Why and how to use distributed version control systems for collaborations in math

A Git Howto for Mathematicians

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Konrad Voelkel (konradvoelkel.com/git-for-math)

based on a talk in May 2015, Freiburg - published online Nov 2015

Intended Audience

UNI FREIBURG

We assume that you know nothing about VCS like SVN or Git.

You might be interested in it if you're planning to write a paper with someone else, or have a book project in mind. For example, if you are currently a PhD student in mathematics.

If you already use software like CVS or Subversion, these slides try to convince you to try out a superior software.

Goal of this talk

UNI FREIBURG

We want to

- Understand the problems
  1. problem 1: loss of data
  2. problem 2: conflicts
- See how not to solve them
- Understand a solution
  - DVCS: distributed version control systems
- Have some fun and learn a practical tool
  - Git

If after these slides you start a *real* Git tutorial, I reached my goal.

Why Should We Care?

UNI FREIBURG

In programming, systems like Git have changed the way people work to a more efficient, more reliable workflow.

Formalized proofs *are* programs, so a human-readable mathematical proof is reasoning about a (hidden) computer program. That (called *Curry-Howard correspondence*) is the abstract reason why Git should also help mathematicians.

There are already reports how Git is helping in the natural sciences[1]. It seems as if investing some time to learn Git might pay off quickly. Why not find out?

---

[1] Karthik Ram 2013, doi:10.1186/1751-0473-8-7

Table of Contents

UNI FREIBURG

Section 1: Working Alone

UNI FREIBURG

We will first concentrate on the situation where 1 person works on a document without others.

The Problem: Loss of data

UNI FREIBURG

If you work alone, your harddisk may crash. That's why you make backups. (You do make backups, right?)

Sometimes you want to remove a section of a document, only to find out later that you wanted to keep something from it. Backups help in these moments.

One way to backup something before removing it is *commenting out text*. This is very convenient, so we generalize:

Ideally, every state of the data is preserved

Solutions to the Problem of Data Loss

UNI FREIBURG

Making frequent backups imposes upon us the burden to decide at which point a backup should be made, and to figure out an organizational scheme to later find a particular backup. This is difficult.

The simple and effective solution instead is complete *version control* - record every “single change”.

Preventing Data Loss: Version Control

UNI FREIBURG

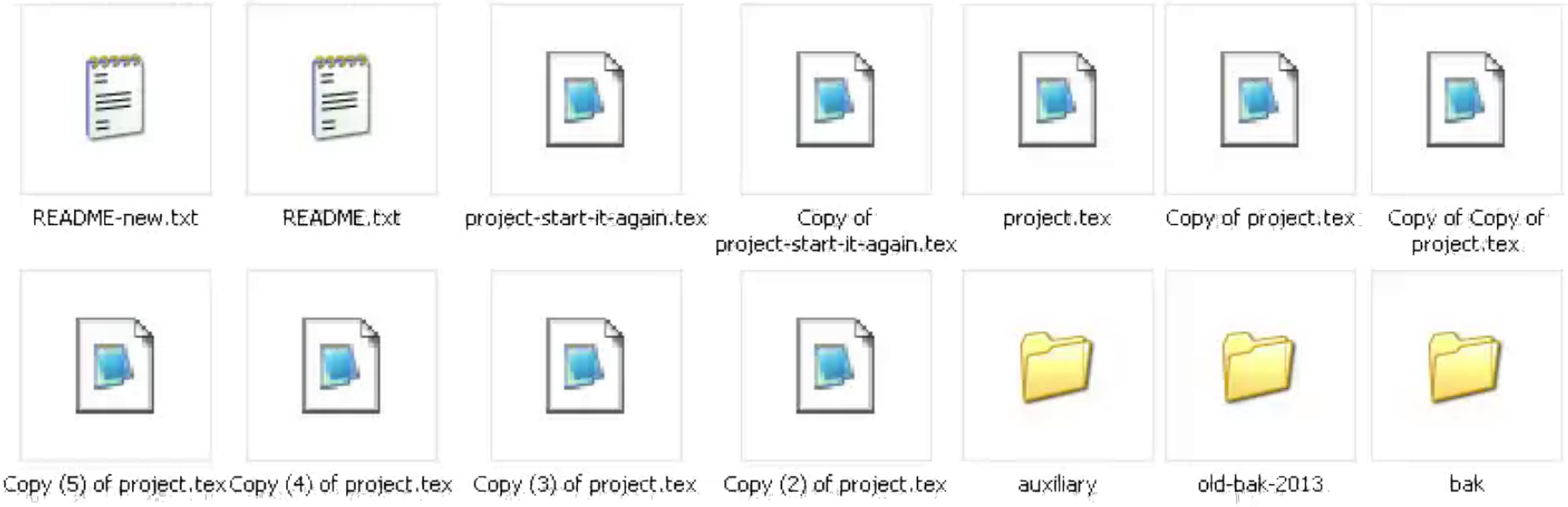
It might seem odd to introduce version control systems as backup solutions, which is strictly only true if one uses some form of *replication* with it.

