



INTRODUCTION TO VERSION CONTROL

Time travel for beginners

June 29, 2018 | Julia Sprenger | INM-6/10



Part I: Why should I care about versions?

"FINAL".doc



FINAL.doc!



FINAL_rev.2.doc



FINAL_rev.6.COMMENTS.doc



FINAL_rev.8.comments5.
CORRECTIONS.doc



JORGE CHAM © 2012

FINAL_rev.18.comments7.
corrections9.MORE.30.doc

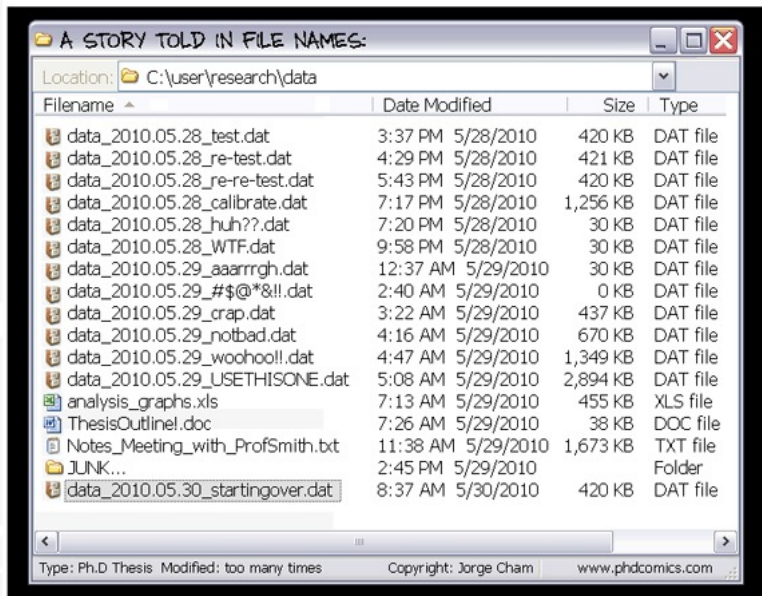


FINAL_rev.22.comments49.
corrections.10. #@\$%WHYDID
ICOMETOGRADSCHOOL????.doc



WWW.PHDCOMICS.COM

<http://phdcomics.com/comics.php?f=1531>



<http://phdcomics.com/comics.php?f=1323>

VERSION CONTROL USING FOLDER AND FILENAMES

- only readable by you ...

VERSION CONTROL USING FOLDER AND FILENAMES

- only readable by you ...
- ... as long as you remember

VERSION CONTROL USING FOLDER AND FILENAMES

- only readable by you ...
- ... as long as you remember
- no consistent structure or naming scheme enforced

VERSION CONTROL USING FOLDER AND FILENAMES

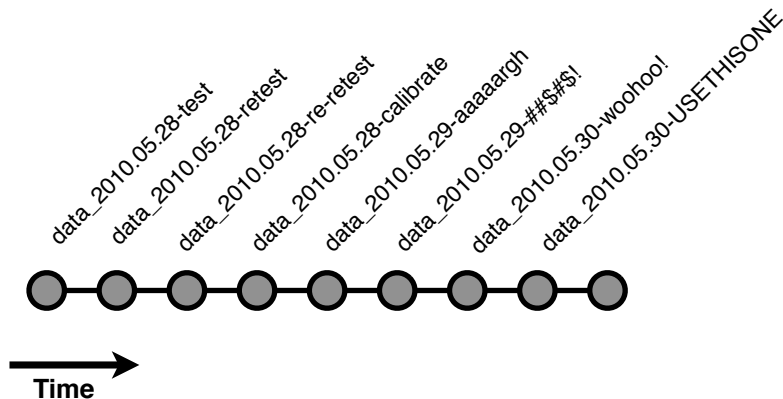
- only readable by you ...
- ... as long as you remember
- no consistent structure or naming scheme enforced
- no detailed description of changes (why were changes performed?)

VERSION CONTROL USING FOLDER AND FILENAMES

- only readable by you ...
- ... as long as you remember
- no consistent structure or naming scheme enforced
- no detailed description of changes (why were changes performed?)
- no easy way of comparing changes between versions (which changes were performed?)

