Version Control: git

What is version control

We already use some kind of version control

by saving both old and new (modified) copies.

- DanaResumeSeptember2015.pdf
- DanaResumeJanuary2016.pdf

  or

- html_exercise_old.html
- html_exercise_new.html

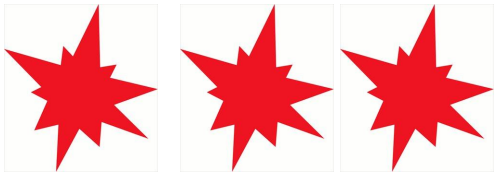
What is version control

Good for papers, maybe class projects, but not for large software projects!

Consider:

- Google - 2,000,000,000 lines of code
- Facebook - 75,000,000 lines of code
- Counter Strike - 1,000,000 lines of code

If some modification causes a code loss, there will be a disaster in a company !!!



Why need a version control

- Backup and Restore
- Synchronization - share files
- Short-term Undo
- Long-term Undo
- Track Changes
- Track Ownership
- Sandbox
- Branching and Isolation

Terms

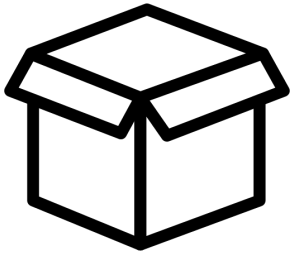
- **Repository:** the “database” storing files
- **Server:** the computer storing to the repo
- **Client:** the computer connecting to the repo
- **Working set/Working copy:** your local directory of files, where you make a copy
- **git:** version control system, very popular

Example: create a repo

Folder where you put your project

Example: create a repo

git init

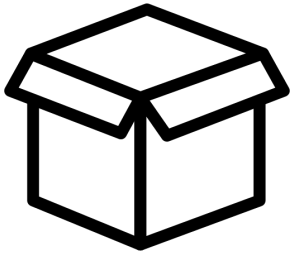


Folder where you put your project

New repository for your project

Example: create a repo

Working copy:
here you add and
modify files

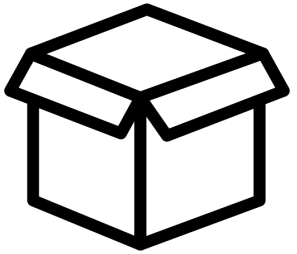


Repository:
stores all versions of
your project

Example: adding file

Working copy

New file
file.txt

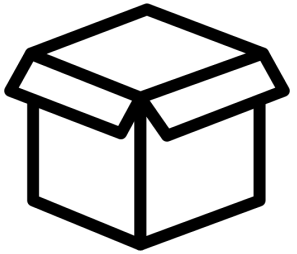


Repository

Example: adding file

Working copy

New file
file.txt

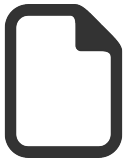


Repository

DOES NOT KNOW ANYTHING ABOUT NEW FILE!!!!

Example: adding file

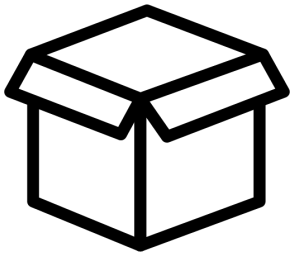
Working copy



file.txt

git add file.txt

Staging area

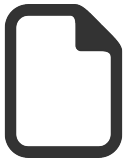


Repository

DOES NOT KNOW ANYTHING ABOUT NEW FILE!!!!

