GIT For Beginners

Release 1.2

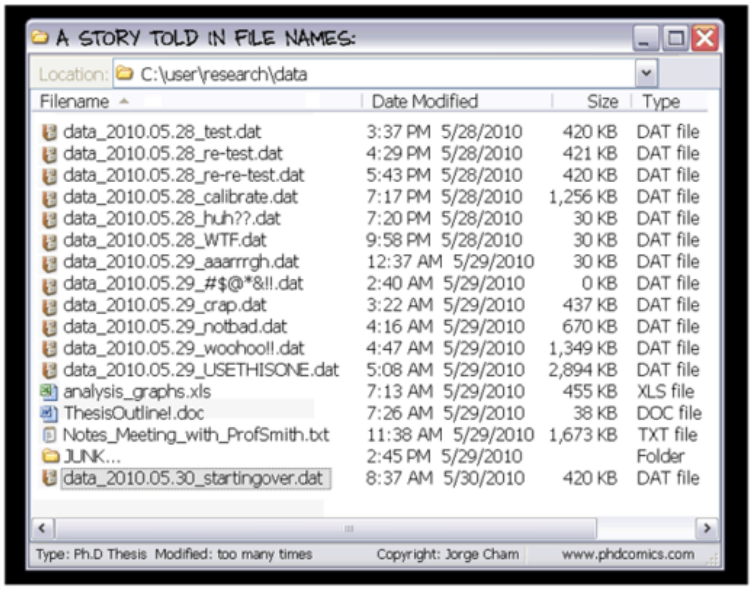
Holger Dinkel

March 21, 2014

Contents

Chapter 1

The Benefits of Version Control



Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. The benefits are at hand:

- **Track incremental backups and recover:** Every document can be backed up automatically and restored at a second's notice.
- **Track every change:** Every infinitesimal change can be recorded and can be used to revert a file to an earlier state.
- **Track writing experiments:** Writing experiments can be sandboxed to copies while keeping the main file intact.
- **Track co-authoring and collaboration:** Teams can work independently on their own files, but merge them into a latest common revision.
- **Track individual contributions:** Good VCS systems tag changes with authors who make them.

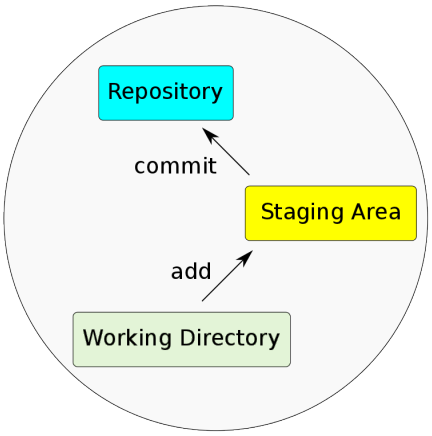


Figure 1.1: Files are *added* from the *working directory*, which always holds the current version of your files, to the *staging area*. *Staged* files will be stored into the repository in the next *commit*. The repository itself contains all previous versions of all files ever committed.

