by the file system, this is the case when using a file system like the fat file system. The `direct` mode is also used by the `git annex assistant`, see assistant by default.

The repository and remote configuration itself is saved inside the `.git/config` file. This file is provided by `git` and extended by `git-annex` for its purposes. A repository is represented by a *UUID*.[3] A typical `.git/config` file which has been extended with a `annex` section containing a *UUID*:

```
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[annex]
    uuid = 59a11cf0-0c01-4298-bb94-71c6593caa5b
    version = 5
[remote "temp"]
    url = /tmp/gannex
    fetch = +refs/heads/*:refs/remotes/temp/*
    annex-uuid = b6e1928e-af46-4478-88e1-d17882fdc9f7
```

4 Introduction to git annex usage

As `git-annex` is primary a command line tool written by developers for developers, it is primarily suitable for power users. But in meantime there is also a web-based GUI tool called git annex webapp. It is a part of the git annex assistant. This chapter gives a short introduction to the way `git-annex` might be used. Basic usage commands are demonstrated only. For a complete list of command and possibilities of `git-annex` see the manpage (`man git annex`) or type `git annex` inside the repository to get a list of all possible commands and options. The tested `git-annex` version is `git-annex version: 5.20141105-g8b19598`.

4.1 Commandline usage

Creating a git-annex repository: To initialize a `git-annex`-repository first you have to initialize a `git`-repository. The complete procedure looks as the following shell session demonstrates:

[3] A universally unique identifier (UUID) is an automatically generated unique identifier which is often used on computer platforms.

```
$ git init 'repositoryname'
$ cd 'repositoryname'
$ git annex init 'annex-repository-name'
```

The created `git`-repository now may be modified using the usual `git`-commands. Adding e.g. a remote repository for synchronization can be achieved by running the `git remote add <remote-name> <remote>` command.

Adding files to a git-annex repository: All data manipulation commands are related to the `git-annex`-command set. A complete add-procedure looks as the following example shows:

```
$ git annex add filename.ext
$ git commit -m 'file added.'
```

Synchronizing repository with remotes: After adding files to a repository the metadata is updated with the „connected" repositories (remotes) by pulling changes from a specific remote and pushing current changes to the remote.

```
$ git pull <remote-name>
$ git push <remote-name>
```

This is the way synchronization is realized when using `git`. By default only meta-data is synchronized. To get the real content the `git annex get <filename>`-command has to be run on the specific file. Wildcards are also supported.

```
$ git annex get <file-name.ext>
```

There is also a more fine grained usage possible. For a more complete introduction guide see *git annex walkthrough* [Reference 18].

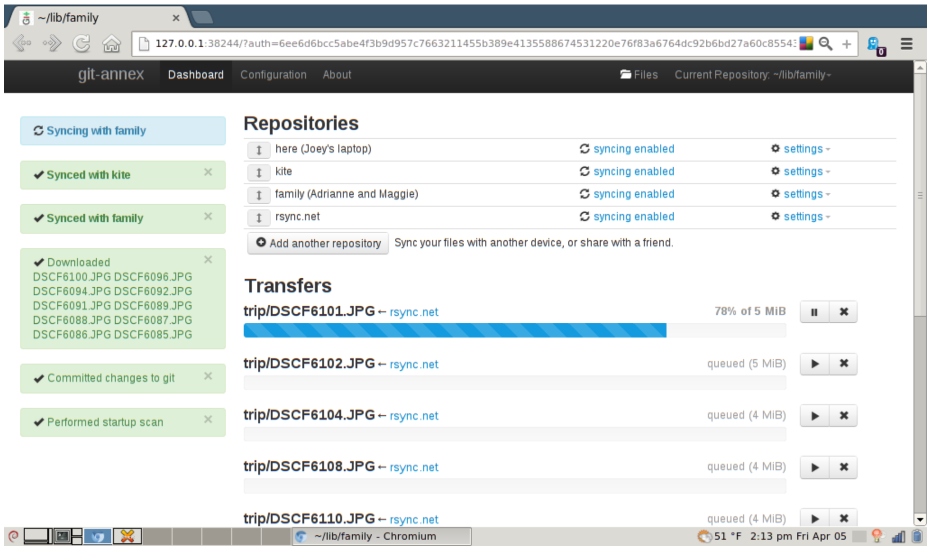
Committing files on every change might seem very uncomfortable, because there is a monitoring daemon included in `git annex`. To automate the complete procedure the daemon can be started by running `git annex watch`. Now all files copied to the directory are added and committed to automatically.