Example: adding file

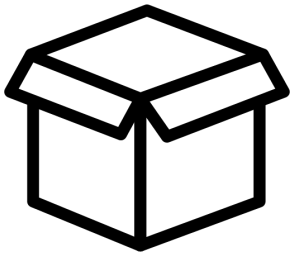
Working copy



file.txt

Staging area

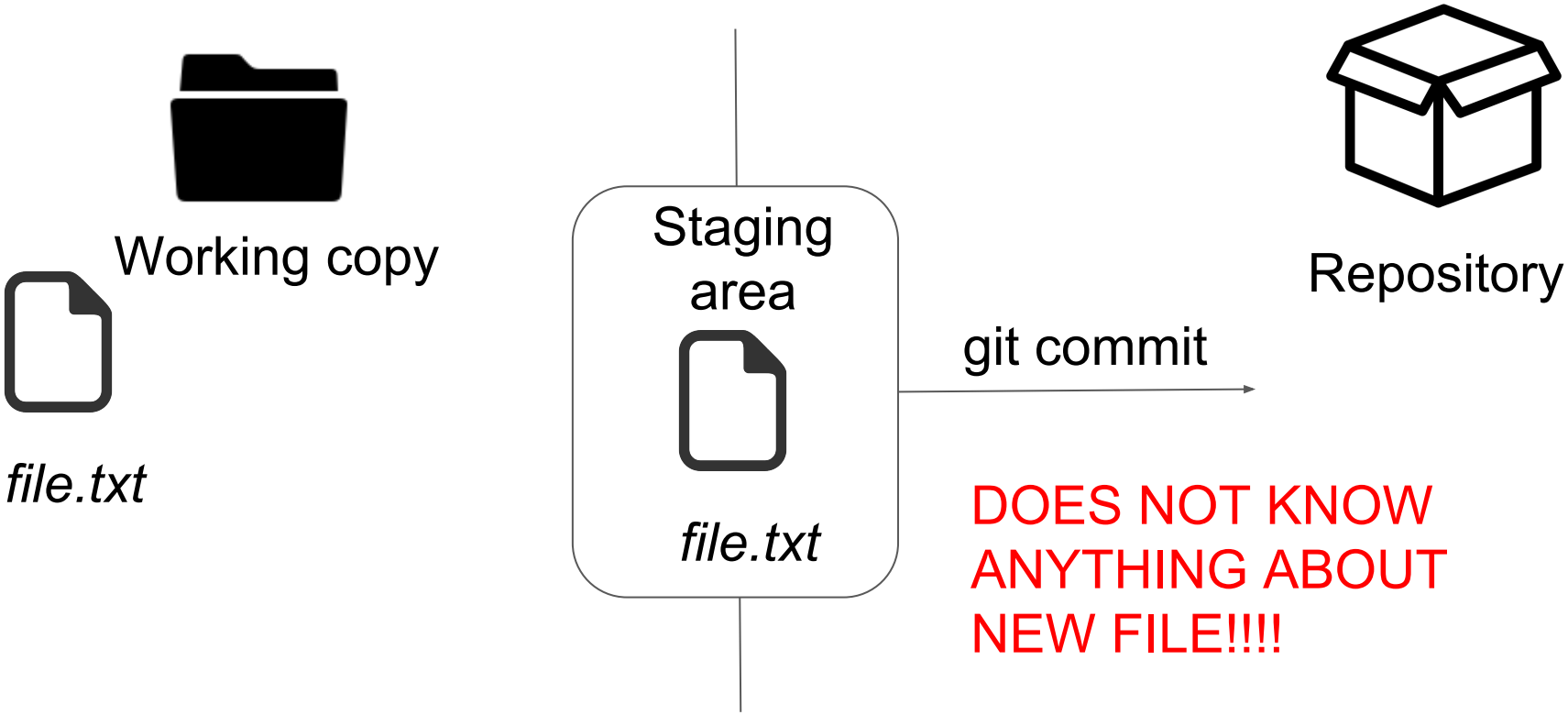
file.txt



Repository

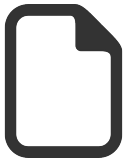
DOES NOT KNOW ANYTHING ABOUT NEW FILE!!!!

