SAPIENZA
UNIVERSITÀ DI ROMA

Robot Programming

GIT

Giorgio Grisetti

Git

- Created by Torvalds,
  father of Linux, in 2005
  - born within the linux community
  - thought for kernel development

- Goals of Git:
  - speed
  - Support for non-linear development
    (thousands of parallel branches)
  - Be distributed
  - Capable of supporting very large projects

(a "git" is a grumpsy elderly guy. Linus meant himself)

Installing/Learning Git

- Git website: http://git-scm.com/
  - Free on-line book: http://git-scm.com/book
  - Reference page for Git: http://gitref.org/index.html
  - Git tutorial: http://schacon.github.com/git/gittutorial.html
  - Git for Computer Scientists:
    - http://eagain.net/articles/git-for-computer-scientists/

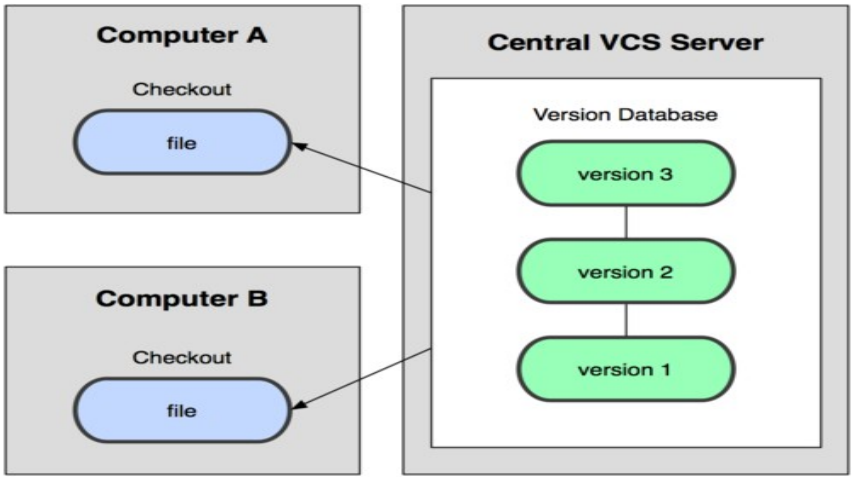
- Command line: *(*where *verb = config, add, commit, etc.)*
  - `git help` *`verb`*

Centralized VCS

- In Subversion, CVS, Perforce, etc.
  a central server (repo)
  keeps the "official copy" of the source
  - the server keeps the only history
    of the repo

- The used performs "checkouts"
  of the repo in his local copy
  - local modifications are not registered
    (versioned)

- When a feature is added, the user performs
  a “check in” to the server
  - the check in increments the version number (and is registered)

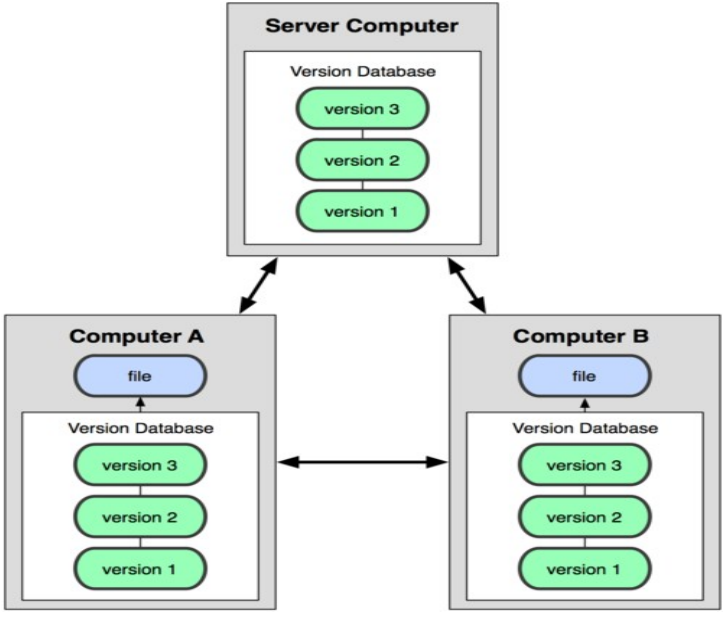


Distributed VCS (Git)

- In git, mercurial, etc., the user does not do a "cheeckout" from a central repo, nistead he "clones" from a server