In our terminology, they are a sophisticated organizational scheme for making backups - which makes very frequent backups convenient.

In fact, version control systems work best with small changes.

We will come back to this point.

A Non-Solution: Many Files



Once you arrive at this point, it becomes clear that something went wrong at the beginning.

Getting Rid of Many Redundant Files

UNI FREIBURG

If you happen to have several files for different versions of the same document, you might want to know what actually changed.

```
$ diff document-version-A.tex document-version-B.tex
```

Yields output similar to this:

```
-An example of text which is changed.
+An example of additional text.
+
+An example of text which is changed, but only a little bit.

-An example of text which is removed entirely.
```

Exercise: reconstruct versions A and B of the text that have this *diff*.

Basic Theory of Version Control Systems (VCS)

UNI FREIBURG

Further reading for the theoretically-inclined:

- O'Sullivan 2009
- Löh-Swierstra-Leijen 2007
  (in particular chapter 3 "Formal model")
- Löh-Swierstra 2014
  "The semantics of version control"

(click on author names to open article)

Some Practical Remarks on Using VCS

UNI FREIBURG

If you want to use VCS effectively and efficiently, you should:

- Write each sentence on a separate line.
- Learn to read and interpret *diffs*.
- In longer documents, use separate files for chapters.
- If you're done with a particular task, commit.

Section 2: Working With Others

UNI FREIBURG

We will now look at collaborations,
where a group writes a document together.

The Problem: Merging

UNI FREIBURG

If 2 or more people write a document together,
the work of each person has to be merged into one document.

Collaboration requires merging.

The Problem: Merging - Sequential Approaches

UNI FREIBURG

One can work *sequentially*, i.e. first person 1 writes something, then person 2 validates it and writes some more, then eventually it gets back to person 1 to validate everything and write again.

This requires a lot of communication to coordinate the times and exchange the data.

It is also really difficult to prevent person 2 from working while person 1 has not finished. In pratice, one always has some form of a *parallel* approach instead.

The Problem: Merging - Parallel Approaches

UNI FREIBURG

One can work parallely, i.e. everyone works at the same time and at some point the stuff written down so far gets exchanged. By some process, consensus is reached upon a common state of data which is then distributed so that the next round of work can begin.

This requires a lot of effort to reach the *consensus*.

If there are *conflicts*, i.e. two people worked differently on the same lines of text, these paragraphs and those depending on them have to be re-written during the merge process.

Linus Torvalds, Inventor of Linux

Linus Torvalds

photo reused under GFDL with permission of Martin Streicher, Editor-in-Chief, LINUXMAG.com

Some History of Linux

UNI FREIBURG

In the beginning, Torvalds got patches to the Linux kernel by email. He did all the merging himself then.

Later, there was a loose “hierarchy” of people Torvalds trusted for subsystems of the kernel.

In particular, they never used a CVS or SVN server!

Some Non-Solutions: Dropbox & friends

UNI FREIBURG

If you’re alone, Dropbox gives you a backup in the cloud, which is better than nothing.

If you’re collaborating, Dropbox has a sharing feature. If you’re working at the same time, it breaks.

That means, you’re forced to work sequentially or in different files. Merging anything is entirely up to you.

Some Non-Solutions: CVS and SVN

UNI FREIBURG

If you're alone, centralized version control systems like CVS and SVN seem like a good idea.

If you're working with others, you will need internet access to your server, and your colleague will see every single change right after you commit it.

So you won't commit half-finished stuff, ever.

