Working with GIT

Florido Paganelli
Lund University
florido.paganelli@hep.lu.se

Required Software

- **Git** - a free and open source distributed version control system
- **Gitg** – a fast git repository viewer (there are many!)

Command line installation (`bash`):

```
sudo apt-get install git gitg
```

Note: this software is NOT installed by default by the Lubuntu system installation.

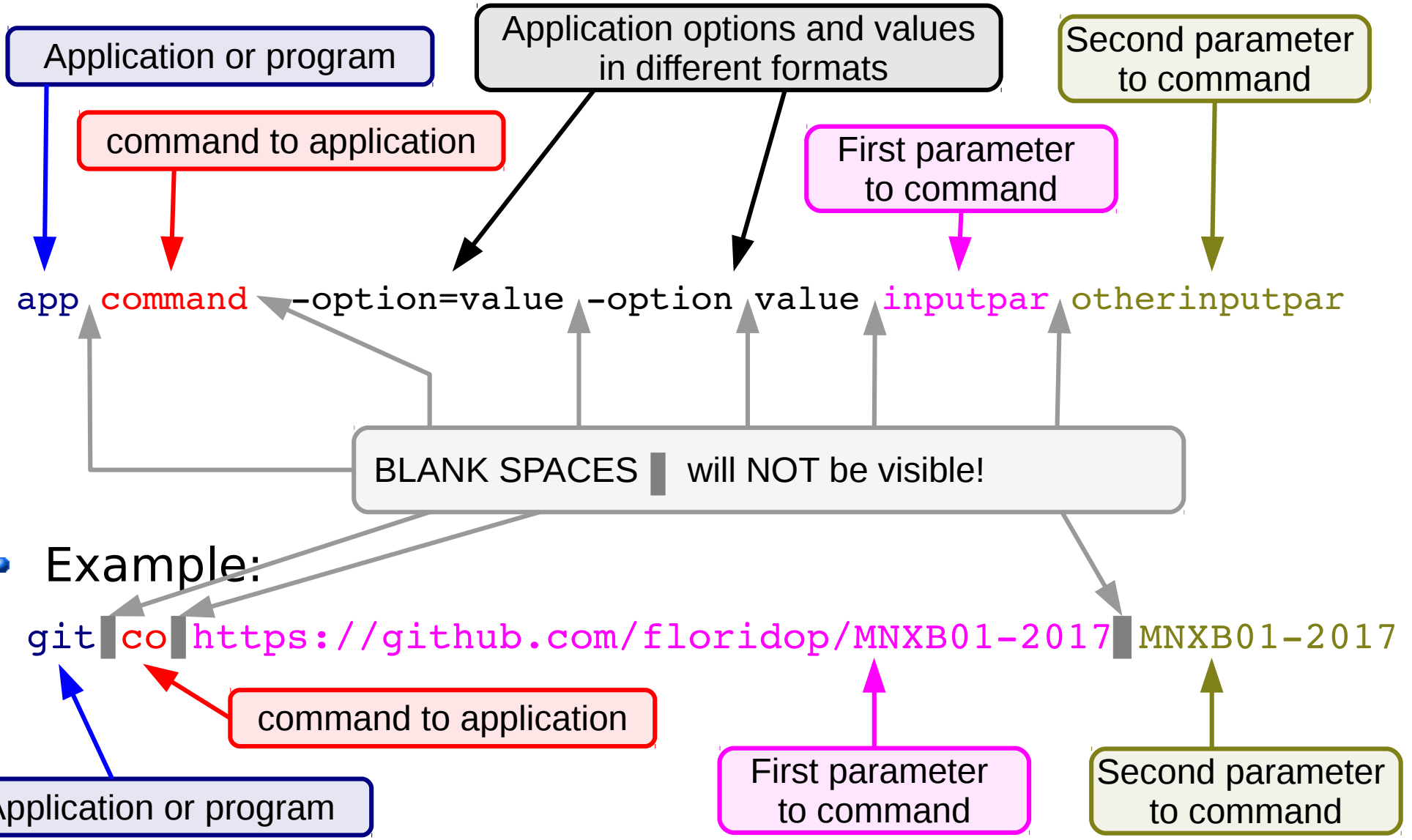
Platform	Package names
Ubuntu, Debian	**`git, gitg`**
RedHat, CentOS, Fedora, SuSE	**`git, gitg`**
Windows	http://www.jamessturtevant.com/posts/5-Ways-to-Install-git-on-Windows/
Mac OS	http://www.macworld.co.uk/how-to/mac-software/how-use-git-github-on-your-mac-3639136/

Outline

- What are version/revision control systems
  - Generic concepts of version/revision systems
- git
  - Generic concepts of git
  - git tutorial
  - Additional useful commands

Notation

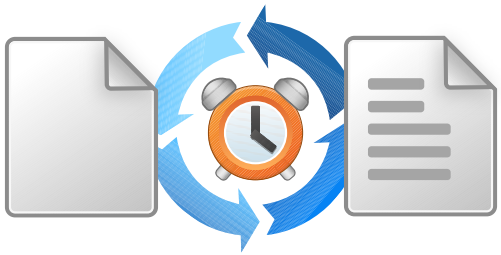
- I will be using the following color code for showing commands:



- Example:

Why version/revision systems?

- Say you wrote some computer program in a text file.
- You discover a bug, something that does not work as it should, and you want to change it.
- You fix the bug, save the file. Try the program again and... **it doesn't work anymore!**
- **What went wrong?** Would be nice if you could **compare** what you **changed**...
- **Solution:** make a backup copy before every (important) change!
- Version systems make it easy to backup and compare **changes**



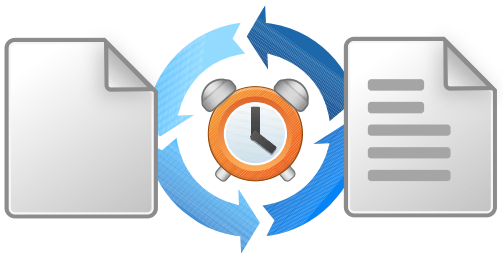
Why version/revision systems?

- If you do many changes, you might not remember what changes you made. Version systems allow you to attach a **comment** to the change.

- If you want to share your code with other developers, it's nice if everybody can see who changed what. Version systems allow you to **author** the changes so the others know what you're done. This helps **sharing** code.

Why version/revision systems?

- Summary:
  - **Backup** each change in your code
  - **Compare** different versions of your code
  - **Comment** and annotate each change
  - **Share** among developers



Version systems: products and features

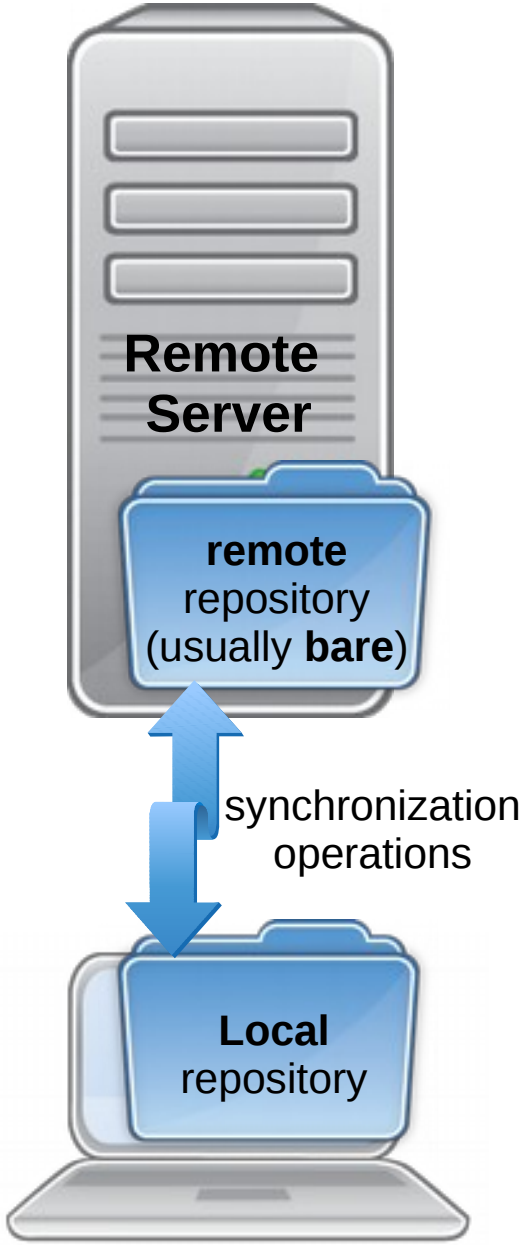
Product	staging	Local commit	diff	Fork/branch management	Distributed/ Collaborative	Compatibility
CVS (Current Version Stable)	N	N	Y	Y	N	?
SVN (SubVersioN)	N	N	Y	N	N	?
Git	Y	Y	Y	Y	Y	SVN CVS

What and why git

- Was created by Linus Torvalds especially for kernel development
  - Highly distributed community contributions
  - Lots of people *forking* and writing their own version of drivers (later I'll explain this term)
- Nowadays there are many collaborative websites systems that use it to share code (github, gitlab) and make it easier to integrate everyone's work with discussion and code revision/testing tools
- Is being used by many because is a free solution that helps distributed cooperation
- Becoming the most used among research projects