- La local copy is complete, and stores all what is on the remote server
  - local copy as good as remote one

- The operations are **local**:
  - check in/out from a *local* repo
  - commit changes in a *local* repo
  - the *local* repo keeps his history

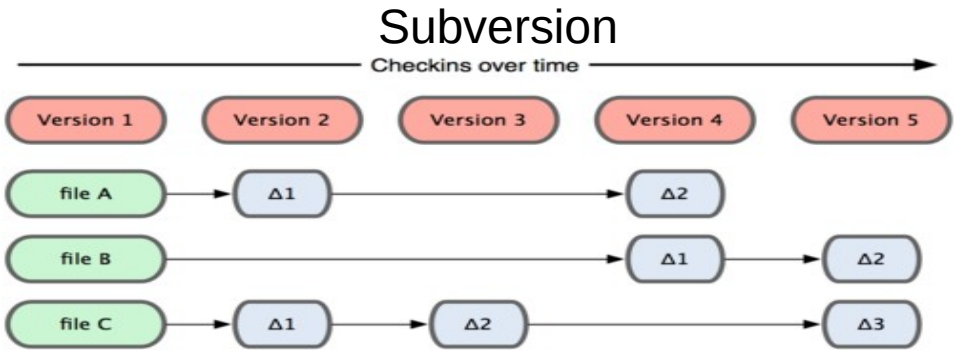


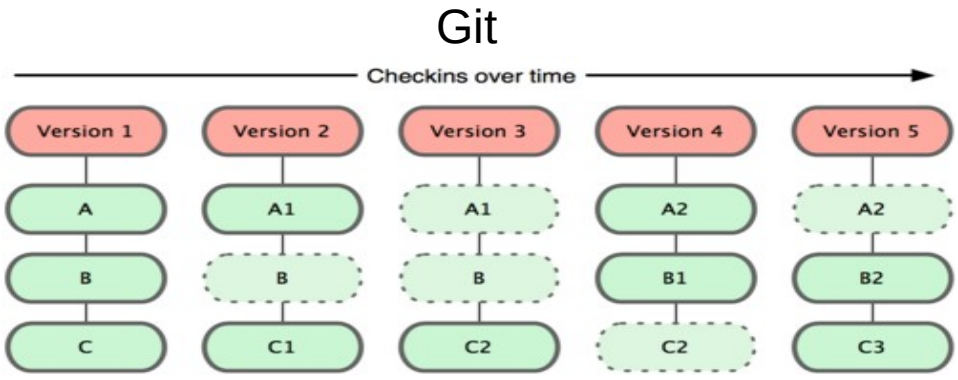
- When ready you can "push" the changes to the server and synchronize the two repos

Git snapshots

- Centralized VCS
  track the version of each
  individual file.

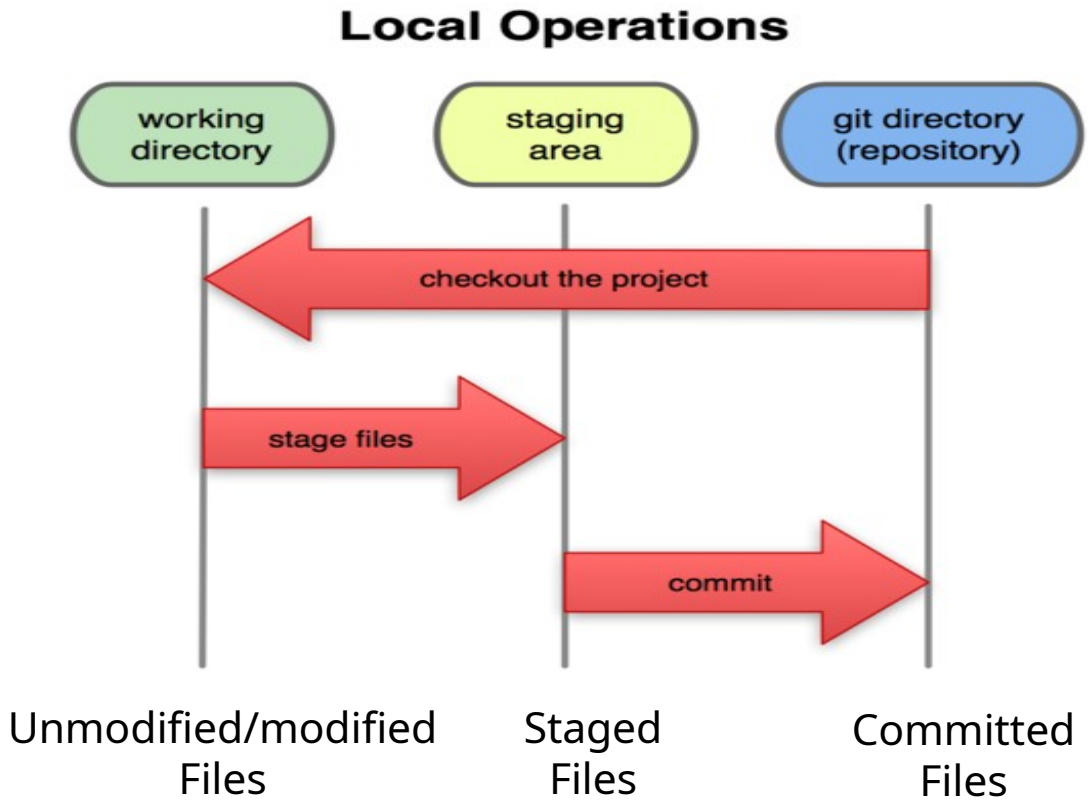
- Git keeps "snapshots"
  of the entire project
  - each version holds all code of all files
  - between checkins, some files
    stay the same, others change
  - redundant, but faster