Example: committing changes



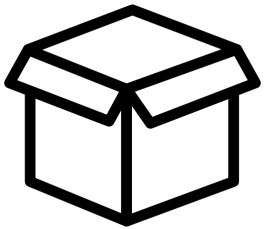
Example: committing changes

Working copy



file.txt

Staging area

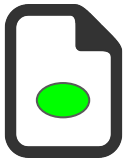


Repository

file.txt

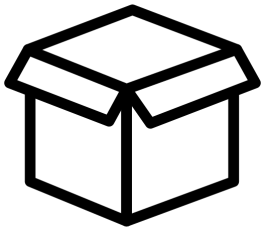
Example: making changes

Working copy



file.txt

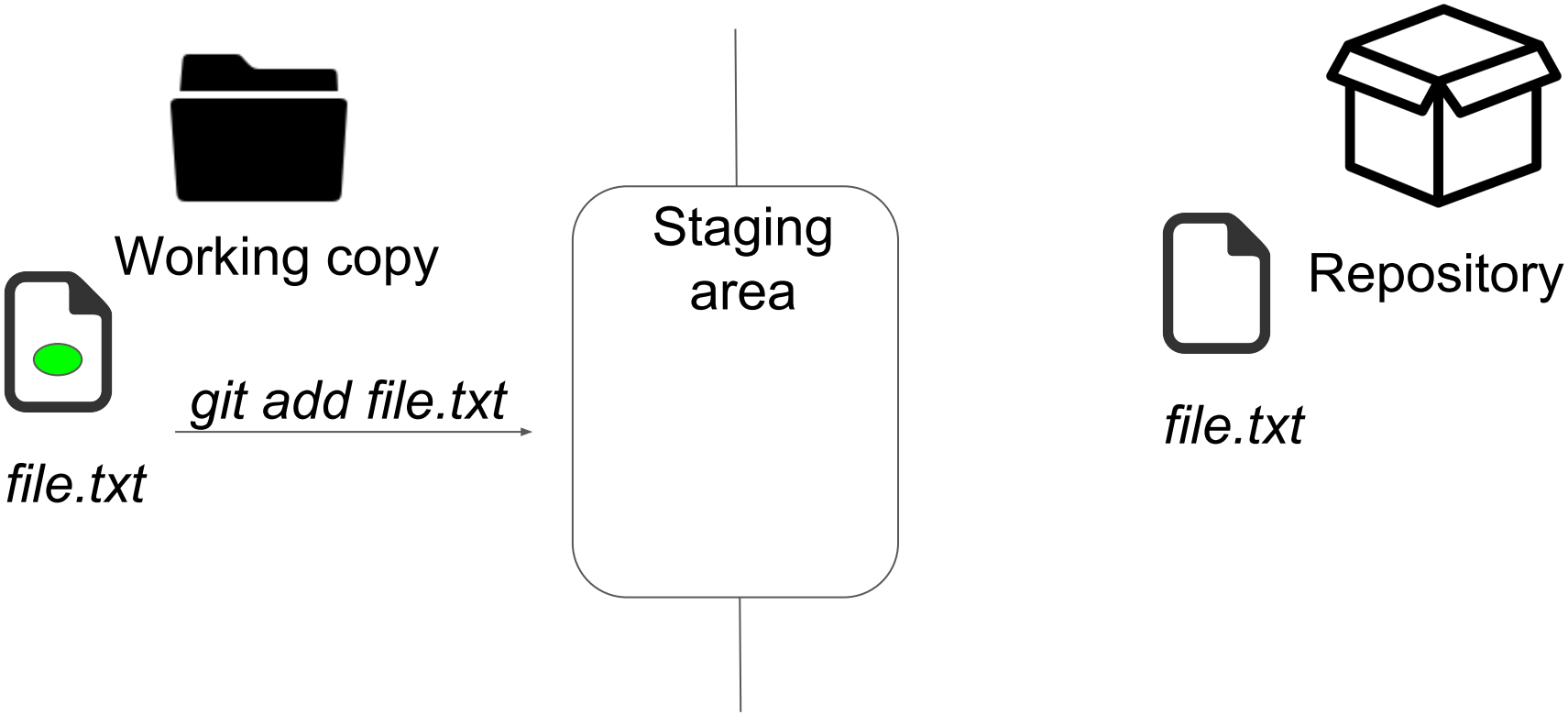
Staging area



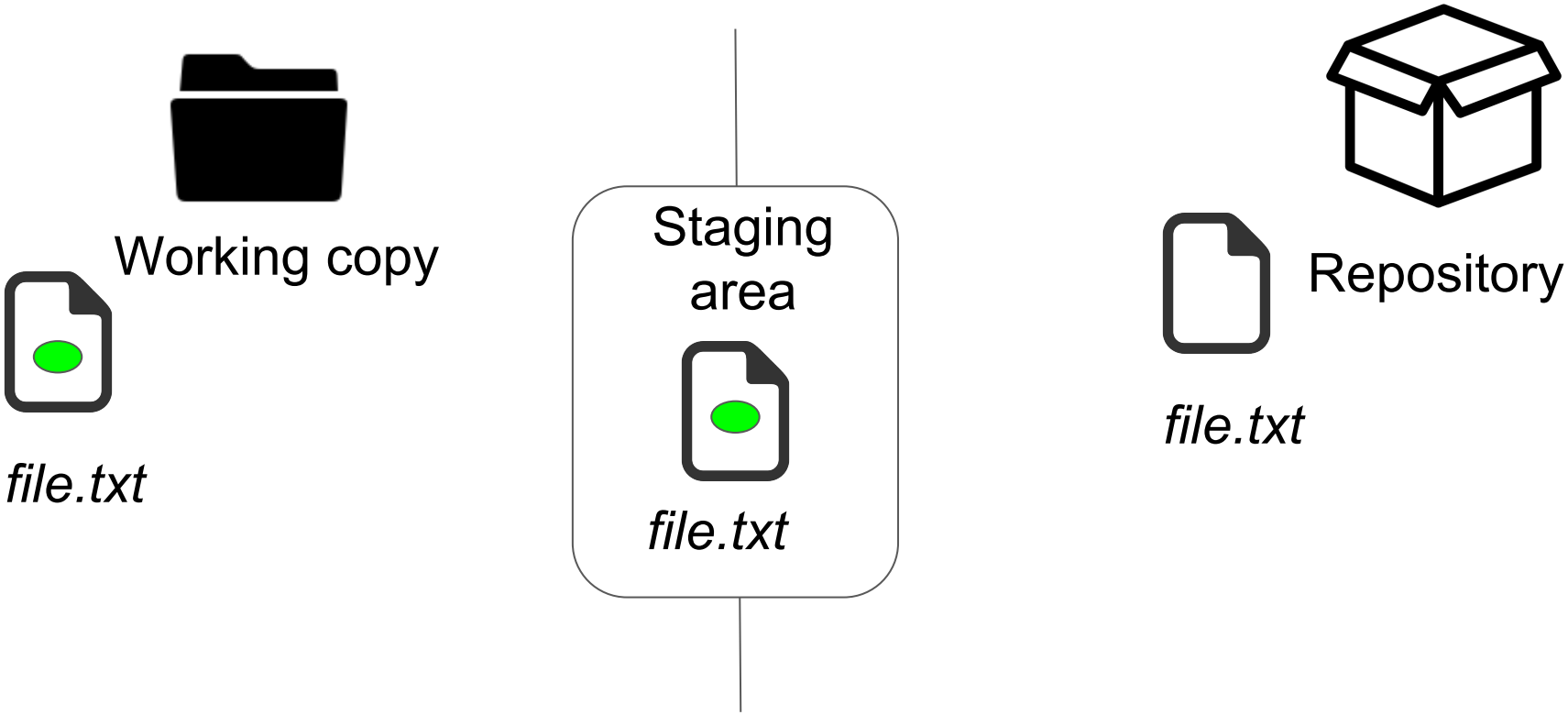
Repository

file.txt

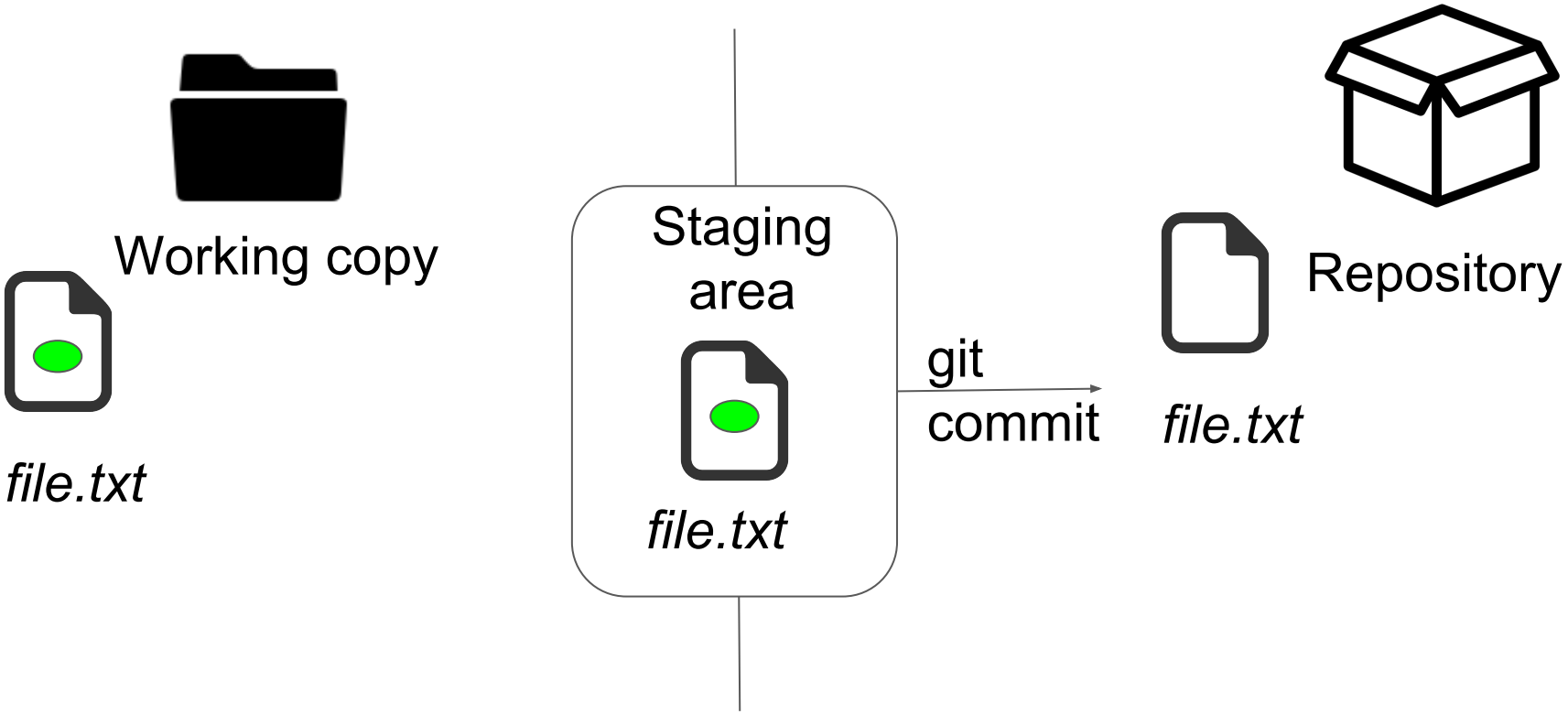