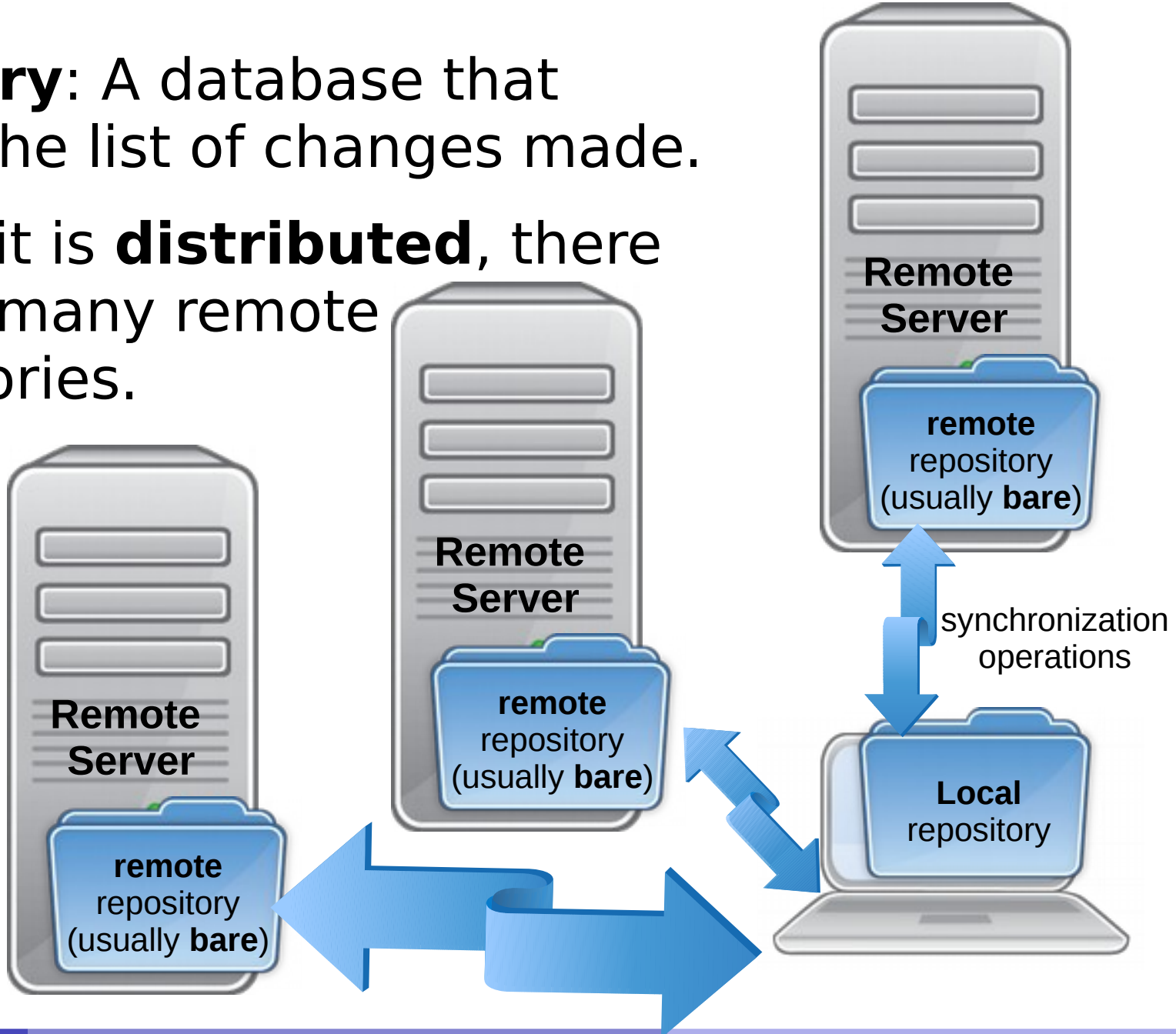
Concepts of version systems in git

- **Repository**: A database that contains the list of changes made.
  - A **local** git repository is shared *locally on your machine* in the **`.git`** invisible folder
  - A **remote** git repository is shared on a *remote server* and can be reached using a URL, like https://github.com/floridop/MNXB01-2017
  - A **bare** git repository can be stored in any folder and contains data in a form that only the git code understands. Can be used to have multiple copies of the same repository. It can be used to share a repository without github.



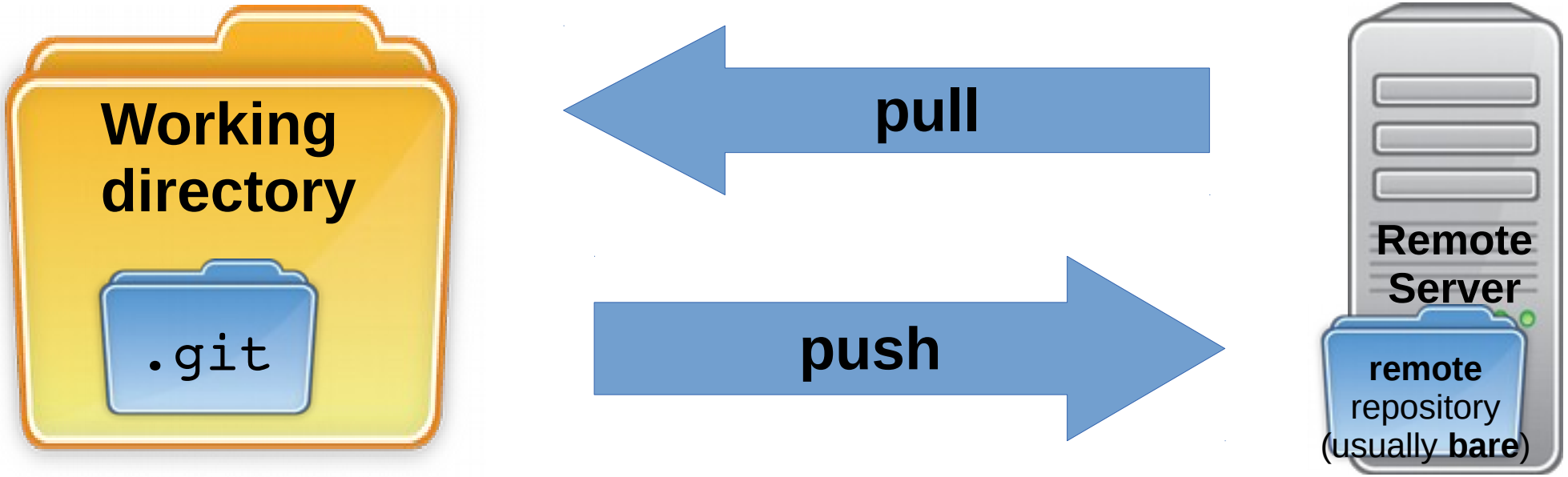
Concepts of version systems in git

- **Repository**: A database that contains the list of changes made.
  - Since git is **distributed**, there can be many remote repositories.



Concepts of version systems in git

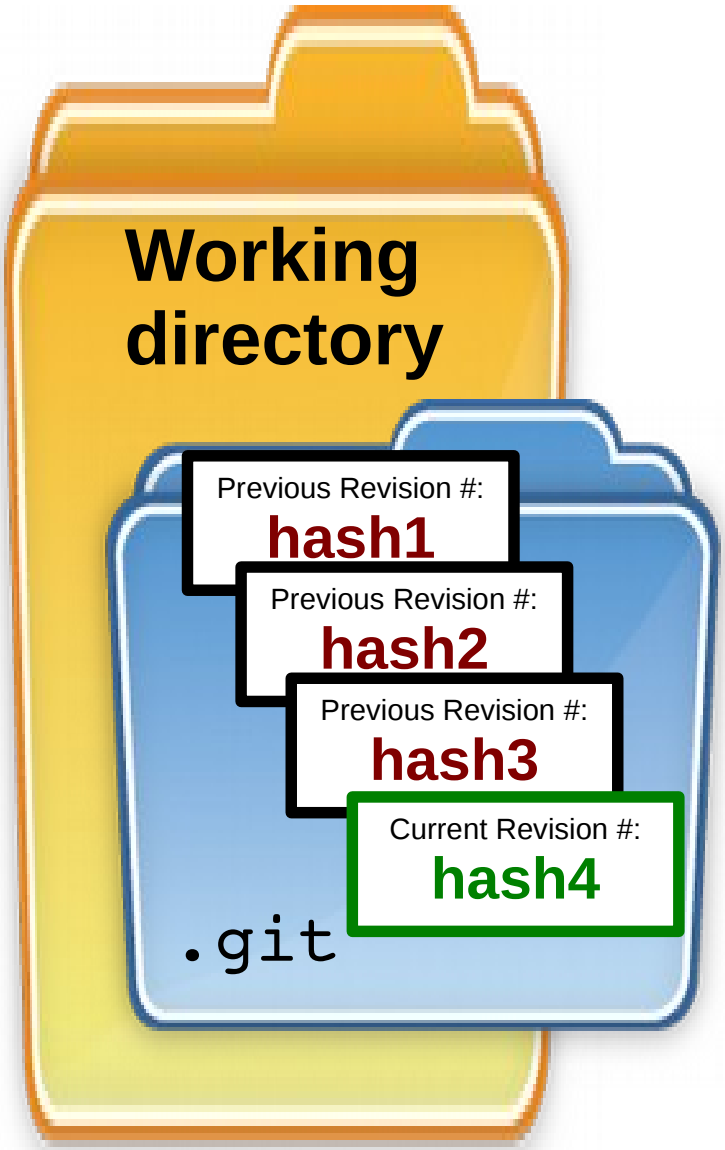
- **Working directory**: the latest version of a set of files that you want to work on. This is usually **local** to your machine.
  - It is usually the result of a **clone**, an exact copy, of some remote repository
  - You can synchronize the local git repository with remote ones using the **push** (send changes) and **pull** (retrieve changes) commands.