VERSION CONTROL USING FOLDER AND FILENAMES

- only readable by you ...
- ... as long as you remember
- no consistent structure or naming scheme enforced
- no detailed description of changes (why were changes performed?)
- no easy way of comparing changes between versions (which changes were performed?)
- ...



The history of a project can be
viewed as a series of changes

Changes

- A unique identifier
- What changed?
- When did it change?
- Who changed it?
- Why did it change?



Part II: Version Control Systems

DIFFERENT VERSION CONTROL SYSTEMS



GIT

distributed system



Mercurial

distributed system



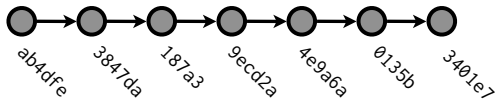
SVN

centralized system

GIT

- only selected version present on disc
- history stored in hidden .git folder
- smart version handling for text based files by using file differences
- user generated commit messages provide reason for and small summary of change

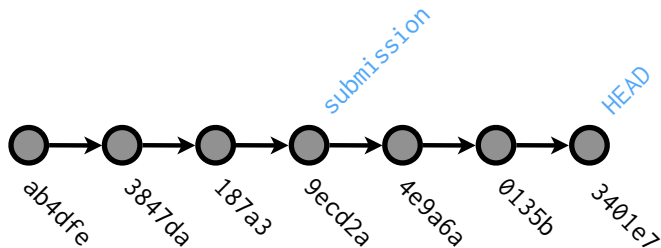
With git, each change (**commit**) is given a unique identifier, called a **sha**



The sha is a key into a database that provides the author, date, and a description

GIT

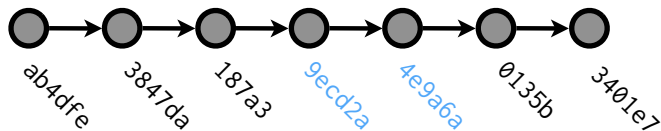
You can also name individual commits



```
git tag submission 9ecd2a
```


GIT

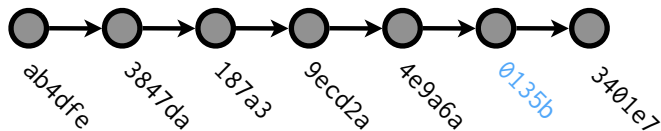
Then see exactly what's changed



```
git diff 9ecd2a..4e9a6a
```

GIT

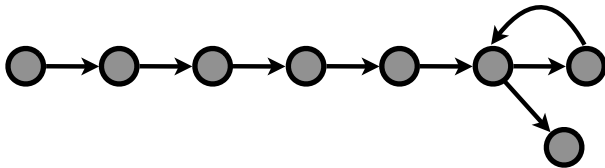
You can revert to a previous change with git checkout



git checkout 0135b

GIT

That allows you to undo mistakes



GIT INTERFACES



Git Cheat Sheet

For more awesome cheat sheets
visit rebellabs.org!



Create a Repository

From scratch -- Create a new local repository
`$ git init [project name]`
Download from an existing repository
`$ git clone my_url`

Observe your Repository

List new or modified files not yet committed
`$ git status`
Show the changes to files not yet staged
`$ git diff`
Show the changes to staged files
`$ git diff --cached`
Show all staged and unstaged file changes
`$ git diff HEAD`
Show the changes between two commit ids
`$ git diff commit1 commit2`
List the change dates and authors for a file
`$ git blame [file]`
Show the file changes for a commit id and/or file
`$ git show [commit]:[file]`
Show full change history
`$ git log`
Show change history for file/directory including diffs
`$ git log -p [file/directory]`

Working with Branches

List all local branches
`$ git branch`
List all branches, local and remote
`$ git branch -av`
Switch to a branch, my_branch, and update working directory
`$ git checkout my_branch`
Create a new branch called new_branch
`$ git branch new_branch`
Delete the branch called my_branch
`$ git branch -d my_branch`
Merge branch_a into branch_b
`$ git checkout branch_b`
`$ git merge branch_a`
Tag the current commit
`$ git tag my_tag`