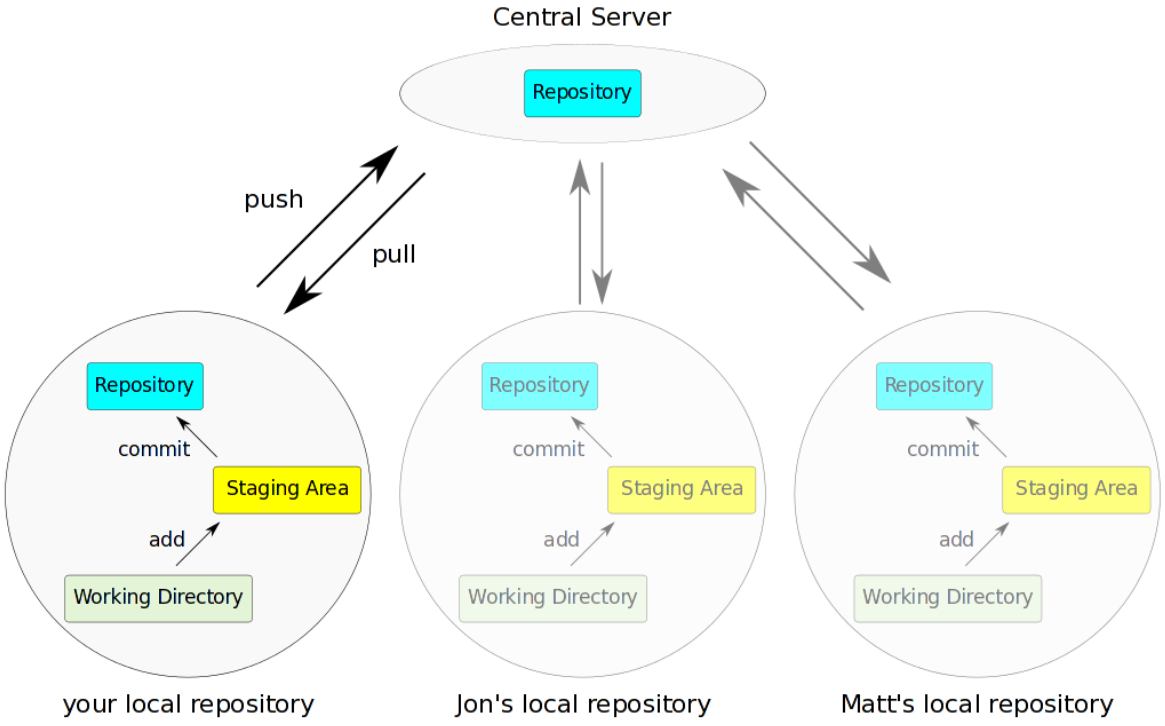


Figure 1.2: Distributed Workflow using a centralized repository. Here, three local copies of one central repository allow you, Jon and Matt to work on the same files and sync files between each other using the central server.

Chapter 2

git at a Glance

The git tool has many subcommands that can be invoked like *git <subcommand>* for instance *git status* to get the status of a repository.

The most important ones (and hence the ones we'll be focusing on) are:

init: initialize a repository

clone: clone a repository

status: get information about a repository

log: view the history and commit messages of the repository

add: add a file to the staging area

commit: commit your changes to your **local** repository

push: push changes to a **remote** repository

pull: pull changes from a **remote** repository

checkout: retrieve a specific version of a file

you can read more about each command by invoking the help:

```
git commit --help
git help commit
```

Chapter 3

git Settings

3.1 setting your identity

Before we start, we should set the user name and e-mail address. This is important because every git commit uses this information and it's also increadibly useful when looking at the history and commit log:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@embl.de
```

Other useful settings include your favorite editor as well as difftool::

```
git config --global core.editor vim
git config --global merge.tool meld
```

3.1.1 Checking Your Settings

You can use the *git config –list* command to list all your settings::

```
git config --list
user.name="John Doe"
user.email=johndoe@embl.de
core.editor=vim
merge.tool=meld
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

Chapter 4

A Typical git Workflow

4.1 Creating a git Repository

Turning an existing directory into a git repository is as simple as changing into that directory and invoking *git init*. Here we first create an empty directory called *new_repository* and create a repository in there::

```
mkdir new_repository
cd new_repository
git init
```

Note: As a result, there should be a directory called *.git* be in this directory...

4.2 Cloning a git Repository

Instead of creating a new directory, we can *index:clone* a repository. That *origin* repository can reside in a different folder on our computer, on a remote machine, or on a dedicated git server:

Local directory::

```
git clone ../other_directory
```

Remote directory::

```
git clone ssh://user@server/project.git
```

Remote git server::

```
git clone git@server:user/project
git clone git@git.embl.de:dinkel/linuxcommandline
```

4.3 Checking the Status

If you don't know in which state the current repository is in, it's always a good idea to check:

```
git status

# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

4.4 Adding files

First, we'll create a new file:

```
echo "First entry in first file!" > file1.txt

git status

# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       file1.txt
nothing added to commit but untracked files present (use "git add" to track)
```

Now we'll add this file to the so called *staging area*:

```
git add file1.txt

git status

# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1.txt
#
```

This tells us that the *file1.txt* has been added and can be committed to the repository.

4.5 Committing changes

It might be a bit confusing at first to find out that *git add* does **not** add a file to the repository. You need to *commit* the file/changes to do that:

```
git commit -m "message describing the changes you made"
```

Note: You **MUST** provide a commit message! git will ignore your attempt to commit if the message is empty. If you do not provide the *-m* parameter, git will open an editor in which you should write your commit message (can be multiple lines of text). Once you save/quit your editor, git will continue to commit...

After succesfully committing, we can check the status again:

```
git status

# On branch master
nothing to commit, working directory clean
```

4.6 Viewing the History

You can use *git log* to view the history of a repository. All previous commits including details such as Name & Email-address of the committer, Date & Time of the commit as well as the actual commit message are shown:

```
git log

commit  <some hash value identifying this commit>
Author: <your name and email address>
Date:   <the actual date of the commit>

message describing the changes you made
```

4.7 Pushing changes

If we had cloned this repository from a remote location, we probably want our changes to be propagated to that repository as well. To push all committed changes, simply type:

```
git push
```

Note: git "knows" from which location you had cloned this repository and will try to push to exactly that location (using the protocol you used to clone: ssh, git, etc)...

Warning: If you get a warning message, you probably 'just' need to pull others changes before you are allowed to push your own...

4.8 Pulling changes

To update your local repository with changes from others, you need to *pull* these changes. In a centralized workflow you actually **must** pull changes that other people have contributed, before you can submit your own.

```
git pull
```

Warning: Ideally, changes from others don't conflict with yours, but whenever someone else has edited the same lines in the same files as you, you will receive an error message about a **merge conflict**. You will need to resolve this conflict manually, then add each resolved file (*git add*) and commit.

4.9 Undo local changes

One of the great features of using version control is that you can revert (undo) changes easily. If you want to undo all changes in a local file, you simply checkout the latest version of this file:

```
git checkout -- <filename>
```

Warning: You will loose all changes you made since the last commit!

Chapter 5

EMBL git server

As part of the Bio-IT initiative, EMBL provides a central git server which can be used as a centralized resource to share and exchange data/code with collaborators:

http://git.embl.de/

The following rules apply:

- Repositories on the EMBL Git server are only granted to EMBL staff members.
- External users can be added as cooperators on a project, but the projects themselves have to be lead by someone with an active EMBL contract.
- Should the project leader leave EMBL, then the project has to be transferred to someone else or the complete repository will be removed.
- Repositories are always installed as sub-repositories of the project leader/repository responsible.
- By default, repositories are installed with only basic access permissions for the repository owner. He/she is then in charge of setting appropriate access permissions as described on the Howto page.

Basically, to use this server, you need to provide your full name, your EMBL email address and username, the name and a short description of the repository/project, along with your SSH public key to the admin and he will set things up so you are able to access your repository:

```
git clone git@git.embl.de:your_username/your_repository
```

Note: It's important to mention that the username for accessing the git.embl.de server is always **git**, not **your** username!

An SSH key can be generated using the command *ssh-keygen* (Windows users might want to use putty) like so:

```
ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/home/username/.ssh/id_rsa):
Created directory '/home/username/.ssh'.
```

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/username/.ssh/id_rsa.
Your public key has been saved in /home/username/.ssh/id_rsa.pub.
The key fingerprint is: 2d:14:f5:d8:... username@hostname
```

This creates two files, in this case */home/username/.ssh/id_rsa* and */home/username/.ssh/id_rsa.pub*. The former is your **private** key and should **never** be handed out to anybody, while the latter one (ending in *.pub*) should be distributed to any server on which you intend to use it...