4.2 Git-annex assistant

The assistant tool allows `git-annex` to implement a completely automated synchronization, similar to the way like Dropbox client works. The `git-annex` assistant can be started with the `git annex assistant --auto`-command. By default only meta data is synchronized. Applying the `--content` option to the git annex assistant client will tell `git-annex` to synchronize the content too.

4.3 Git-annex webapp

The webapp of `git-annex` is a part of the `git annex assistant`. Running the command `git annex webapp` starts a `git-annex`-service listening on localhost.


~/lib/family
127.0.0.1:38244/?auth=6ee6d6bcc5abe4f3b9d957c7663211455b389e4135588674531220e76f83a6764dc92b6bd27a60c8554:
git-annex
Dashboard
Configuration
About
Files
Current Repository: ~/lib/family
Add another repository
~/lib/family - Chromium
51 °F 2:13 pm Fri Apr 05

Syncing with family

Synced with kite

Synced with family

Downloaded
DSCF6100.JPG DSCF6096.JPG
DSCF6094.JPG DSCF6092.JPG
DSCF6091.JPG DSCF6089.JPG
DSCF6088.JPG DSCF6087.JPG
DSCF6086.JPG DSCF6085.JPG

Committed changes to git

Performed startup scan

Repositories

here (Joey's laptop)	syncing enabled	settings
kite	syncing enabled	settings
family (Adrianne and Maggie)	syncing enabled	settings
rsync.net	syncing enabled	settings

Sync your files with another device, or share with a friend.

Transfers

trip/DSCF6101.JPG ← rsync.net	78% of 5 MiB
trip/DSCF6102.JPG ← rsync.net	queued (5 MiB)
trip/DSCF6104.JPG ← rsync.net	queued (4 MiB)
trip/DSCF6108.JPG ← rsync.net	queued (4 MiB)
trip/DSCF6110.JPG ← rsync.net	queued (4 MiB)

Fig. 2. Git annex webapp syncing files.

The webapp allows the configuration and management of repositories and remotes in a user friendly way. Fig. 2 shows the `git annex webapp`. By using the webapp, the user has the possibility to setup `git-annex` repositories completely command line free. It is also possible to configure special remotes and use features like file encryption. Another purpose is also to provide a „Dropbox like" user experience.

The webapp GUI has also a interactive pairing feature for clients in the local network. Beside this feature stuff like periodical file integrity checks or *XMPP*-account for file sharing (synchronisation) with friends or not directly connected devices is possible. Also there is a log file and the option to restart or shutdown the `git-annex` daemon.

5 Git-annex features

5.1 Repository Groups

`Git annex` introduces `repository groups` as a way to specify the behaviour of a repository. Using `repository groups` gives the user more control about the way how a repository behaves. For example if a repository is used as a *backup* repository, all synchronized files get accumulated. There are also `repository groups` for archival purposes. User repositories are usually in the *client* group. Repositories used to exchange between not directly connected devices are typically in the *transfer group*. `Repository groups` can be controlled by using the `git annex vicfg`-command. For a complete list of standard groups see [Reference 19].

5.2 Special Remotes

`Git-annex` extends the `git`-repository concept by introducing `special remotes`. These remotes can by used like typical `git annex`-repositories, `git`-commands however cannot be used. Usually there are remotes where no `git`-repositories can be managed directly. Typical `special remotes` are cloud storage services like `Amazon S3` or `Box.com`. For a complete list of currently supported special remotes by `git-annex`, see [Reference 20]. Usually there are three *types* of remotes, `special remotes`, `archive remotes` and `local remotes` like USB drives.

As `git-annex` follows a modular approach, there is the possibility to *connect* `git-annex` to yet unsupported `special remotes` by writing a extension. This can be done in three different ways. For more information see [Reference 21].

5.3 Encryption

By default the data saved in a repository is not encrypted. Everyone who has access to the repository, whether it is a `git` repository or a `special remote`, sees the saved content. As data may be saved on untrusted cloud storage services by using `special remotes`, `git-annex` support three different encryption concepts to ensure privacy.