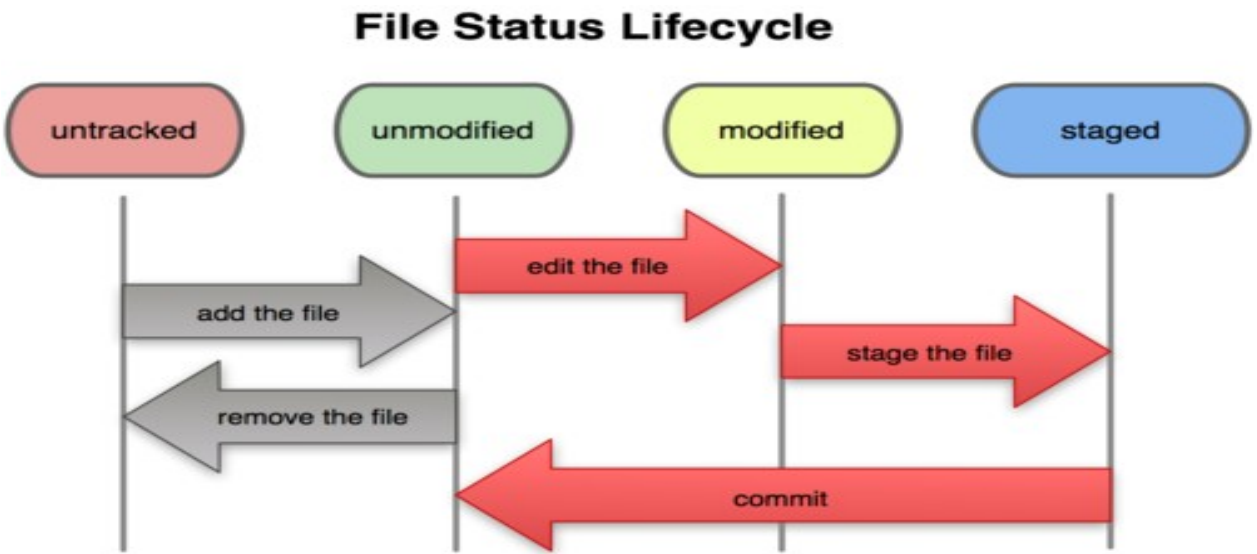
Local Repos

- In your local copy, the files can be:
  - in the repo (committed)
  - in the repo and modified (working copy)
  - in an intermediate status, the **"staging" area**
    - staged files are ready to be committed, but the effective commit has not been done yet
    - performing a commit saves them on the repo and assigns a version



Git workflow

- **Modify** the files in the working directory
- **Stage** the files you want to keep by adding snapshots (add/delete)
- **Commit:** takes the staged files and stores them in a local repo, permanently



Initial Configuration

- Set the user mail and credentials to say git who you are:
  - `git config --global user.name "Bugs Bunny"`
  - `git config --global user.email bugs@gmail.com`
  - invoke `git config –list` to verify they are correct

- Choose the editor to use for the commit messages:
  - `git config --global core.editor emacs`
    - (it is vim by default)

Git commit checksums

- In Subversion each change to the central repo increments the version number.
  - In Git, each user has his copy of the repo and commits the change in his local copy **before** sending it to the central repo
  - Git generates a unique hash **SHA-1**(40 character string of hex digits) for each commit.
  - A commit is characterized by its ID, instead of version number.