Concepts of version systems

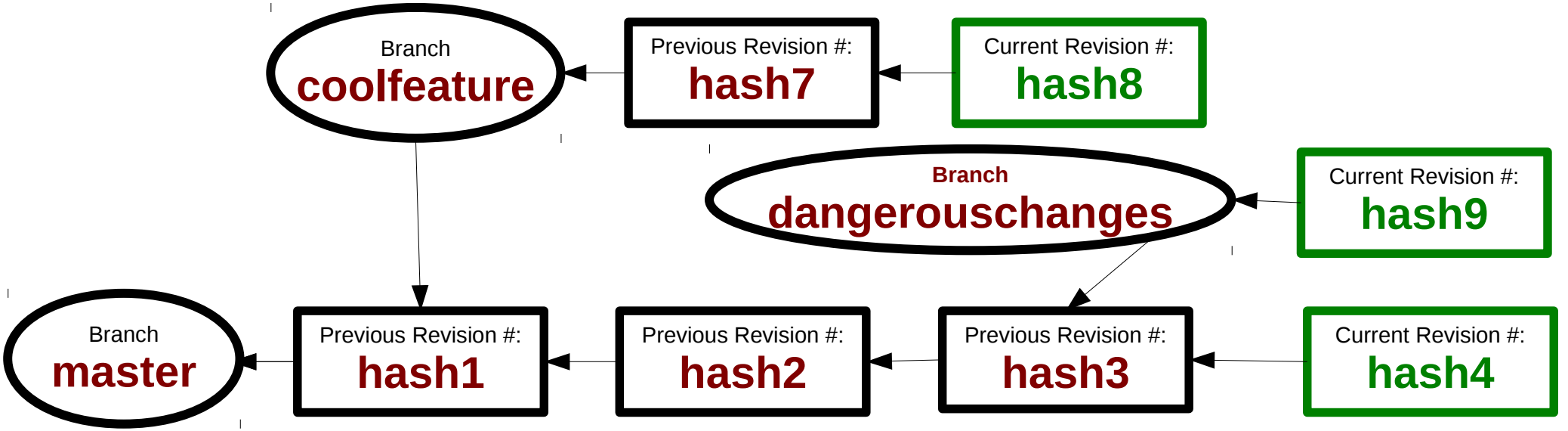
- **Revisions** or **commit ID**: every "version" of one or more files gets a **revision tag**. This can be a number, a label, a string. In git usually is an hash*.
  It somewhat identifies the moment in time when these files were "accepted" as good for the rest of the project.
  For this reason these systems are also known as
  Revision Systems, as every revision gets a label that depends on time and person who made the change.

*Hash: a special injective function that returns a value from a finite set of strings. The return values are uniques under certain conditions.



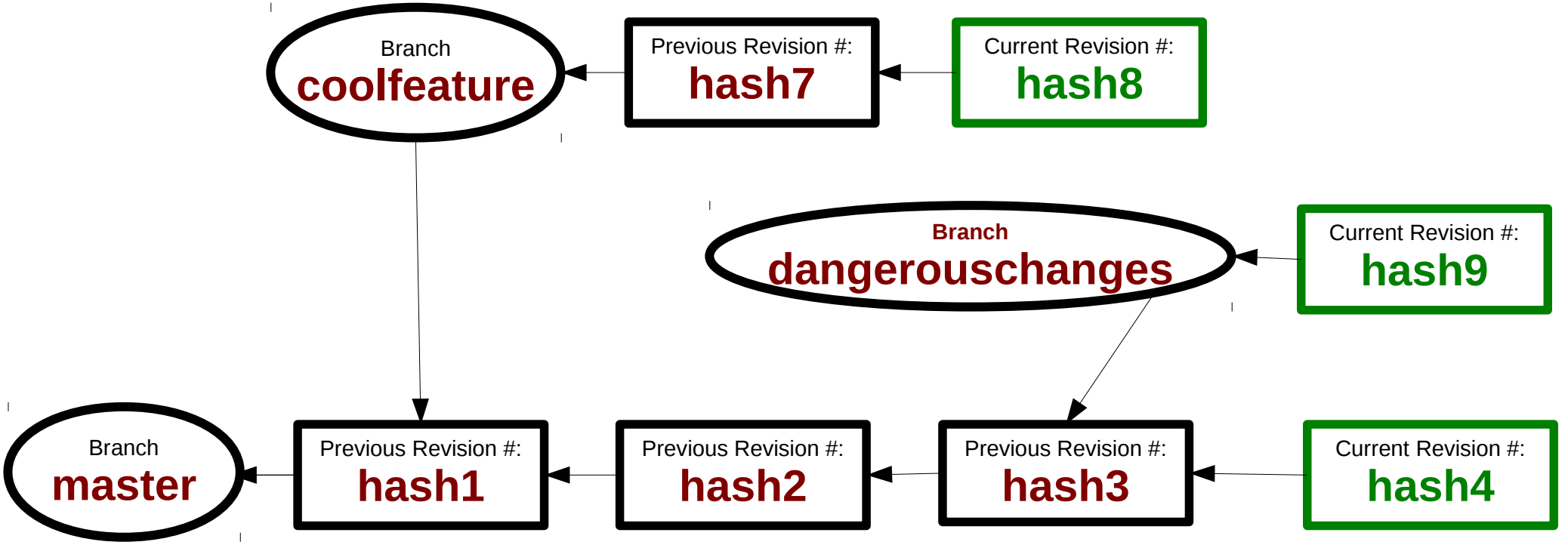
Concepts of version systems
git specifics

- A repository might have one or more **branch**es, that is, different version of the same repository which modify or propose different features.
  - They're called branches because they can be visualized like a tree as they diverge from some initial branch, usually called **master**.
    Every branch has a **name**.



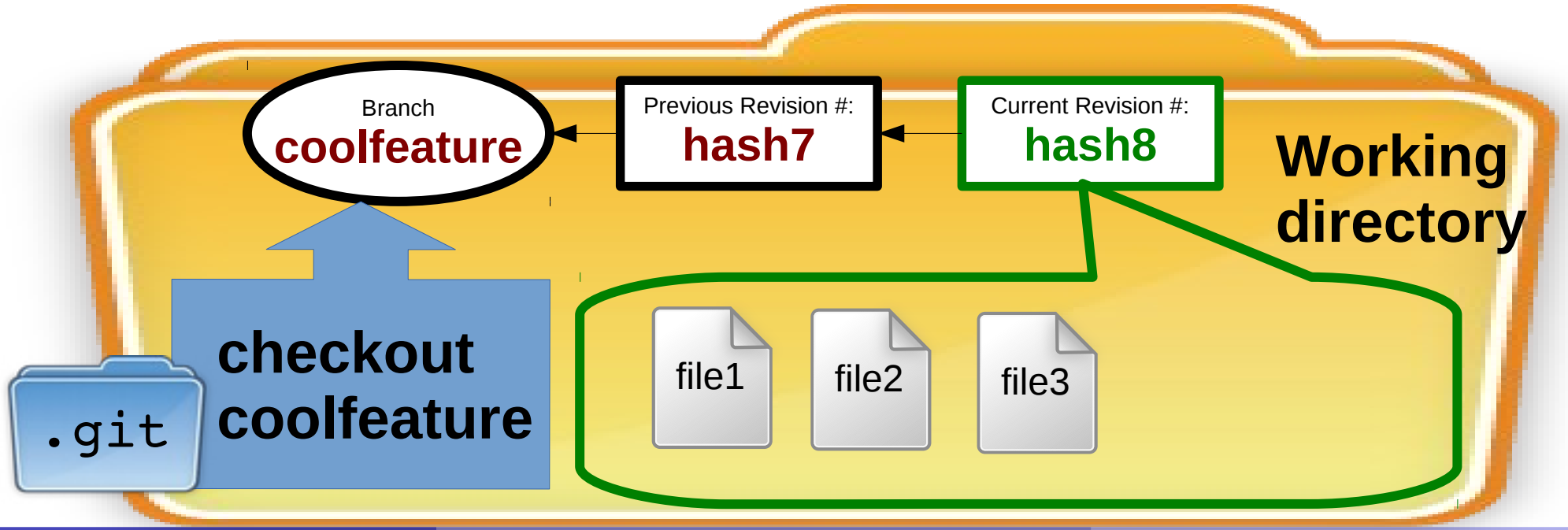
Concepts of version systems
git branch

- Every branch **history** is a continuation of the history where the master was branched.
- It is possible to branch from a branch, not just from the master. Use with care, can be confusing!