Shared encryption is the most simple way to implement encryption when using `git-annex`. Using this encryption concept a *symmetric key* is generated and saved inside the user's repository. This is only a save approach if the user can trust all clones of all repositories, as there is a copy of the encryption key to encrypt all data which is saved on a `special remote`.

Hybrid encryption is the *recommended* way to encrypt when synchronizing data with untrusted `special remotes`. When using *hybrid encryption*, the symmetric encryption key is additionally encrypted with a *gpg*[4] public key of a user, for more information see public-key cryptography [Reference 10]. This approach allows `git-annex` to share a encrypted repository with different users without sharing a common passphrase.

Public key encryption is the third method that may by used to encrypt your data. For details about *public key encryption* see, [Reference 10].

[4] GNU Privacy Guard, a free tool that implements public key cryptography

When data is saved in a `git`-remote, there is also the possibility to encrypt the `git`-remote using gcrypt.[5]

5.4 Communication

For communication between `git-annex` repositories usually `ssh` (secure shell) is used. File transfer is realized with the `rsync`-tool.[6] The `XMPP`-protocol is used to communicate with repositories if a direct connection is not possible, e.g. because of firewall restrictions.

There is also a local UDP based paring mechanism for the local network, this mechanism is only used by the `git annex webapp` as it needs user interaction to exchange a secret key to pair repositories with each other.

5.5 Misc features

Preferred content: This option gives the user the ability to configure which content to be managed within a repository by *including* or *excluding* different filetypes. For more information, see [Reference 22].

Minimum number of copies: `Git-annex` can be configured to „backup" your files by always ensuring that a *minimum* number of copies is preserved. This can be done via command line by running:

```
$ git annex numcopies <number>
```

There is also the possibility to set this option by creating a hidden `.gitattributes` file inside the `git-annex`-repository. The following example tells `git-annex` to preserve three copies of *mp3*-files:

```
$ echo "*.mp3 annex.numcopies=3" >> .gitattributes
```

Trustiness levels: `Git-annex` supports different levels of *trustness*. By default repositories are set to *semitrusted*. If a repository is *semitrusted* `git-annex` checks if the minimum needed number of file copies is present when a file is dropped by the user. Trusted repositories are assumed to be save and therefore no check is done.

Data integrity: By default files are managed in a key-value backend manner using a cryptographic hash sum. There is also the possibility to validate the data integrity by calculating the hash sum. This can be done by running the `git annex fsck [<filename>]` command.

[5] Encryption tool for git repositories, https://github.com/bluss/git-remote-gcrypt
[6] A widely-used copy tool, http://en.wikipedia.org/rsync

Collision handling: `Git-annex` also implements collision handling by adding a suffix to a file which would collide with another file that has the same name.

Location tracking: Another useful feature which makes `git-annex` a pleasure to use is *location tracking*. `Git-annex` by itself is able to tell you where your files are stored. By running

```
$ git annex whereis
whereis 01.jpg (2 copies)
    32bef0e7-1e0f-478f-8b33-1718daa67502 -- [usbdrive]
    a1c3bc37-4c78-420b-a83f-9504c195b8e7 -- [box.com]
ok
whereis evidence.png (3 copies)
    32bef0e7-1e0f-478f-8b33-1718daa67502 -- [usbdrive]
    a1c3bc37-4c78-420b-a83f-9504c195b8e7 -- [box.com]
    ac61251d-fb87-467e-9852-1a61cf414bcb -- athome [here]
```

`git-annex` tells you, where your files are stored and how much copies are present across all repositories.

JSON Interface: This is a `git-annex` option which makes the command line tool suitable for scripting tasks. Adding the `--json` option to a command returns a well formatted *JSON*-document which e.g. might be easily parsed by a python script.

```
$ git annex info --json
{
    "command":"info",
    "repository mode":"indirect",
    "trusted repositories":[],
    "semitrusted repositories":[
        {"uuid":"00000000-0000-0000-0000-000000000001",
            "description":"web","here":false},
        {"uuid":"d9b74881-8579-48ef-8964-9fc445f2978d",
            "description":"myrepo","here":true}
        ],
    "untrusted repositories":[],
    "available local disk space":"56.52 gigabytes (+1 megabyte reserved)",
    "local annex keys":1,
    "local annex size":"5 bytes",
    "annexed files in working tree":1,
    "size of annexed files in working tree":"5 bytes",
    "bloom filter size":"16 mebibytes (0% full)",
    "success":true
}
```

Testsuite: `Git-annex` implements a `git annex test`-command to run all available test cases for `git-annex`. There is also a `git annex testremote`-command which makes it possible to test a remote repository. When running this command a lot of `git-annex`-commands are executed to verify that the `git-annex`-remote is working correctly. This is also a nice feature to test a self implemented `special remote`-module.