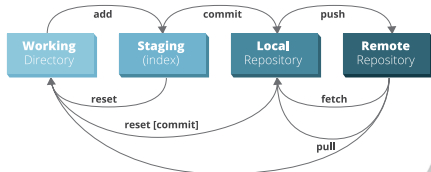
Make a change

Stages the file, ready for commit
`$ git add [file]`
Stage all changed files, ready for commit
`$ git add .`
Commit all staged files to versioned history
`$ git commit -m "commit message"`
Commit all your tracked files to versioned history
`$ git commit -am "commit message"`
Unstages file, keeping the file changes
`$ git reset [file]`
Revert everything to the last commit
`$ git reset --hard`

Synchronize

Get the latest changes from origin (no merge)
`$ git fetch`
Fetch the latest changes from origin and merge
`$ git pull`
Fetch the latest changes from origin and rebase
`$ git pull --rebase`
Push local changes to the origin
`$ git push`
Finally!
When in doubt, use git help
`$ git command --help`


Or visit <https://training.github.com/> for official GitHub training.



BROUGHT TO YOU BY
JRebel

http://files.zereturnaround.com/pdf/zt_git_cheat_sheet.pdf

GIT INTERFACES

 --distributed-even-if-your-workflow-isnt

[About](#)
[Documentation](#)
[Downloads](#)
[GUI Clients](#)
[Logos](#)
[Community](#)

The entire **Pro Git** book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

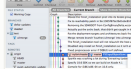
GUI Clients

Git comes with built-in GUI tools for committing (`git-gui`) and browsing (`gitk`), but there are several third-party tools for users looking for platform-specific experience.

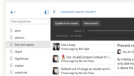
If you want to add another GUI tool to this list, just [follow the instructions](#).

[All](#) [Windows](#) [Mac](#) [Linux](#) [Android](#) [iOS](#)

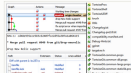
19 Windows GUIs are shown below ↓



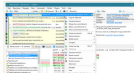
SourceTree
Platform: Mac, Windows
Price: Free
License: Proprietary




GitHub Desktop
Platform: Mac, Windows
Price: Free
License: MIT



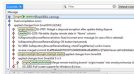
TortoiseGit
Platform: Windows
Price: Free
License: GNU GPL



Git Extensions
Platform: Linux, Mac, Windows
Price: Free
License: GNU GPL



GitKraken
Platform: Linux, Mac, Windows
Price: Free for non-commercial use
License: Proprietary



SmartGit
Platform: Linux, Mac, Windows
Price: Free for non-commercial use
License: Proprietary

<https://git-scm.com/download/gui>

GIT INTERFACES

The screenshot displays a web-based Git interface for a repository named `phd_meeting`. The left sidebar shows navigation options: All commits, Branches (with `master` selected), Remotes, and Tags. The main area shows a list of commits on the `master` branch, all authored by Julia Sprenger. The selected commit is titled "finished introduction". Below the commit list, the user's profile for Julia Sprenger is shown, followed by the commit message "finished introduction". The diff view shows changes to `introduction_version_control.tex`, with line 14 highlighted in red (deletion) and lines 14-15 highlighted in green (addition). The diff content is as follows:

```
... @@ -10,9 +10,9 @@ english,
10 10
11 11 \usepackage{babel}
12 12 \usepackage[utf8]{inputenc}
13 13 \usepackage{hyperref}
14 13 \usepackage[labelfont=bf]{caption}
15 -
14 + \usepackage[colorlinks=true, urlcolor=blue]{hyperref}
15 + \usepackage{pdfpages}
16 16
17 16 \title{Introduction to Version Control}
18 18 \subtitle{Time travel for beginners}
... @@ -21,6 +21,13 @@ english,
21 21 \date{June 29, 2018}
22 22 \titlegraphic{\includegraphics[width=\paperwidth]{graphics/clock-1527693_960_720.jpg}}
23 23 \usepackage{fontspec}
```