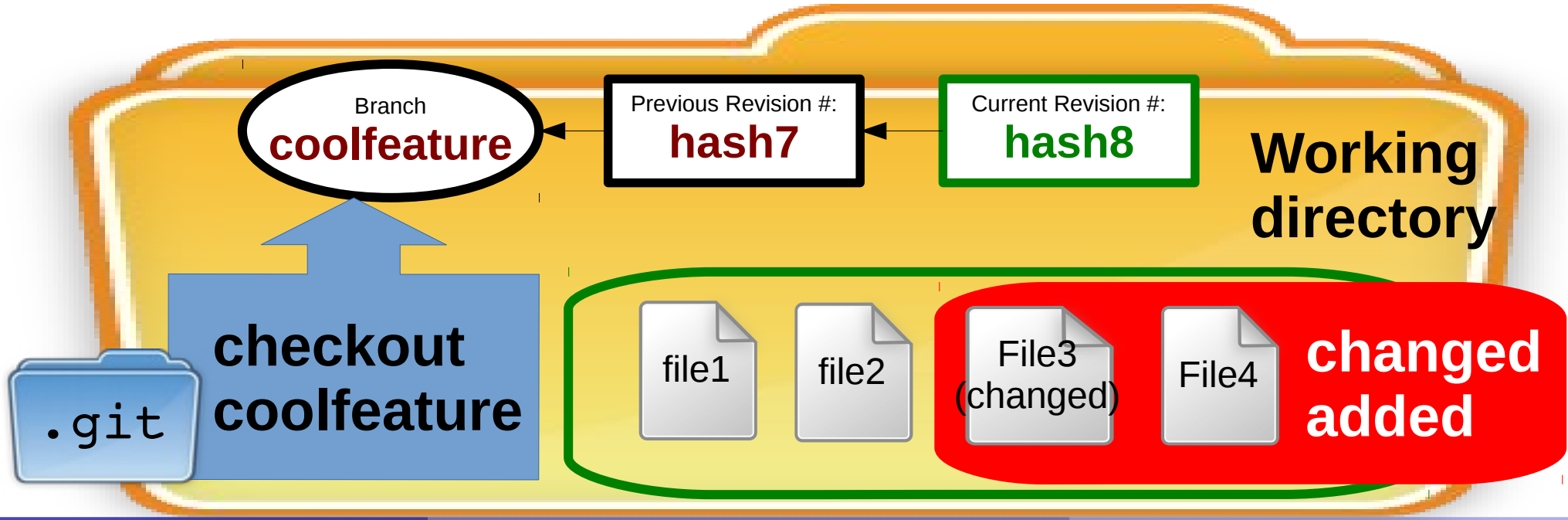
Concepts of version systems git branch

- A branch can be made *active* with the **checkout** operation. When a branch is checked out you will be able to **see its files in your working directory**.
  - To check out a branch means to select a certain history of changes.



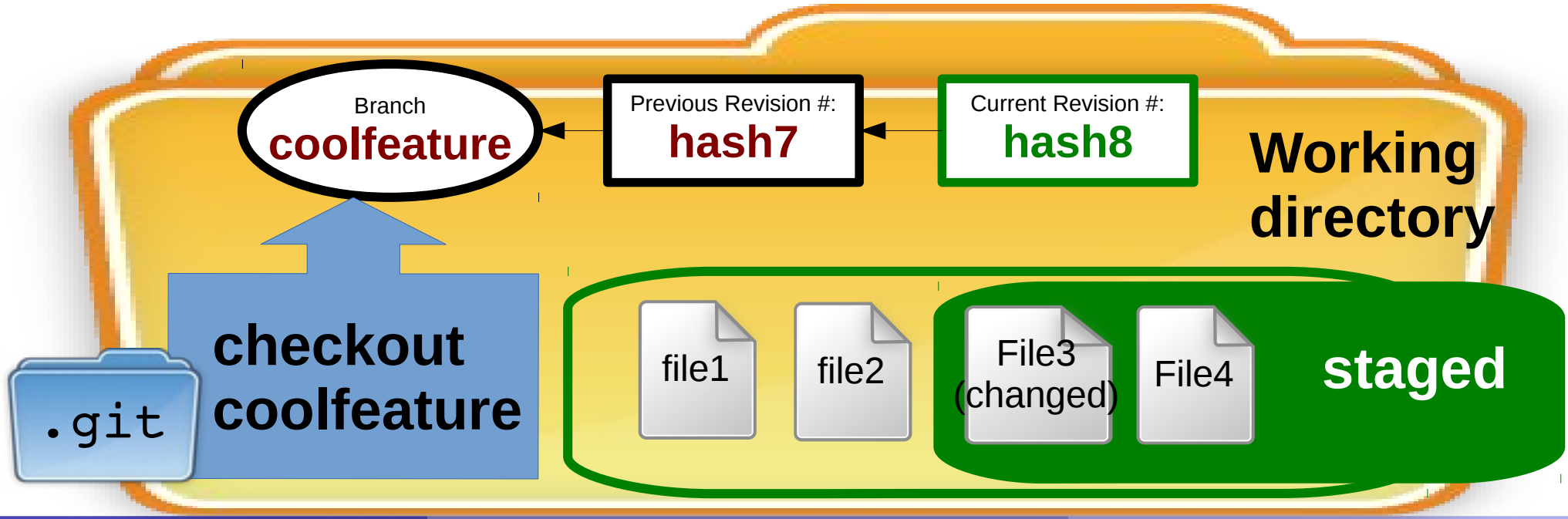
Concepts of version systems git add

- If one modifies or changes files contained in a certain revision, git can see it, and gives the choice to **add** these changes to the database



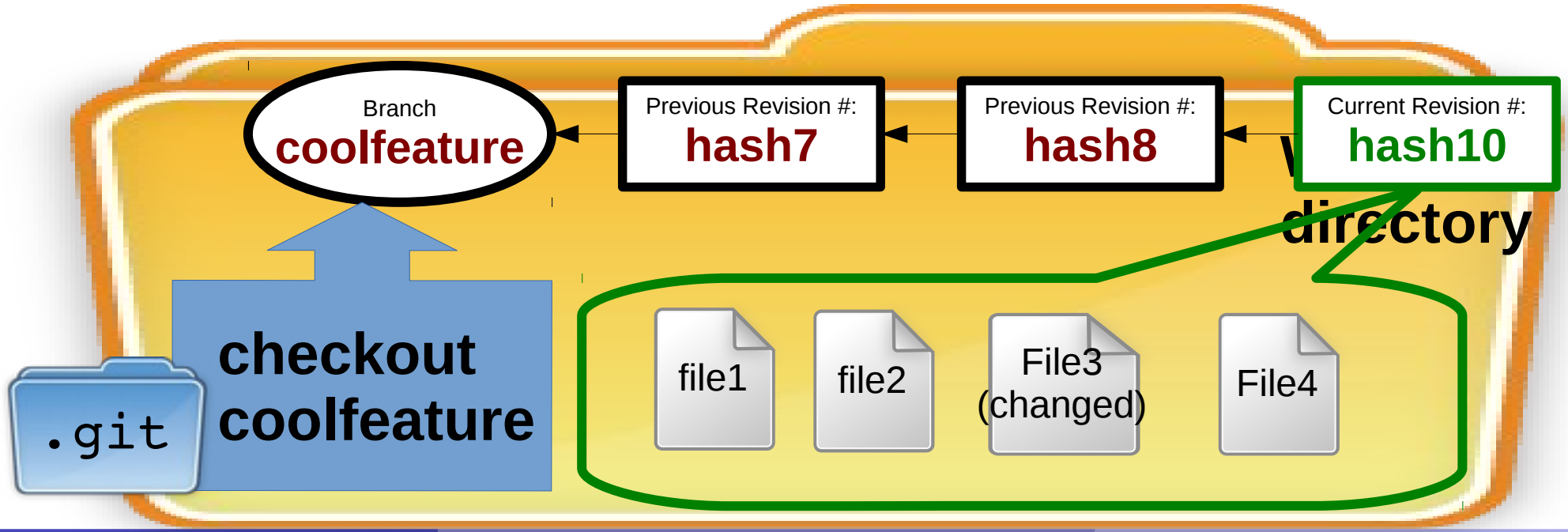
Concepts of version systems git add

- Added files are *staged* to be part of a next revision.



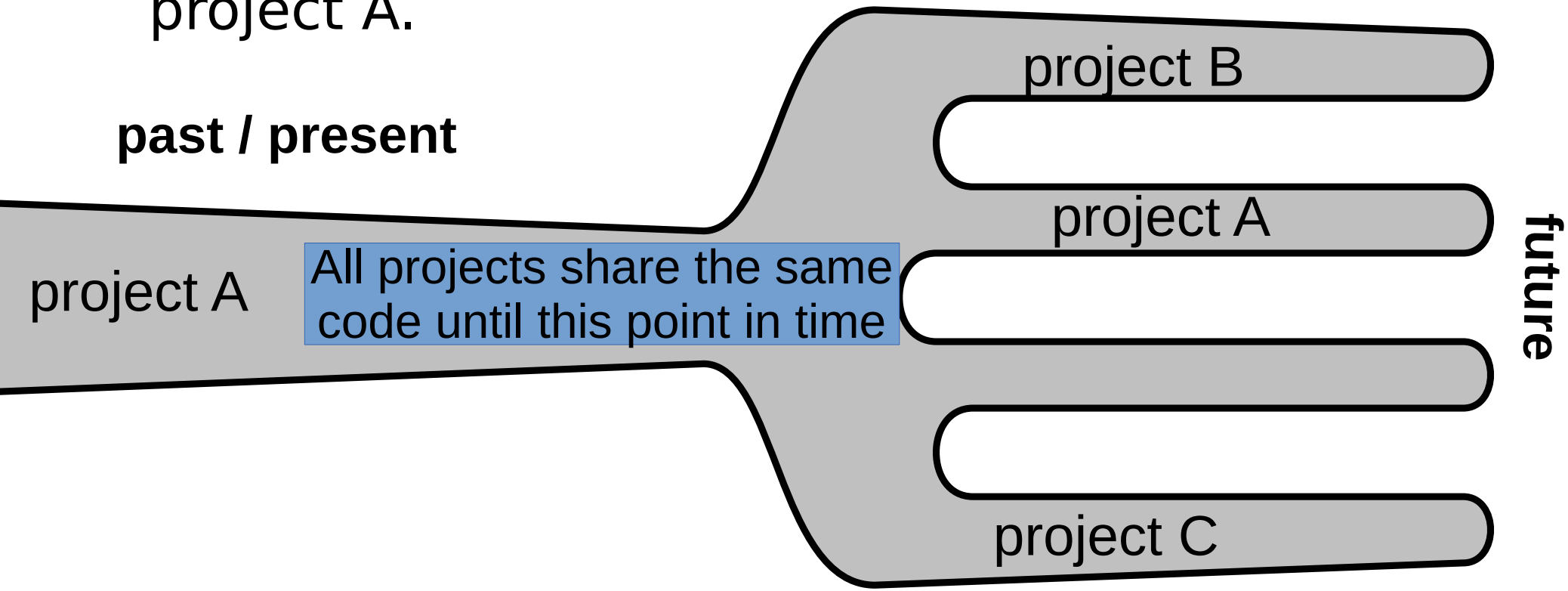
Concepts of version systems
git commit

- *Staged* files will then be actually become part of a new revision in the database once the user **commits** them.