  - often we see only the first 7 digits:
    - `1677b2d Edited first line of readme`
    - `258efa7 Added line to readme`
    - `0e52da7 Initial commit`

Creating a git repo

Two common scenarios: (alternative)

- To create a local repo in the current directory:
  - `git init`
    - this creates a directory `.git` in the current directory.
    - now you can make commit of the files in the directory
  - `git add` *`filename`* `#` `adds a file to the staging area`
  - `git commit –m "`*`commit message`*`" # manda sends the file`

- To **clone a remote repo** to your current directory:
  - `git clone` *`url localDirectoryName`*
    - Creates
      - the local directory, storing a copy of the files in the remote repo
      - a .git directory storing the git infos

Adding a file

- The **first time** we tell the system to track the file, **before each commit**, need to add the file to the staging area
  - `git add Hello.java Goodbye.java`

- We permanently store the changes by creating a snapshot:
  - `git commit –m "Fixing bug #22"`

- To remove a file from the staging area *before* a commit we do
  - `git reset HEAD -- filename` (unstages the file)
  - `git checkout -- filename` (undoes your changes)
  - All these commands operate on the local copy

Showing/Reverting Changes

- To view status of files in working directory and staging area:
  - `git status` **or** `git status –s` (short version)
- To see what is modified but unstaged:
  - `git diff`
- To see a list of staged changes:
  - `git diff --cached`
- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

    - `git log -5` (to show only the 5 most recent updates), etc.

Commands

command	description
`git clone` ***`url [dir]`***	copies a remote repo from **url** to **dir**
`git add` ***`file`***	adds a file to the staging area
`git commit`	registers the shapshot in the staging area
`git status`	shows the status of the files in the working directory and in the staging area
`git diff`	shows the file differences between what is staged and what has been modified but not staged
`git help` *`[`****`command`***`]`	shows the help for a command
`git pull`	synchronizes the local repo FROM the remote
`git push`	synchronizes the remote repo FROM the local
other commands: `init, reset, branch, checkout, merge, log, tag`	

Branching and Merging

Git uses branching heavily to commute between tasks

- Creating a new local branch:
  - `git branch` *`name`*
- Show the actual branch: (* = current branch)
  - `git branch`
- switch to a specific local branch:
  - `git checkout` *`branchname`*
- Merging the changes from a branch to the main branch (master):
  - `git checkout master`
  - `git merge` *`branchname`*

Conflicts

- Conflicting files will contain sections `<<< ... >>>`, indicating the points where git could not resolve the conflict on his own.

```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

branch 1's version

branch 2's version

- To resolve a conflicts you will have to seek for all these sections to bring the file in a correct status.

github bitbucket gitlab

- Sites offering storage and an web interface to host git repos.
  - **You can create a remote repo and push your local branch, so that others can cooperate**.
  - They are used in plenty of open source projects (and also closed source)
  - They are for free

- *Question: do I absolutely need to use one of these services*?

  Answer: NO you can either:
  - use git locally
  - install a private server at your home or your office.

Interacting with a Remote Repo

- **Pull** from remote the last changes
  - (resolve conflicts if needed and add/commit them)
  - `git pull origin master`

- **Push** to remote your version
  - `git push origin master`

Git: Home server demo

Setting up a VPN at home and using an old PC lets you having a private GIT server accessible from wherever you are

- Wireguard (default on the router of some provider)
- Ssh server on the PC
- Git user
- Share ssh key for git user

Git: Home server demo

On server

```
mkdir my_repo.git
cd my_repo.git
git init --bare
```

Git: Home server demo

On client

```
mkdir my_repo
cd my_repo
git init
git remote add origin git@<server>:path_to_git
git push --set-upstream origin master
```