And you have to do large merges by hand.

The Real Problem: Merging Large Changesets

UNI FREIBURG

Merging two different changes of the same line of text (conflicts) requires to decide for one of the two versions or re-writing it.

If there are several conflicts at once, solutions in one line of text may depend on solutions in another line of text.

Therefore, the merging person has to keep the whole changeset of both conflicting versions in his mind.

The larger the conflicting changeset, the higher the burden to keep in mind and the higher the cost of possibly re-writing everything affected.

There is no solution other than communication and hard work.

The Solution to the Problem of Merge Conflicts

UNI FREIBURG

Don’t merge large changesets!

Figure out a workflow in which there are only small changesets.

Figure out how to optimize the communication,
which is still necessary.

Distributed Version Control Systems (DVCS)

UNI FREIBURG

If everyone has his own repository, one can commit very often, small changes.

This makes merges much less painful, most of the time almost automatically.

One can use a cloud repo (e.g. GitHub) for backups and synchronization.

Branches are really easy to use.

Practical Solutions

UNI FREIBURG

Now we will look into practical solutions

Linus Torvalds, Inventor of Git

Linus Torvalds

photo reused under GFDL with permission of Martin Streicher, Editor-in-Chief, LINUXMAG.com

Some History of Linux, second time

UNI FREIBURG

After a while maintining the Linux kernel, Torvalds wanted to use version control after all. None of the systems available satisfied his need for speed, so he set out to *do it right*^{TM}

A Particular Implementation: Git

UNI FREIBURG

Git is a DVCS (a distributed version control system).

Git is widely used and really fast.

Fun fact: you can even get a DOI for a Git repo:
`guides.github.com/activities/citable-code/`

A Particular Implementation: Git, its workflow

UNI FREIBURG

With Git, the repository you work on is always 100% yours, on your harddisk.

You may also use some cloud provider for backups.
The originally intended workflow is not to connect to some central repository where others have write-access, although that will also work well, at least with a small number of collaborators.

The intended workflow is the “branch-pull” model, where you develop/write any distinct new “feature” (like a book chapter or reworking of a proof) on a branch that you then offer your collaborators to pull, i.e. merge into their repository.

In theory, there are at least as many branches of a project than people working on it. In practice, one might decide to put together a single branch for publication. There is no need to kill other branches, though.

A Particular Cloud Provider: GitHub

UNI FREIBURG

GitHub is a website where you can have an account and as many publicly readable Git repositories as you like.

If you have a university email address, you can have 3 private repositories, too.

More private repositories cost money.

GitHub offers, additionally to a Git repository (cloud) things like an issue tracker, a wiki, some statistics, a text editor and more.

While an issue tracker is essential for larger software projects and a wiki can be very useful for documentation, most of these tools are not necessary for usual collaborations in mathematics.

An Apology

UNI FREIBURG

While Git and GitHub are nice, they are not the only game in town - just the most popular.

Other DVCS like Git include *Mercurial* and *Darcs*. It is said that Mercurial is easier to learn and Darcs makes more sense mathematically (the documentation is full of talk about quantum patch algebra).

Instead of GitHub, you can also have a look at *BitBucket*. If you want to “host your own cloud”, take a look at *GitLab*. Actually, GitLab might be the best for mathematicians.