What is a software fork

- In software engineering, a **fork** of a software project A it's a copy of the software source code of A to develop features for a project B,C,... that follow completely independent choices from project A.



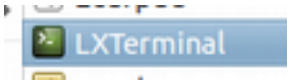
Preparing for the tutorial

- Create a folder in your home folder
  - `mkdir ~/git/`
  - `cd ~/git`
- Download the tutorial app (also available on L@L):
- `wget http://www.hep.lu.se/staff/paganelli/fileshare/gitmnxb01.tgz`
- Unpack the tutorial app:
  - `tar zxvf gitmnxb01.tgz`
- Enter the created directory and start the app:
  - `cd Git-it-linux-ia32`
  - `./Git-it &`

Reminder: the ~ symbol means "my home folder", that is `/home/courseuser/`
the above commands will create (make directory) and go inside (change directory) `/home/courseuser/git/`

Creating and editing a file

- During the tutorial you'll be asked many times to do things with files. For those of you not familiar with **file editing**, here's a small how-to.
- There are many ways of creating a file.
  One way is by using a **text editor**
- The favorite text editor for this course is called **geany**. Can you find the icon in the menu? Open it by clicking on the icon.

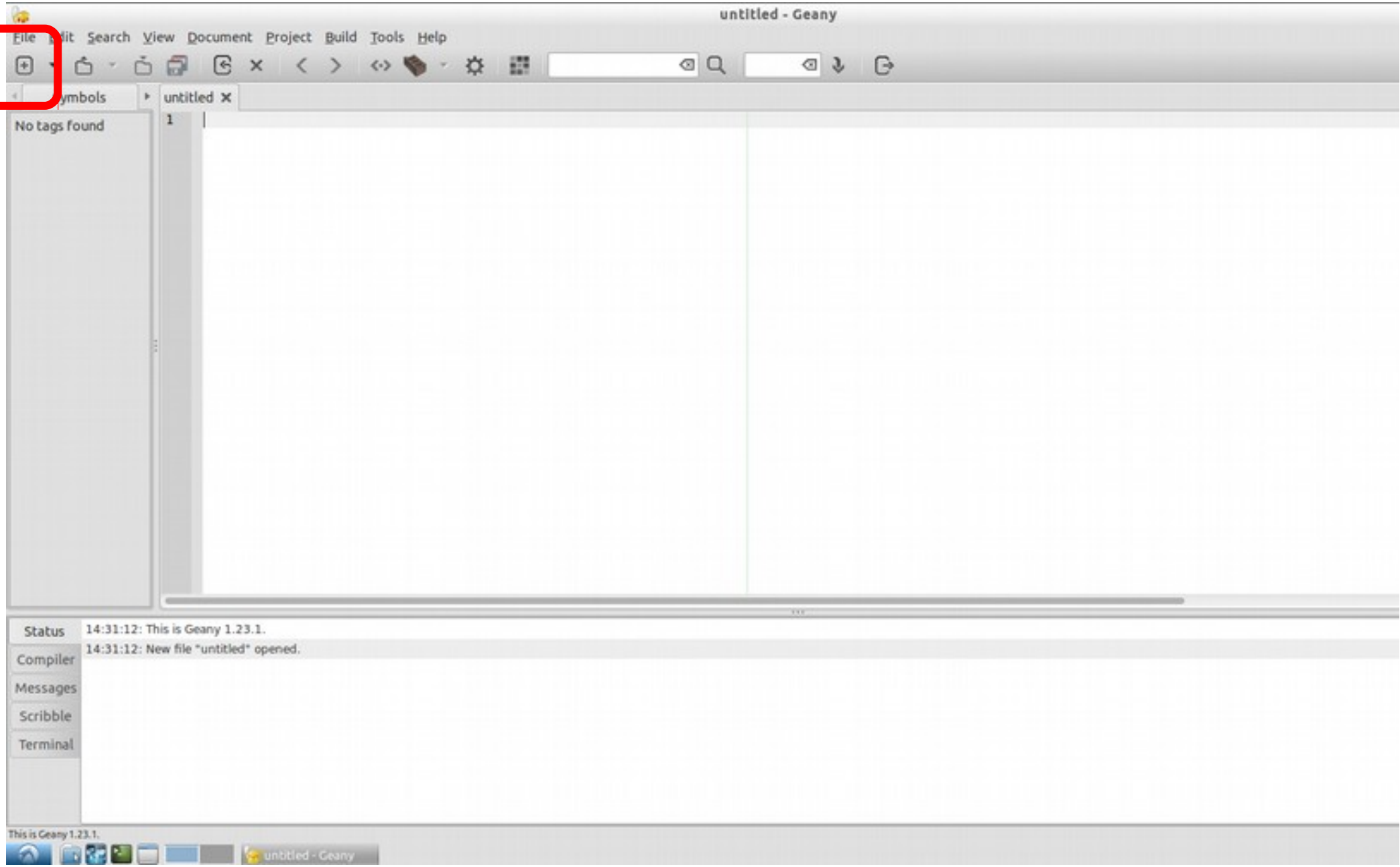
- Alternatively, open a terminal 

and write the command:

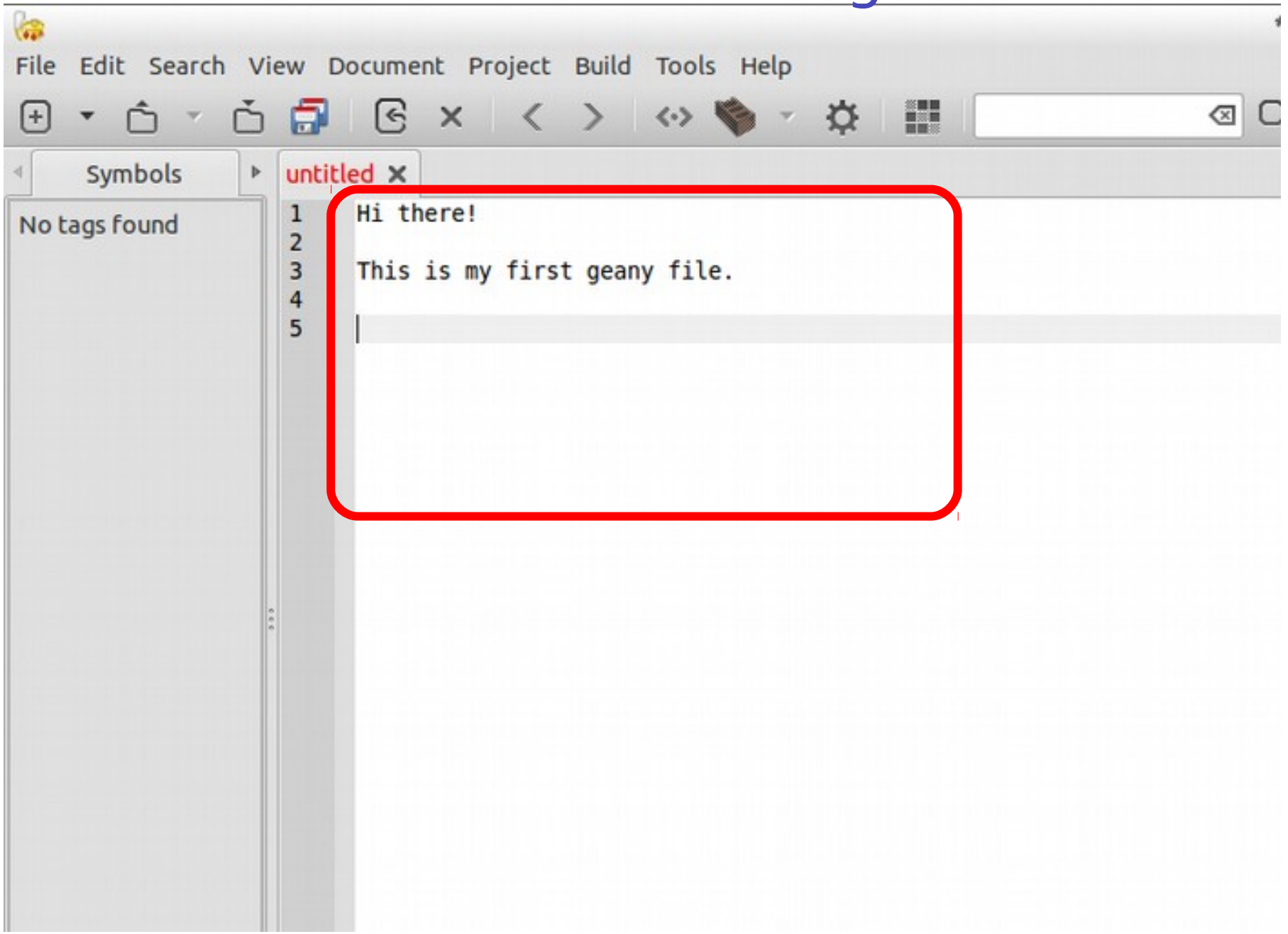
```
geany &
```

(the & symbol sends the command execution in background, see tutorial 1b!)