Example: adding changes



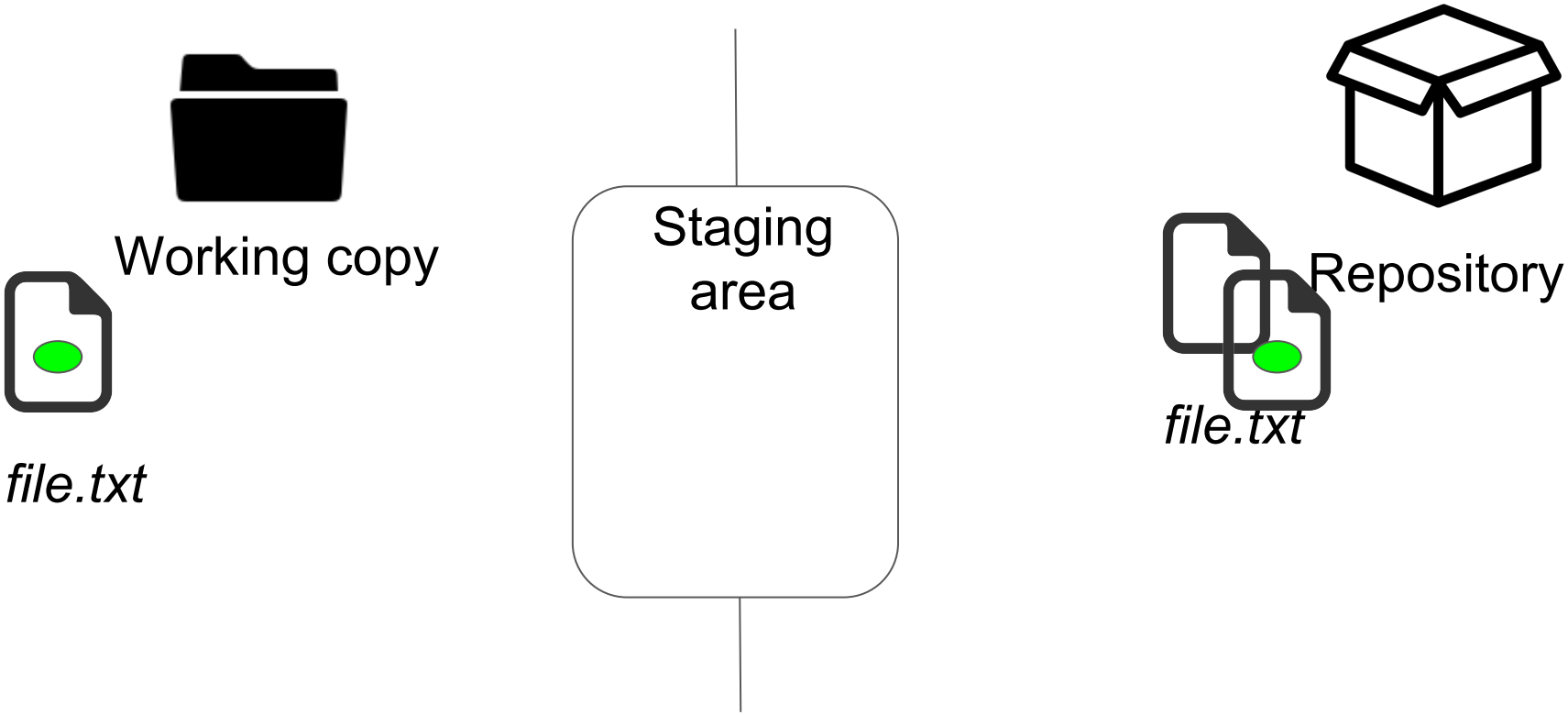
Example: adding changes



Example: committing changes



Example: making changes



git

1

Working Directory

Make changes to files:

+ additions
- deletions
  modifications

2

Staging Area

Bring changes into the staging area

3

Repository

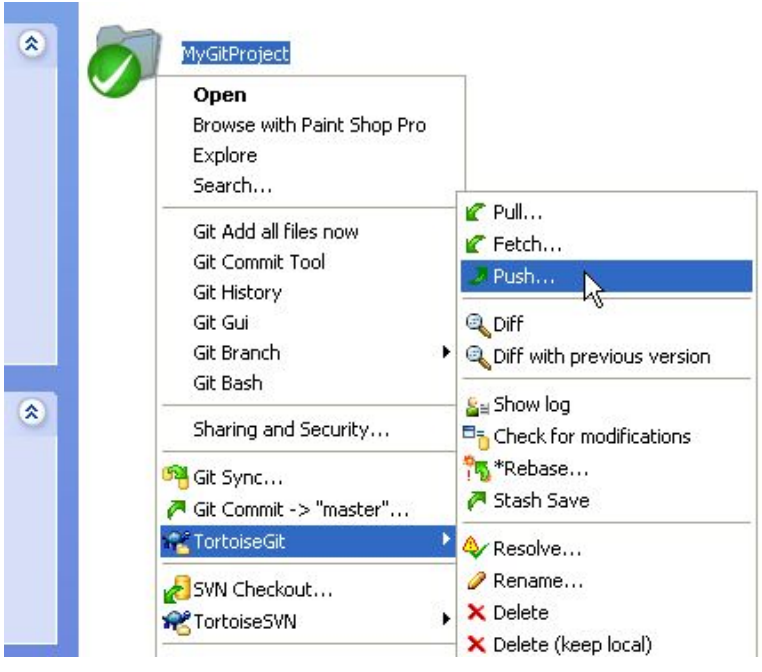
Save changes to the repository as a 'commit'

git important commands

- Create repository
- Add
- Commit

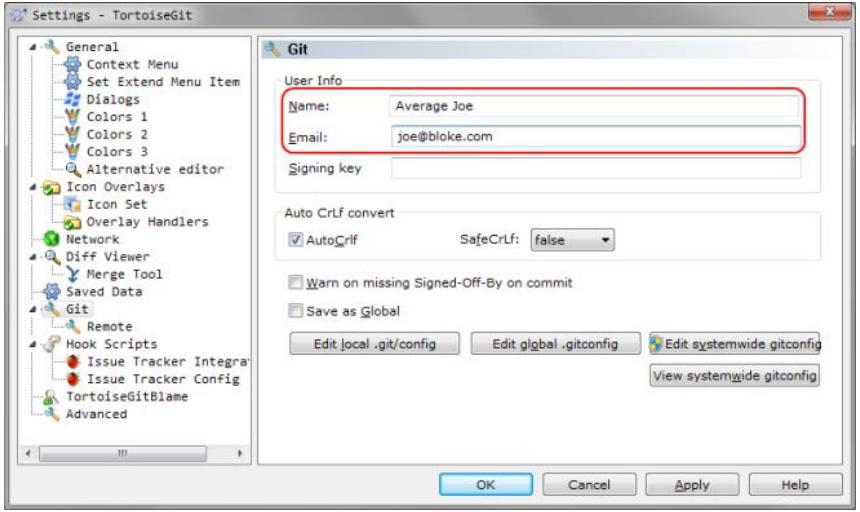
TortoiseGit

TortoiseGit is a convenient Graphic User Interface to work with git version control system.