Section 3: Getting Your Collaborator to use Git

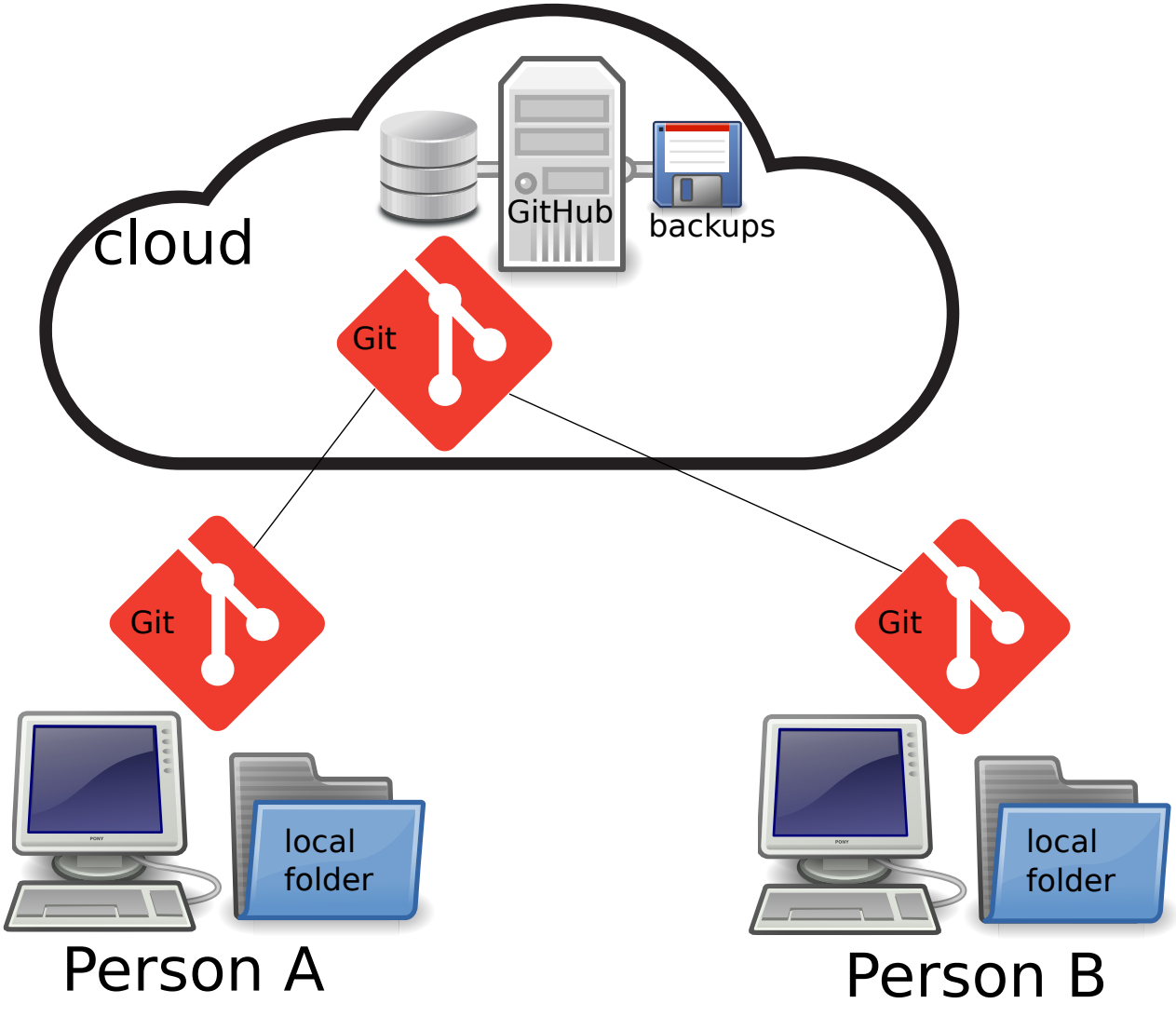
UNI FREIBURG

How to get others to use (D)VCS?