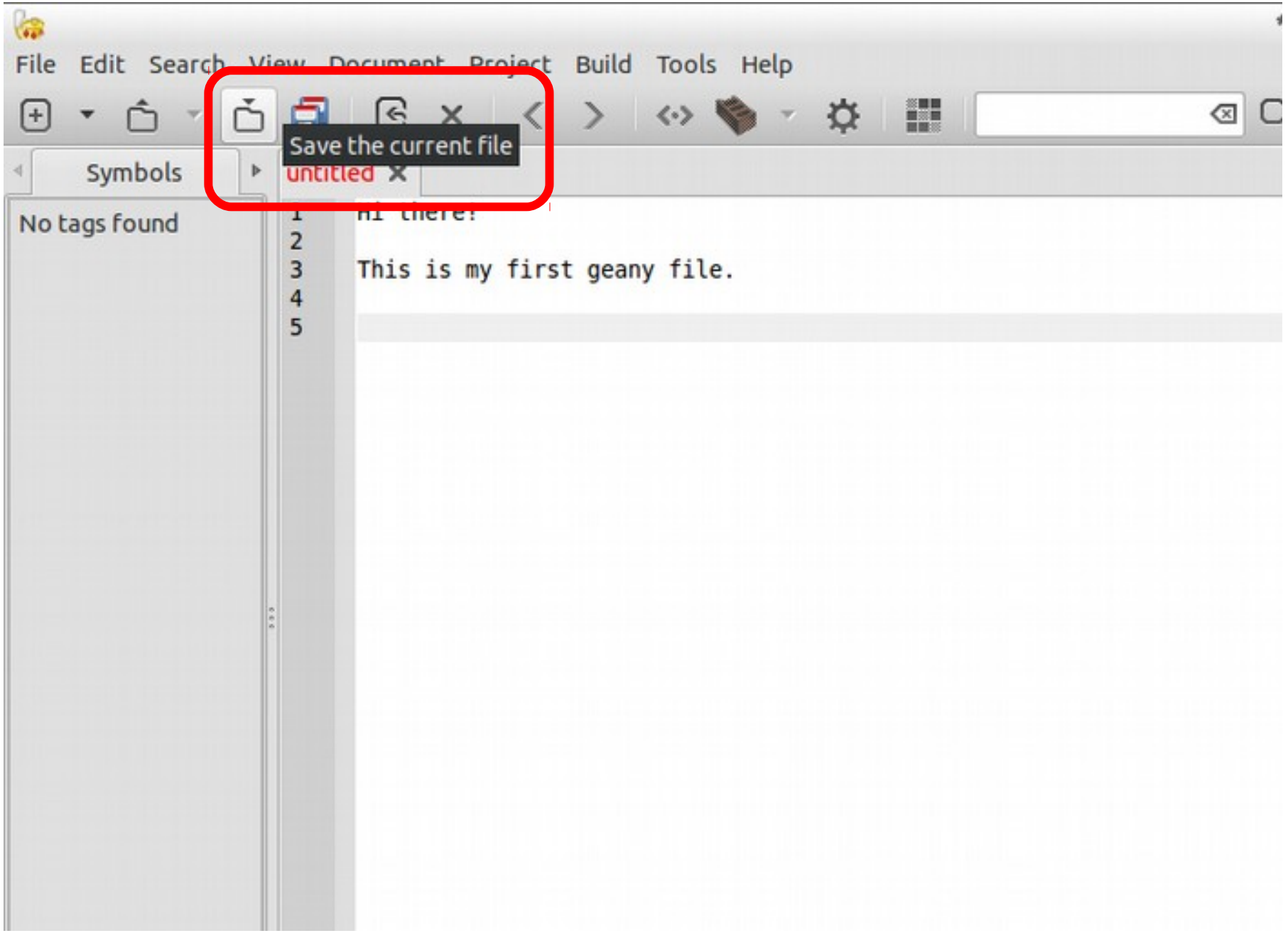
Editing and saving a file: create new



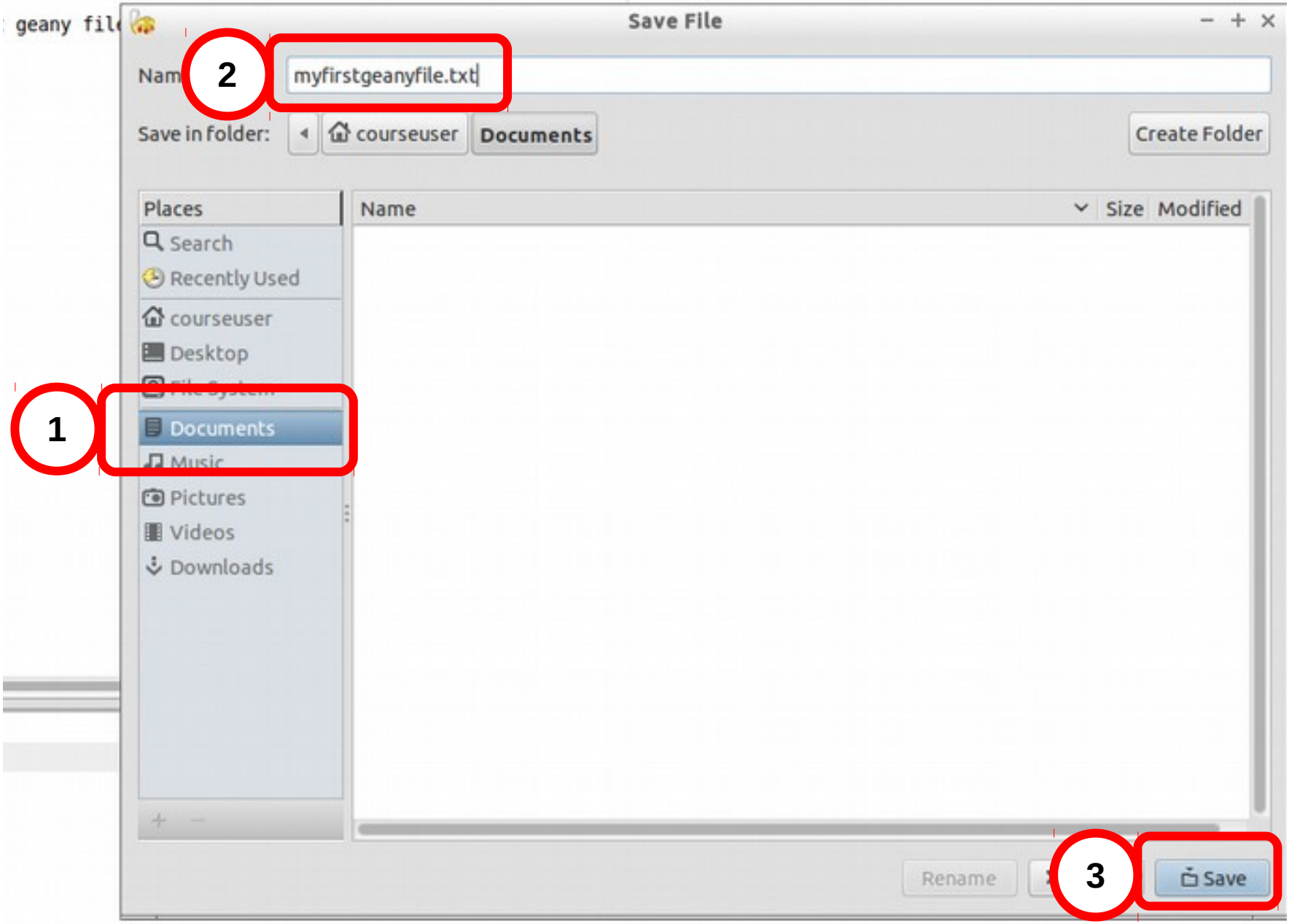
Editing and saving a file: write something