Chapter 6

Links/References

the git program itself:

- git for Windows
- git for Mac

Tools:

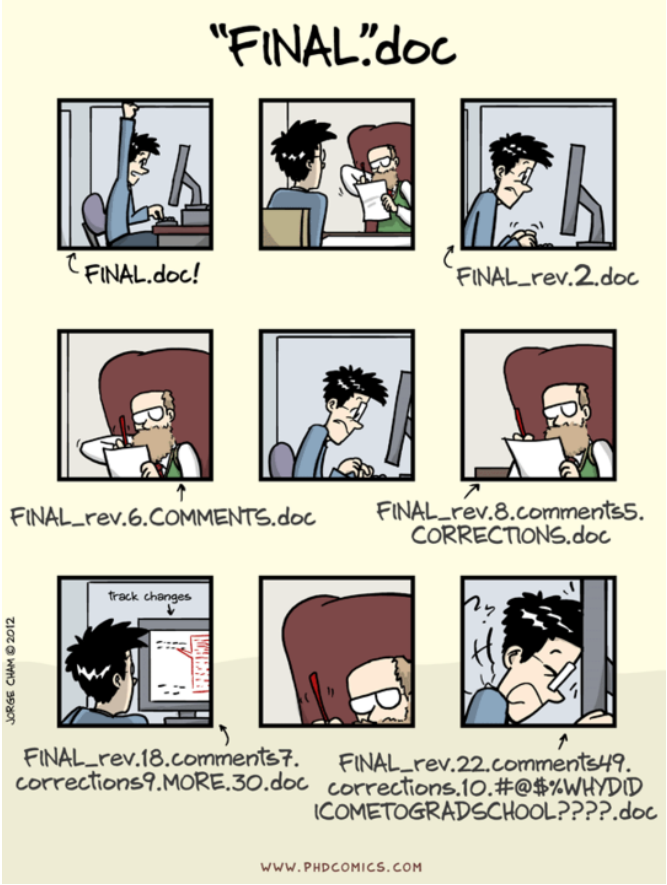
- SourceTree (a graphical user interface for git)
- DiffMerge (a graphical merge tool)
- Kdiff3 (another graphical merge tool)

References:

- Try Git
- A Visual Git Reference
- A visual guide to version control
- Version control for scientific research
- Software Carpentry's introduction to git

Scientific Articles About Git:

- Git can facilitate greater reproducibility and increased transparency in science
- Improving the reuse of computational models through version control


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Chapter 7

About Bio-IT

Bio-IT is a community project aiming to develop and strengthen the bioinformatics user community at EMBL Heidelberg. It is made up of members across the different EMBL Heidelberg units and core facilities. The project works to achieve these aims, firstly, by providing a forum for discussing and sharing information and ideas on computational biology and bioinformatics, focused on the Bio-IT portal. Secondly, we organise and participate in a range of different networking and social activites aiming to strengthen ties across the community.

7.1 Resources

A list of biocomputing-related resources associated with the project, in the top-left "Resources" menu, including, for example there is help provided for installing software on Linux computers at EMBL, instructions on using the Git versions control system server provided by EMBL, and various other kinds of information.

7.2 Training and Outreach

The "Training and Outreach" menu provides information on events (courses and conferences), both internal to EMBL and organised elsewhere by other organisations, that are related to biocomputing and bioinformatics

7.3 Networking

Several different kinds of networking events for the Bio-IT community are being organised, including beer sessions for the EMBL community, and within-Heidelberg events for the larger Heidelberg biocomputing community.

7.4 Biocomputing expertise at EMBL

You can use the Bio-IT portal to search for people working at EMBL who have experience working with data or tools you might be interested in.

If you've not yet got a page up on the portal describing your own expertise, please do so. If you need any help doing this, you can read about this in the portal's FAQ section, or get in touch with one of the site administrators.

7.5 Centers

EMBL Centres are 'horizontal', cross-departmental structures that promote innovative research projects across disciplines. All the EMBL Centres listed below have a strong computational component.

7.5.1 Biomolecular Network Analysis

The CBNA disseminates expertise, know-how and guidance in network integration and analysis throughout EMBL.

7.5.2 Statistical Data Analysis

The CSDA helps EMBL scientists to use adequate statistical methods for their specific technological or biological applications.

7.5.3 Molecular and Cellular Imaging

The CMCI makes your life in image processing/analysis easier and more fun.

7.5.4 Modeling

The Centre for Biological Modeling (CBM) aims to support people to adopt mathematical modeling techniques into their everyday research.

Chapter 8

Acknowledgements

Handouts provided by EMBL Heidelberg Photolab (Many thanks to Udo Ringeisen)

The git server at git.embl.de is maintained by Frank Thommen.