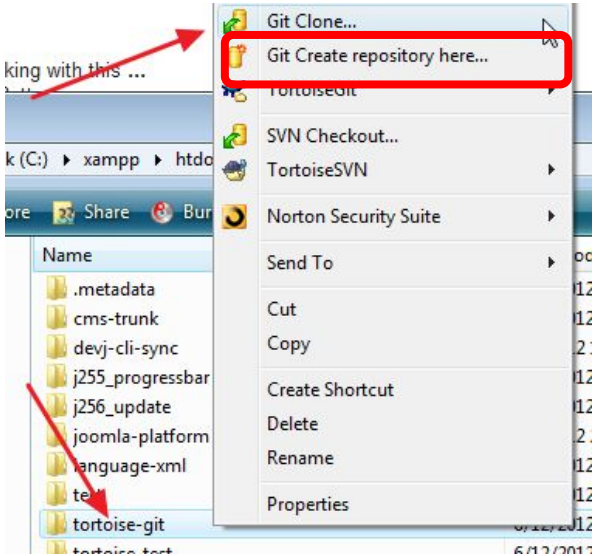
TortoiseGit: setup

1. You should input your username and email:
   this way you and your collaborators will see who made which changes



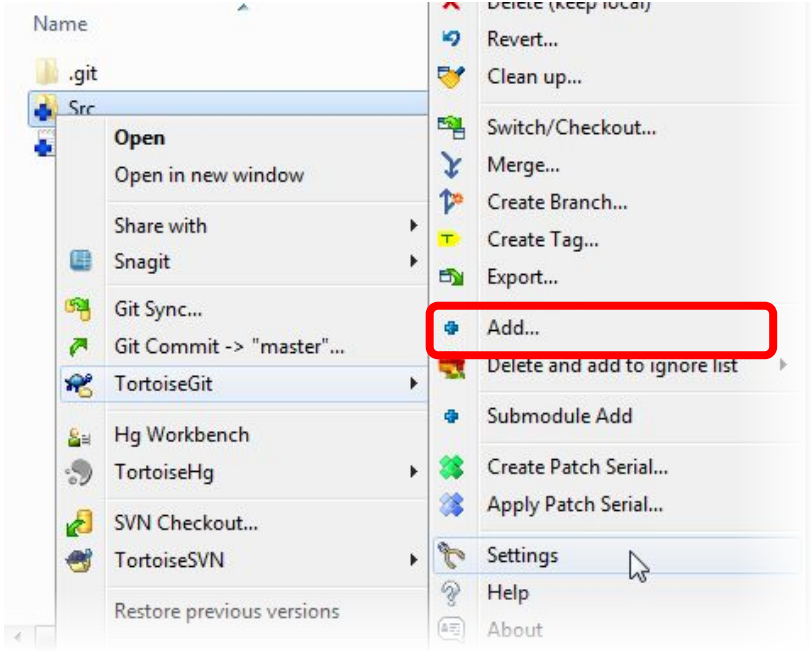
TortoiseGit: create repo

2. Right click on the folder with your project, and select “Git Create repository here...”



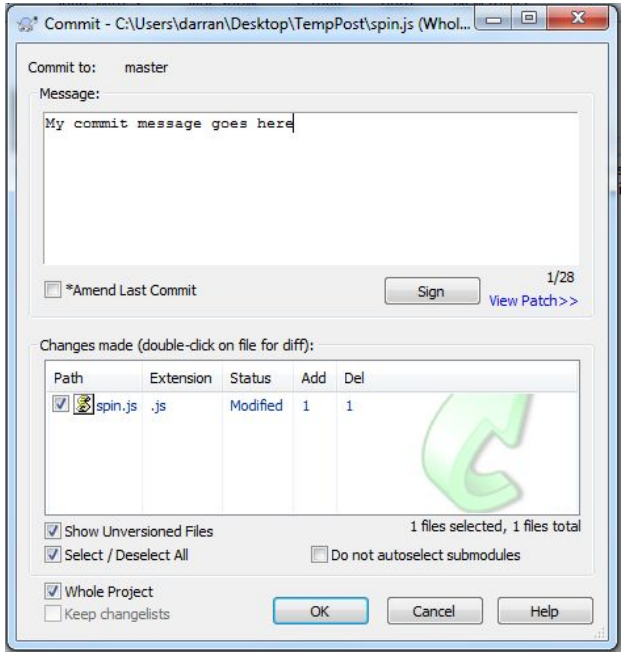
TortoiseGit: add files

3. When you add a new file/change existing files:



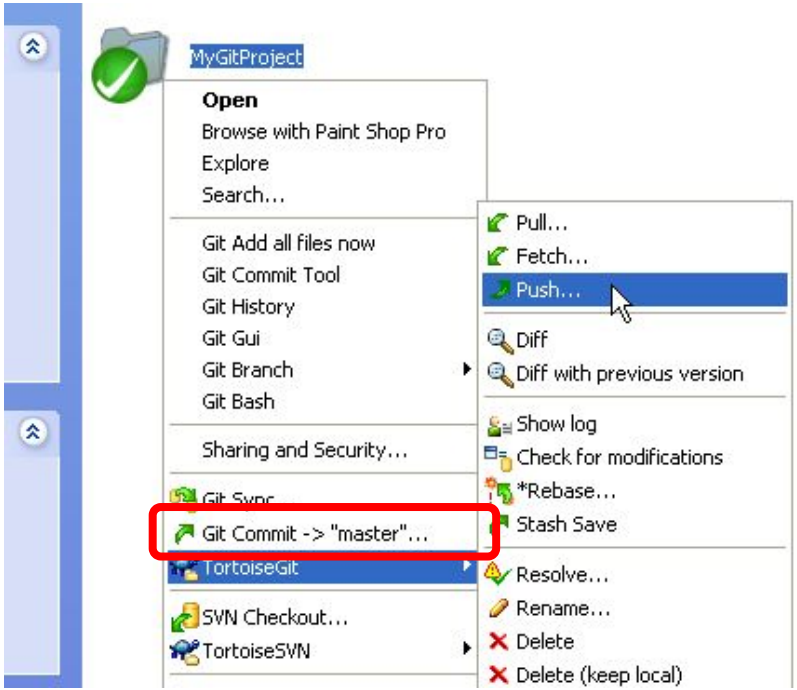
TortoiseGit: commit changes

4. To commit changes in the staging area, you can do it right after clicking on “Add”:



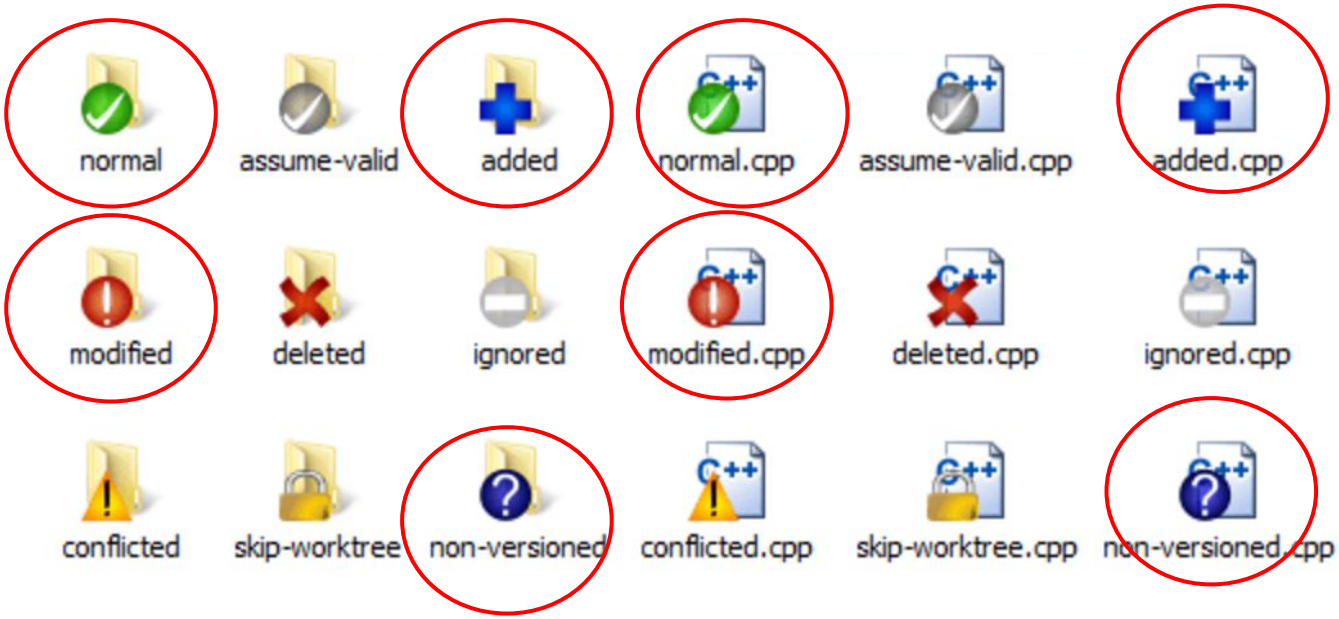
TortoiseGit: commit changes

… Or commit everything together later!



TortoiseGit: status

Status show which files were/were not added/committed



TortoiseGit: commit SHA

Commits are enumerated by specific string, called SHA.

It is something like a 40-digit “object name” and looks like:
`6ff87c4664981e4397625791c8ea3bbb5f2279a3`

The string depends on the version content and is a result of hashing (applying specific cryptographic function) the content.

SHA - Secure Hash Algorithm

TortoiseGit: .git folder

Notice .git folder inside your repository!

```
.
|-- COMMIT_EDITMSG
|-- FETCH_HEAD
|-- HEAD