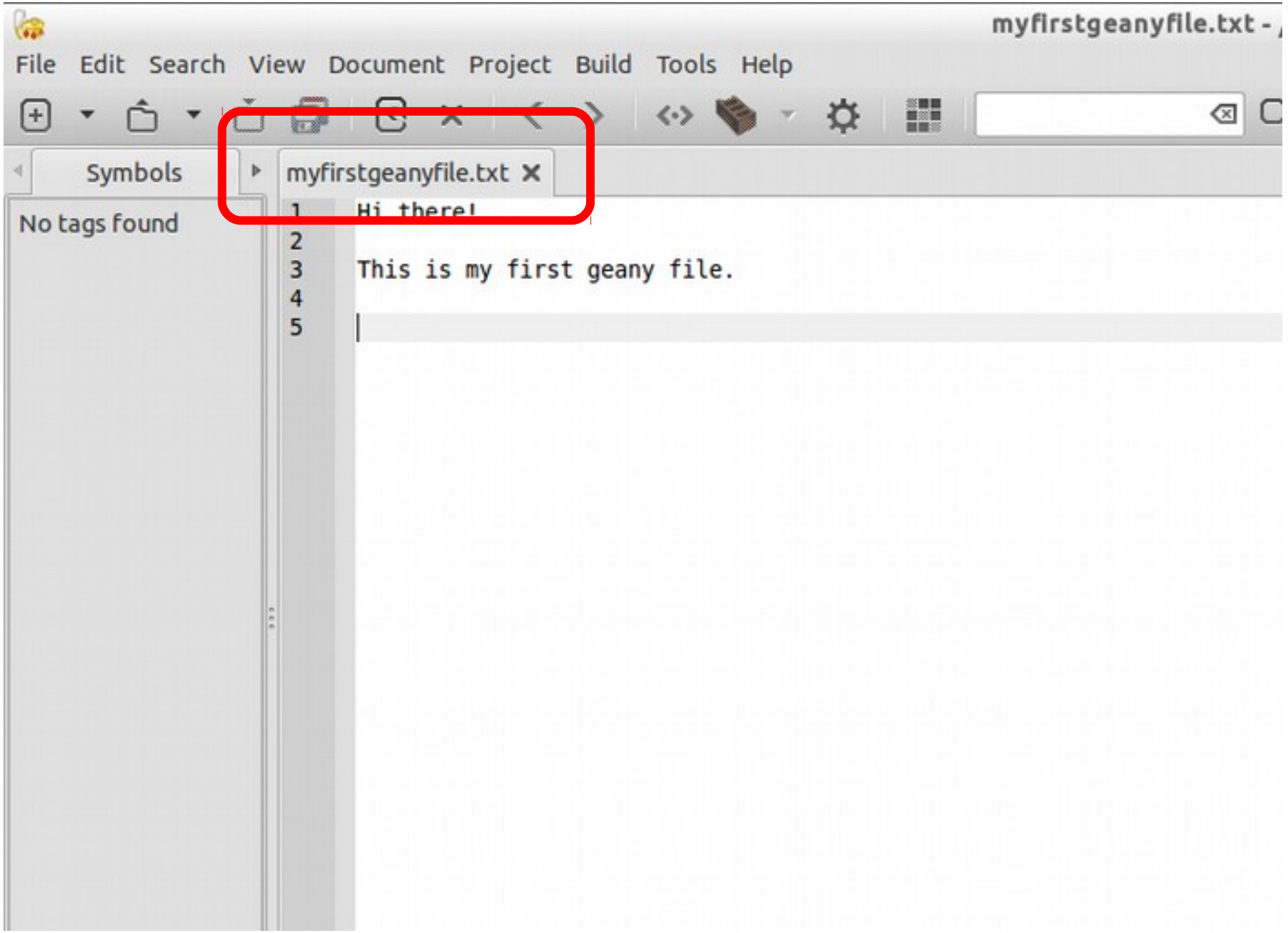
Editing and saving a file: save or save as



Editing and saving a file: choose location and filename



Editing and saving a file



Have fun with the Git-it tutorial!

- Created by jlord, see https://github.com/jlord/git-it-electron
- Contributed by various authors
- Written in JavaScript and HTML using a framework called node.js
- Once done the tutorial shows you some other useful commands and tools.

Setting your default editor with git

- If you commit without the -m option, git will automatically open a text editor for you to write a commit comment.
- It is good practice to write a commit title, leave a blank line, and describe your commit in more detail.
- We will use geany as the default editor, but you can use any editor you like.
- If you don't configure anything, the default is a text editor called **nano**, which for some is a bit weird at first. But I suggest to use it so you just use the command line. Press “CTRL + O” to save the file, ”CTRL + X” to exit.

Setting geany as the default git editor

- Run:

```
git config core.editor geany
```

- Note that the commit will only happen ONCE when you save the file in geany.
- Test by running

```
git commit
```

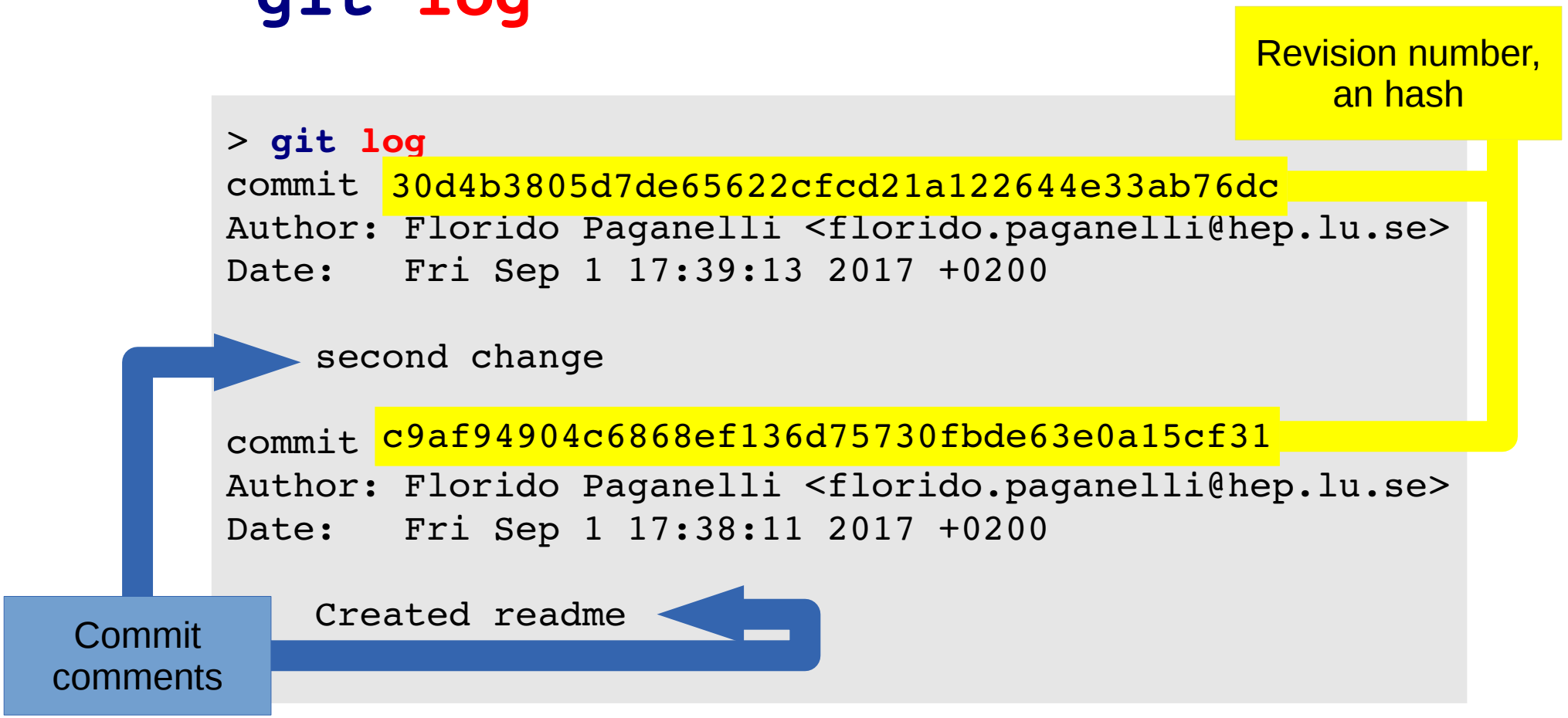
- **If you don't like it, revert to default by writing**

```
git config --unset core.editor
```

Git log, commit history, revision numbers

- All the commit history with you messages can be browsed using the command

  `git log`



Git log, commit history, revision numbers

- To see which files have changed for each commit:

```
git log --name-status
```

Removing or renaming a file

- **Removing**: Sometimes one can decide that files in the directory should not be part of the repository anymore. Rather than deleting them with the rm command, one can use

  ```
  git rm filename
  ```

  - Remove a file using the above command.
  - Check the output of `git status` .
  - `git commit -m 'I have deleted file filename'`
    Remember: CLEARLY STATE that you removed some files in the commit message!

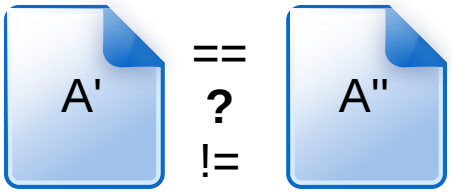
- **Renaming**: `git mv oldfilename newfilename` is equivalent to

  ```
  git rm oldfilename
  ```

  followed by

  ```
  git add newfilename
  ```

Graphical Diffing



- Run

  git diff

```
> git diff
Index: thisisfloridofile.txt
===================================================================
--- thisisfloridofile.txt (revision 6)
+++ thisisfloridofile.txt (working copy)
@@ -1 +1,2 @@
 Hello! this is florido's file.
+I am adding this change.
```

Line numbers of the two files:
-1 : showing line 1 of of file ---
+1,2 : showing lines 1 to 2 of file +++

- If you want a graphical tool to check the diffs, I suggest **meld**

  sudo apt-get install meld

- Use meld as the default diff tool:

  git config diff.tool meld

  git difftool thisisfloridofile.txt

Undoing not committed changes

- Say that we are not happy with the changes we just made **to a single file** and we want to go back to the latest commit (also called HEAD)
- Change one of the files in your repository and issue `git status`.
- The best to do is a **simple checkout** of the file from the last commit
```
git checkout thisisfloridofile.txt
git diff
```
- **Careful! You will lose all the changes done and not committed!!!**
- Note that this is equivalent to checkout the file at the **latest revision HEAD**:
```
git checkout HEAD thisisfloridofile.txt
```
- Checking out HEAD of all files in a directory will cancel all the changes done to the uncommitted files in that directory.
```
git checkout HEAD *
```
- **Play a bit with these commands by changing files and see what happens.**

Reverting to a previous revision

- Say that we don't like the current revision state, and we want to roll back the code to a state of a different revision back in time.
- The main suggestion is:
  try to never go back in the revision history.
  This is actually nice because in a collaborative environment, keeps track of who-did-what with no cheating allowed :)
  Unfortunately git allows for “cheating” by changing the revision history. It can be useful sometimes, but must be used with extreme care. **Changing the revision history gives no UNDO.**
- To experience with this, change some files and commit.