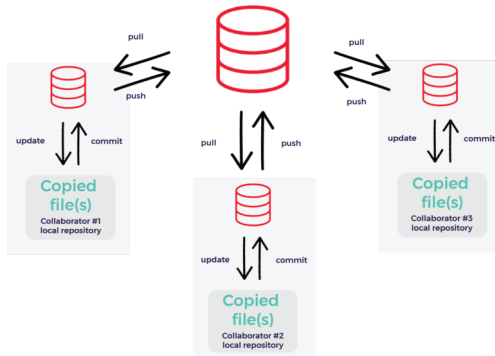


Part III: Collective Version Control

COLLABORATIVE CHAOS CONTROL

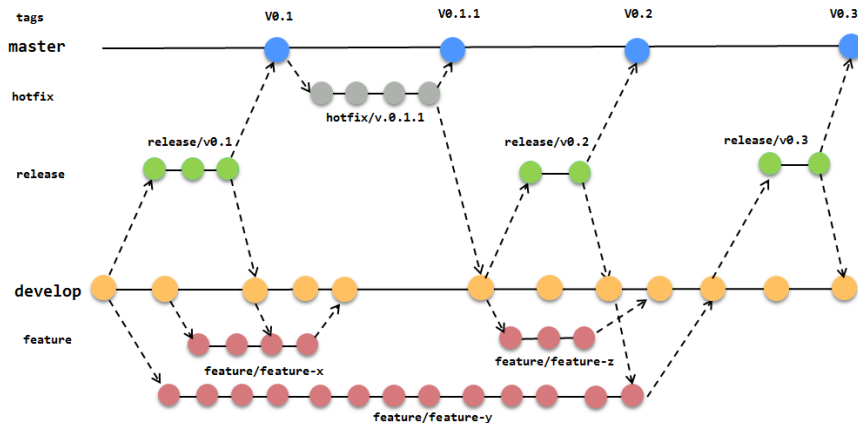
Distributed Version Control

Main Server Repository

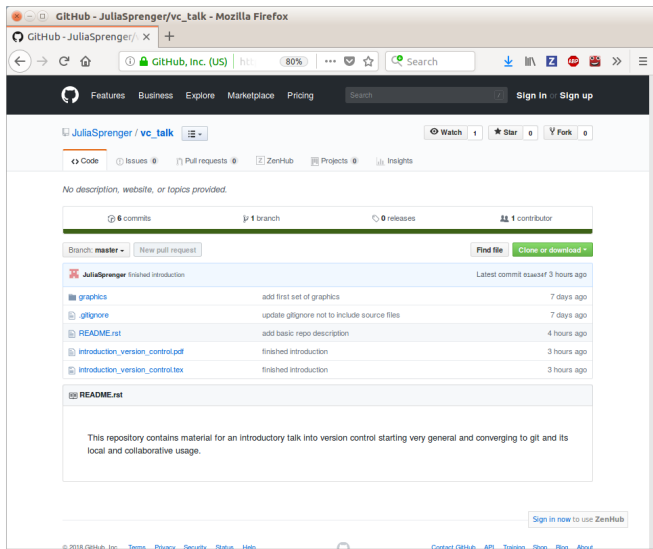


<https://www.positivethinkingcompany.com/articles/articles-web-mobile/git-technology-simplifies-coding-collaboration/>