5.6 Conclusion

`Git-annex` is by no means not only a tool for *unix hackers* and *power users*. There has also been a lot of development done yet to make `git-annex` more user friendly by introducing the git annex webapp.

The `git-annex` command line utility is very mighty and well structured. It implements over 70 different commands grouped in command subcategories like „commonly used“, „repository setup”, „repository maintenance“, „query”, „metadata“, „utility”, „plumbing" and „testing“. It is not only suitable for manual usage but also for complex scripting tasks by using the *JSON* interface.

But one have to remember, with great power comes great responsibility — so the `git-annex` command line utility is definitely not a tool for a command line or *unix* newcomer. But for this user group, there is the webapp with a user friendly graphical interface, which makes using `git-annex` as easy as other comparable cloud storage service provider tools. Another good thing is that all major operating systems are supported.

There are also some downsides. `Git-annex` is still under heavy development and the mighty command line tool is a high barrier for newcomers.

In the end, the users themselves have to decide which way to go and if the data is important and needs to be secured and validated.

References

[1]J. Linder, “Cloud storage providers: Comparison of features and prices.” http://www.tomshardware.com/reviews/cloud-storage-provider-comparison,3905.html.

[2]E. Bott, “Dropbox sync glitch results in lost data for some subscribers.” http://www.zdnet.com/dropbox-sync-glitch-results-in-lost-data-for-some-subscribers-700003461.

[3]D. McCullagh, “Dropbox confirms security glitch–no password required.” http://www.cnet.com/news/dropbox-confirms-security-glitch-no-password-required.

[4]D. Coldewey, “Megaupload taken down on piracy allegations.” http://techcrunch.com/2012/01/19/megaupload-taken-down-on-piracy-allegations.

[5]Wikipedia, “PRISM (surveillance program).” http://en.wikipedia.org/w/index.php?title=PRISM_(surveillance_program), 2014.

[6]L. Pustina, “Your hardware will fail – just not the way you expect.” https://blog.codecentric.de/en/2013/11/hardware-will-fail-just-way-expect/.

[7]Wikipedia, “Data degradation.” http://en.wikipedia.org/w/index.php?title=Data_degradation&oldid=634140013, 2014.

[8]Wikipedia, “Data integrity.” http://en.wikipedia.org/w/index.php?title=Data_integrity&oldid=636149302, 2014.

[9]Wikipedia, “Checksum.” http://en.wikipedia.org/w/index.php?title=Checksum&oldid=635529055, 2014.

[10]Wikipedia, “Cryptography.” http://en.wikipedia.org/w/index.php?title=Cryptography&oldid=636931047, 2014.

[11]S. Chacon and B. Straub, “Pro Git.” http://git-scm.com/book/en/v2, 2014.

[12]A. Lebedev, “Github, git-media source code.” https://github.com/alebedev/git-media.

[13]caca labs, “Git-bigfiles, git for big files, source code.” http://caca.zoy.org/wiki/git-bigfiles.

[14]J. Brown, “Github, git-fat source code.” https://github.com/jedbrown/git-fat.

[15]J. Hess, “git-annex special remotes.” http://git-annex.branchable.com/not/.

[16]N. Quaranto, “what’s inside your .git directory.” http://gitready.com/advanced/2009/03/23/whats-inside-your-git-directory.html.

[17]J. Hess, “Backends.” https://git-annex.branchable.com/backends.

[18]J. Hess, “Walkthrough.” https://git-annex.branchable.com/walkthrough/.

[19]J. Hess, “Standard groups.” http://git-annex.branchable.com/preferred_content/standard_groups/.

[20]J. Hess, “git-annex special remotes.” http://git-annex.branchable.com/special_remotes/.

[21]J. Hess, “git-annex special remotes.” http://git-annex.branchable.com/special_remotes/external.

[22]J. Hess, “Preferred content.” http://git-annex.branchable.com/preferred_content/.