Reverting to a previous revision the safe way: revert

- The **revert** command restores the state of all files at a certain revision to the current working dir.
- Usually the output of a revert gives hints about the steps to take before committing.
- Make sure you have at least three commits (check git log)
- Create a fourth commit

Reverting to a previous revision the safe way: revert

- Try to git revert everything to your second commit in the log:
  ```
  git revert commithash
  ```
- Example:
  ```
  git revert c9af94904c6868ef136d75730fbde63e0a15cf31
  ```
- Read the git status output to see what changed
- Take action to make the files ready for commit, and commit
  - Git will automatically start a commit and open the text editor for you. It will add the “Revert commithash” comment to your commit and wait for your input.

Reverting to a previous revision the unsafe way: reset

- The reset command does something different. It does not preserve history and allows you to modify an existing commit. For a detailed explanation see https://www.atlassian.com/git/tutorials/undoing-changes
- I suggest to use it **only** when one of these two happen:
  - You already staged some changes to a file and you want to unstage them

```
git reset filetounstage
```

  - You are totally unhappy with whatever you did so far and want to **unstage all staged files**:

```
git reset
```

Fixing commit mistakes

- Commit allows you to amend or change the latest commit if, for example, you forgot a file or you wrote the wrong comment:

```
git commit --amend
```

- Note that this will create a new revision hash, but will **DELETE** the previous commit hash.
- See https://git-scm.com/book/id/v2/Git-Basics-Undoing-Things

Importance of git within the course (especially for the final project!)

- **Problem**: the virtual machine disk you're using can be wiped all time, and there is no guarantee the files you left there will be kept.
- **Solution**: From this tutorial on, you're invited to put your code files on your github repository at the end of each tutorial session.
  - Suggestion: create a directory `TutorialXY` in your /username/ git folder for each tutorial
  - You may or may not want to pull request your changes. In some cases this will be requested by the Homework.
- The final course project material you will create can be only handed out using a special git repository we will indicate, so get familiar with git!

Graphical Clients

- Want to try a graphical client?
  - Minimalistic one: in the folder where a git repository exists, run

    `gitg &`

    - Check out how it shows branches!
  - Feature-rich one (not available in repositories): `https://www.gitkraken.com/`
    - This one is NOT available on Lubuntu repositories. You need to download it from the internet if you want the latest version.

Homework Tutorial 2b (HW2b)

1)Create a github account (you should already have it after the tutorial)

2)Fork the repository:
https://github.com/floridop/MNXB01-2017.git

3)Clone the repository you forked.

4)Using the **git remote** command, add:

- your fork repository as the remote *origin*
- My upstream branch https://github.com/floridop/MNXB01-2017.git as the *upstream* remote branch

5) At the root of the repository, create a folder with the first three letters of your name and the surname. For example my name is Florido Paganelli, I created:
flopaganelli

6) In the above folder, create a folder called HW2a and upload the homework Oxana assigned to you yesterday.

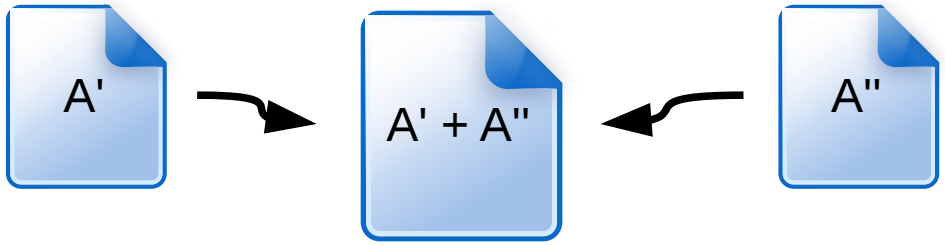
7)Add the new files and **commit. Remember to write an explicative comment in the commit. Stupid comments will be rejected.**

8) push to the remote *origin* and submit me a **pull request** on github.

9)Copy the link of your github fork and a link to your pull request on Live@Lund.

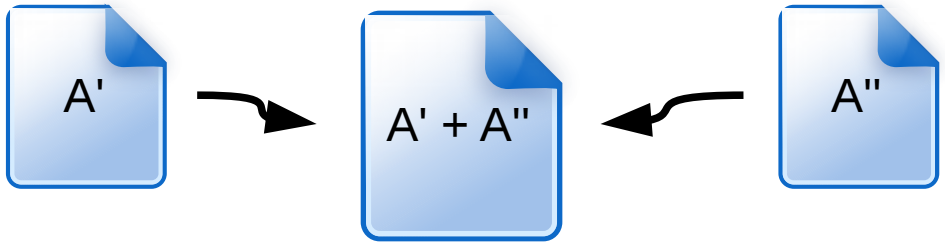
Advanced topics

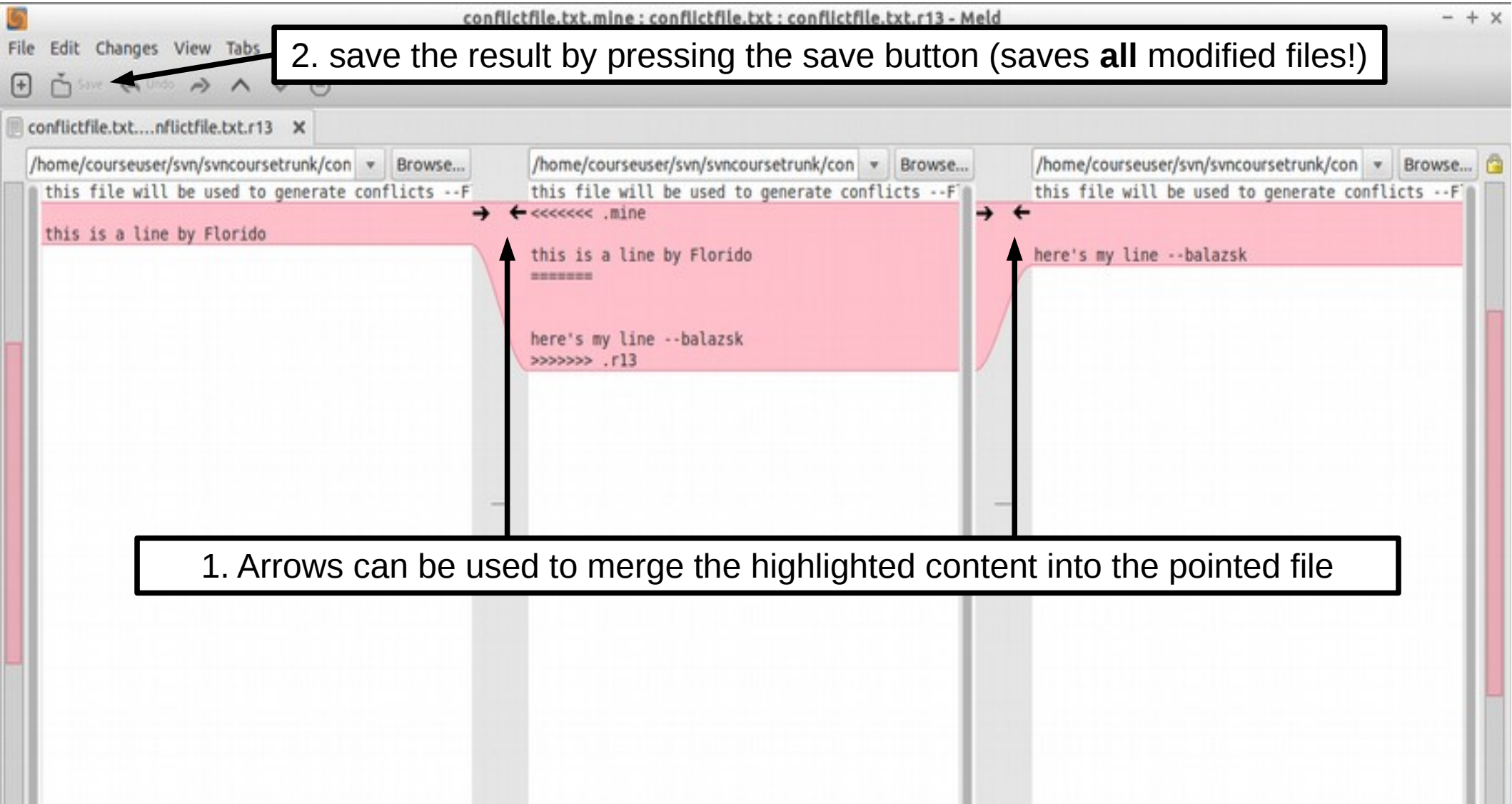
Merging



- Suppose we have two versions of a document with different contents
- We want to make one out of two
- This is often referred as three-way-merge
- We need to choose which part of each document we want to keep
- There exist tools to do it, for example the excellent `meld`
- git can attempt to do merges for us:
  - If the merges are simple, i.e. the changed content of A' can be easily mixed with that of the content of A''. For example, the documents differ a little but the changes in each document are not overlapping.
  - If we provide it with some hint on how to do the merges
  - If the above fail, it will ask us to do the merge manually, for example using `meld`
- The most frequent case of merge is in case of conflicts, we will not see them in this course.

Merging with `meld`





References

- git cheat sheets:
  https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf
  https://jan-krueger.net/wordpress/wp-content/uploads/2007/09/git-cheat-sheet.pdf
  https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet
- Quick guide to githttp://rogerdudler.github.io/git-guide/
- Jlord's git-it:
  https://github.com/jlord/git-it-electron
- Merging with meld
  https://www.youtube.com/watch?v=3Qynj8WUwgs
- Reverting
  https://www.atlassian.com/git/tutorials/undoing-changes
  https://git-scm.com/book/id/v2/Git-Basics-Undoing-Things

Pictures references

- https://openclipart.org/
- http://www.libreoffice.org/