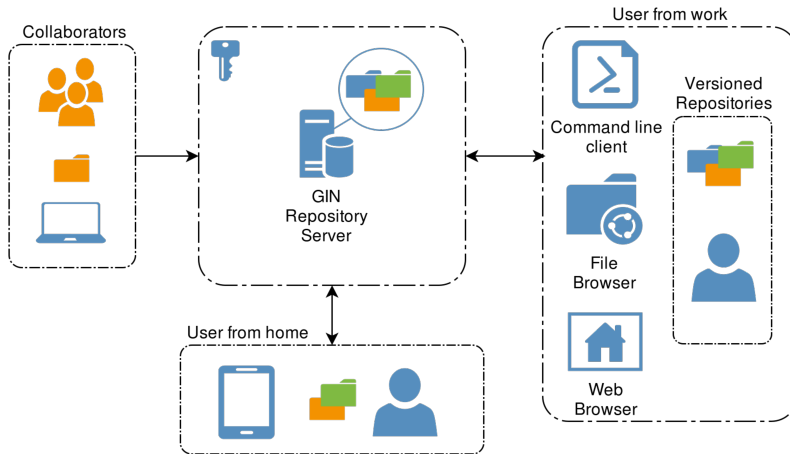
COLLABORATIVE CHAOS CONTROL



<https://fpy.cz/pub/slides/git-workshop>



- visualization of repository content and changes
- issue collection & discussion
- pull requests
- statistics
- **public** & private repositories





Part IV: Version Control for 'big data'

GIT-ANNEX & GIN



GIT-ANNEX

<https://git-annex.branchable.com>

- better suited for large and binary files
- large files are only copied when necessary
- large files are only downloaded by request

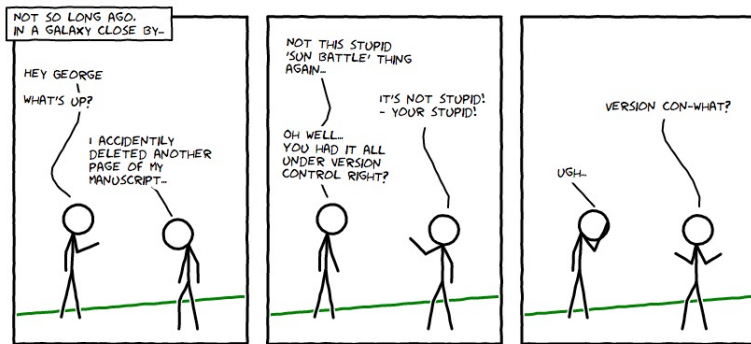


<https://web.gin.g-node.org>

- data management service based on git and git-annex
- public & private repositories
- DOI service
- version controlled
- open source
- developed and hosted by German Neuroinformatics Node

SUMMARY

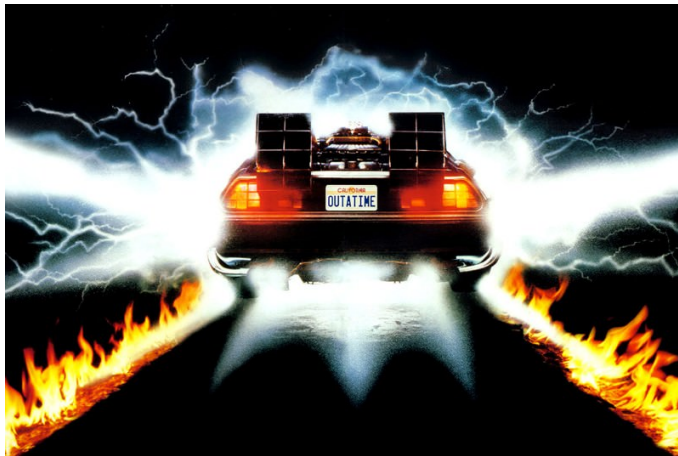
- Version control can help manage your files for
 - local projects
 - collaborative projects via GitHub
- GIT is a distributed version control system ideal for text based files
- GIT-ANNEX extends GIT functionality to also cover large, binary files
- GIN is a data management platform using GIT and GIT-ANNEX



<http://smutch.github.io/VersionControlTutorial>

THANKS FOR YOUR ATTENTION

For new time travel fans: Hands-on GIT session in a future PhD Meeting



<https://www.newscientist.com/article/dn28374-back-to-the-future-does-physics-of-martys-time-travel-add-up/>

REFERENCES AND FURTHER READS

This presentation is available at https://github.com/JuliaSprenger/vc_talk

Inspiration for this presentation comes from

- Version Control Tutorial & GIT

<https://github.com/rstudio/webinars/tree/master/06-Collaboration-and-time-travel-version-control>

More interesting references

- GIT cheatsheet

<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>

- Interactive GIT cheatsheet <http://ndpsoftware.com/git-cheatsheet.html>

- GitHub <https://github.com>

- GIN data management platform <https://web.gin.g-node.org>