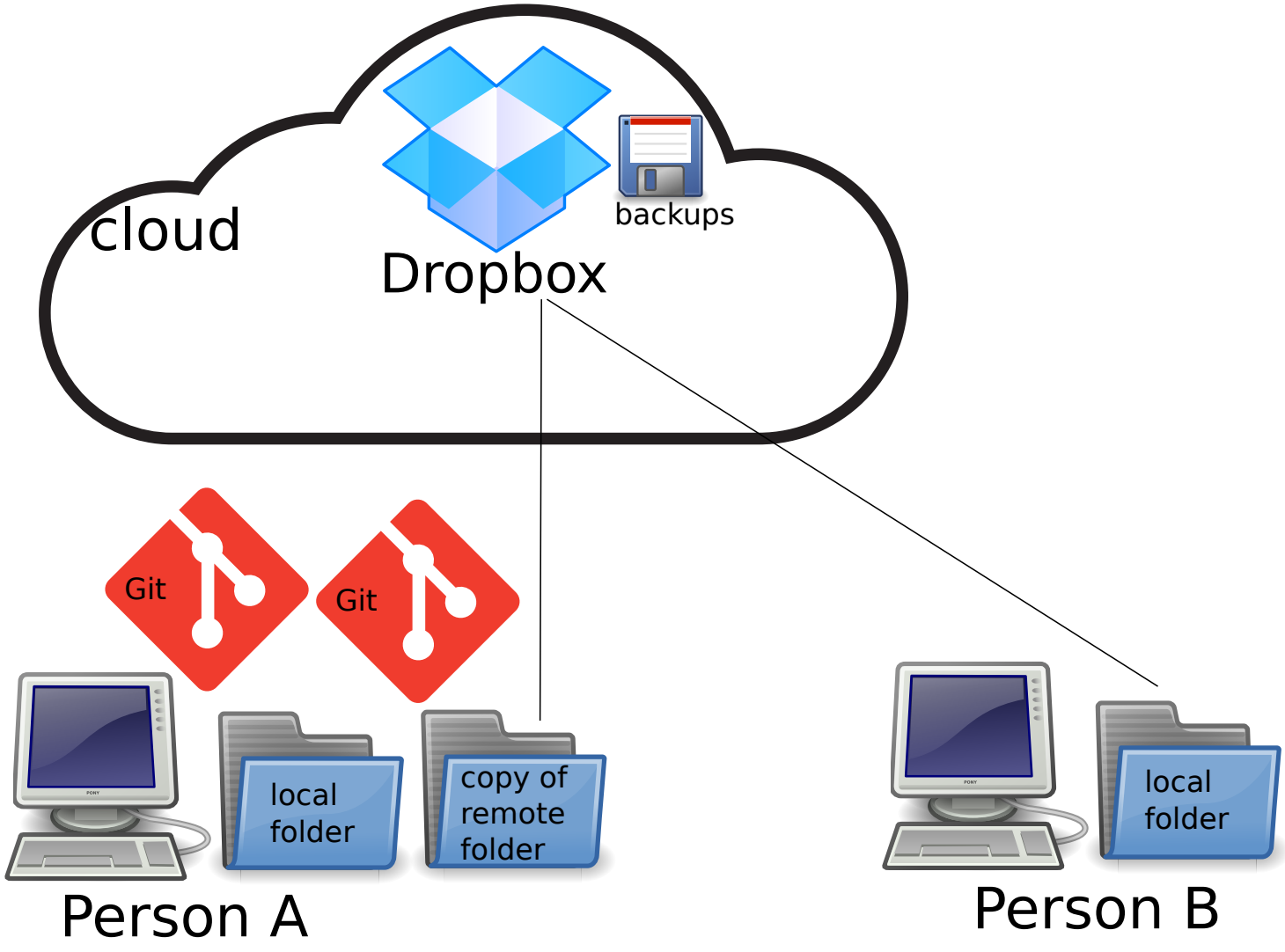
Explain it to them in simple terms.
Then walk them through these slides.

If this doesn't work, you can still use Git while they use something inferior.

Explain Git with a Picture



Use Git while your Colleage is on Dropbox



Section 4: The Mathematics of Git

UNI FREIBURG

Let’s have a look at the structures behind Git.

Not so much mathematics, actually...

Datastructures of Git: the index

UNI FREIBURG

- There is the *folder* of files your Git repository resides in, which has some state (i.e. files with certain content in it).
- There is the *index* of changes from the folder with respect to the last *commit* which were *staged* for the next commit.
- There is the *HEAD* which points to the last commit.

Directly after performing `git init` or `git clone` or `git push` successfully, your folder will contain no unstaged changes, and the index will be empty, that is there are also no staged changes.
If you have unstaged changes in your folder, after performing `git add` or with one of the commands `git mv` or `git rm`, you will have staged changes, i.e. a changed index.
With `git commit` you can commit the index.

Datastructures of Git: repositories

UNI FREIBURG

In your local folder, you have a single Git *repository*, short: *repo*.
In it you can have many branches, one usually called *master*.
If you use a cloud provider like GitHub, there is also a *remote repo*, often called *origin*.

Datastructures of Git: under the hood

UNI FREIBURG

Under the hood, which in Git is called plumbing, in contrast to the porcelain user interface, Git is a content-addressable filesystem, i.e. a key-value store.
Everything is addressed by its *hash*. There are (typed) *blob* objects which are the leaves of *tree* objects. There are *commit* objects which contain certain trees, *reference* objects which point to commits (and behaves like branches) and *tag* objects which are like a reference with metadata. Finally, there are *remote* references, which point to the last commit on a remote repository.