|-- ORIG_HEAD
|-- branches
|-- config
|-- description
|-- hooks
|   |-- applypatch-msg
|   |-- commit-msg
|   |-- post-commit
|   |-- post-receive
|   |-- post-update
|   |-- pre-applypatch
|   |-- pre-commit
|   |-- pre-rebase
|   |-- prepare-commit-msg
|   `-- update
|-- index
|-- info
|   `-- exclude
|-- logs
|   |-- HEAD
|   `-- refs
|-- objects
`-- refs
    |-- heads
    |-- remotes
    |-- stash
    `-- tags
```

TortoiseGit: exercises

1. Create a directory “github_exercises”.
2. Create a repository for your directory.
   What happens in the file system?
3. Go to Settings and enter your email and name!
4. Create two random files inside your directory, call them **f1**, **f2**.
   What happens in the file system GUI?
5. Add the **f1** to the staging area.
   What happens in the file system GUI?
6. Modify **f1** by removing some character.
   What happens in the file system GUI?
7. Add the **f2** to the staging area..
   What happens in the file system GUI?
8. Commit the changes.
   What version of file **f1** do you expect to find in your repository?

TortoiseGit: undoing

HEAD - current (most recent) commit

To discard changes:

git checkout HEAD
or
git checkout HEAD *filename*

To unstage files but keep changes:

git reset HEAD
or
git reset HEAD *filename*