Git as a Graph

UNI FREIBURG

Essentially, Git is a graph with commits as nodes and the Git commands are graph manipulation tools (create/remove nodes).
Creating a branch, that is, creating a reference makes a commit reachable/adressable.
Creating a branch is like making an explicit savegame before you do something potentially damaging with new commits.

A Few Words on Hashes

UNI FREIBURG

Internally, everything in Git is adressed by *hashes*. They look like
`cf23df2207d99a74fbe169e3eba035e633b65d94`
(in the case of SHA-1, used by Git).

If you're coming from other systems, you might expect version numbers. In Git, you have to add version numbers as extra structure, with tags.

Since Git is a graph and version numbers impose a linear order, this makes sense.

Git as a HIT

UNI FREIBURG

See Slides from Dan Licata

`dlicata.web.wesleyan.edu/pubs/l13git/git.pdf`

Patches can be formalized as quotient types in “ordinary” dependent type theory, since two patches created differently might be equal in effect (this has practical consequences)!

If one formalizes entire repositories as higher inductive types (HITs) in homotopy type theory, one has to use fewer axioms (4 instead of 14) and patches fall out naturally.

As a starter, here is a micro-tutorial on using Git

This serves the purpose of showing you the average complexity of dealing with Git, while not being a full tutorial.

Find out what these commands do to the graph:
`www.wei-wang.com/ExplainGitWithD3/`

See also: Git for Scientists: A Tutorial (by John McDonnell)
`nyuccl.org/pages/gittutorial/`

The Most Important Commands

UNI FREIBURG

Try these at home:

`git help $command`	what are the 1001 options?
`git status`	what's going on in this folder?
`git clone $remoterepo`	download remote repo!
`git init`	create a fresh & empty repo here!
`git add $filenames`	add toindex (= stage files).
`git diff`	what has changed?
`git commit`	commit the index.
`git commit -a`	-a = add all changed files first.
`git pull`	fetch changes from remote repo.
`git push`	push (commited) changes to remote repo.

Configuration

UNI FREIBURG

Once you use Git, you will want to configure it:

```
git config -global user.name 'Firstname Lastname'
git config -global user.email 'firstname.lastname@sth'
git config -global color.ui 'auto'
git config -global credential.helper cache
git config -global credential.helper 'cache -timeout=7200'
```

There are more `credential-helper`s that help you save time entering your credentials (username, password for remote repositories).

A micro-workflow

UNI FREIBURG

To begin with, you could do the following:

1. Create a new empty folder somewhere
2. Open a terminal in this folder
3. Run `git init`
4. Create a new file (some tex document)
5. Run `git add $filename`
6. Run `git commit`
7. Enter the commit message (describing the content)
8. Change the file (with your usual text editor)
9. Run `git commit -a`
10. Enter the commit message (a description of the changes)

That works fine when you are alone.

A workflow for collaboration with GitHub

UNI FREIBURG

If want to start a collaborative project, try this:

1. Create a new (possibly private) project on GitHub
2. Run `git clone $repourl` where you want to have the files
3. Change some file (with your usual text editor)
4. Take another look at what you did with `git diff`
5. Run `git commit -a`
6. Enter the commit message (a description of the changes)
7. Run `git pull` to make sure you have the latest changes
8. Eventually merge them with your changes
9. Run `git push`
10. Read the help section on GitHub again ;-)

The Actual Merge Process

UNI FREIBURG

Remember, merging real conflicts is always work.
We just try really hard to avoid conflicts.

To actually merge, it is highly recommended to use some mergetool like *meld*, for example.
Emacs also has good support for merging and Git, the right search keywords are *magit* and *ediff*.

Please expect to invest some time before the tools feel helpful.
You can still do your first merges “by hand”.

Good luck!

UNI FREIBURG

Here are some mantras to take home:

1. Version control takes care of your backup organization
2. The cloud makes your backups reliable and accessible
3. Distributed version control allows for small commits
4. So we can avoid large merges, which are always painful

Hopefully that's enough to get you interested in DVCS!

Recommended reading: Got 15 minutes and want to learn Git?
`try.github.io/levels/1/challenges/1`