Mini Git Tutorial

http://git-scm.com/

Cristian Lucchesi <cristian.lucchesi@iit.cnr.it>, @criluc

https://github.com/criluc/mini-git-tutorial

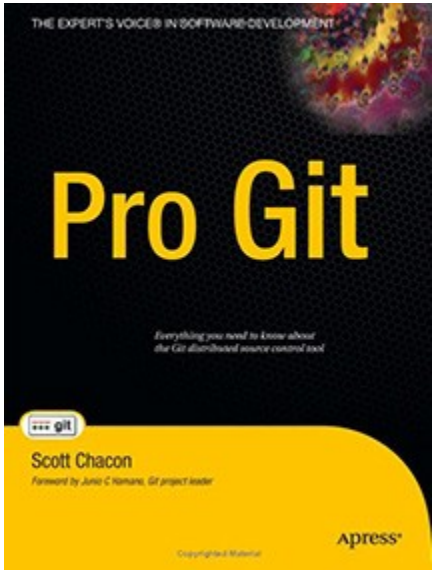
22 novembre 2012

Consiglio Nazionale delle Ricerche

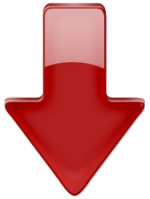
Istituto di Informatica e Telematica

Disclaimer

- Tutorial arranged in one day... from a Git end user
- Strongly based on:
  - Pro Git book, written by Scott Chacon and published by Apress is available to read online for free http://git-scm.com/book



Company using Git



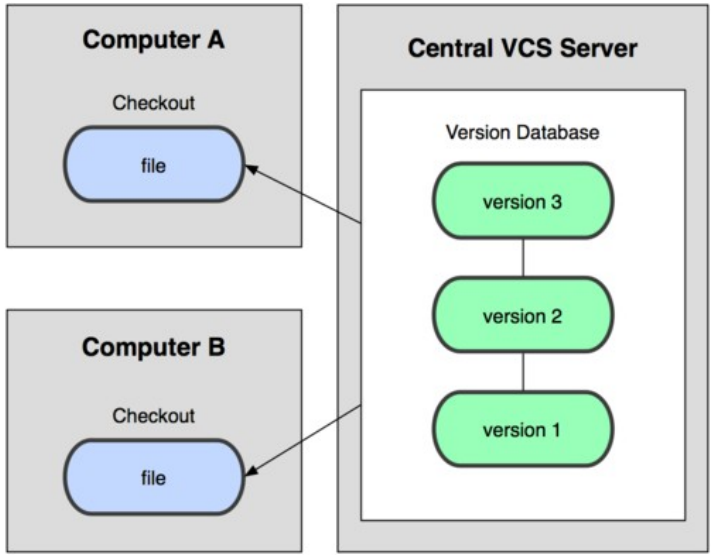
Istituto di Informatica e Telematica del CNR

Version Control System

- a system that records changes to a file or set of files over time so that you can recall specific versions later.
- It allows you:
  - to revert files back to a previous state
  - revert the entire project back to a previous state
  - compare changes over time
  - see who last modified something that might be causing a problem
  - who introduced an issue and when, and more.
  - using a VCS also generally means that if you screw things up or lose files, you can easily recover
  - you get all this for very little overhead.

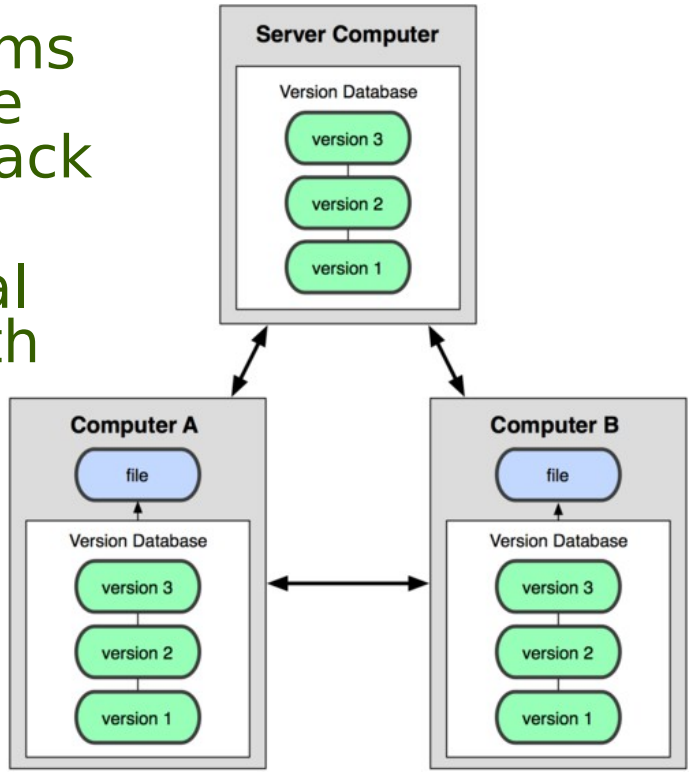
Centralized VCS (CVS,SVN..)

- have a single server that contains all the versioned files
- clients check out files from that central place
- everyone knows to a certain degree what everyone else on the project is doing
- administrators have fine-grained control over who can do what
- it’s far easier to administer
- single point of failure that the centralized server represents
- whenever you have the entire history of the project in a single place, you risk losing everything



Distribuited VCS (Git, Mercurial, Bazar, ...)

- clients don’t just check out the latest snapshot of the files: they fully mirror the repository
- if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it
- deal pretty well with having several remote repositories it can work with
- allows you to set up several types of workflows that aren’t possible in centralized systems, such as hierarchical models
- you need a tutorial like this to use it :-)

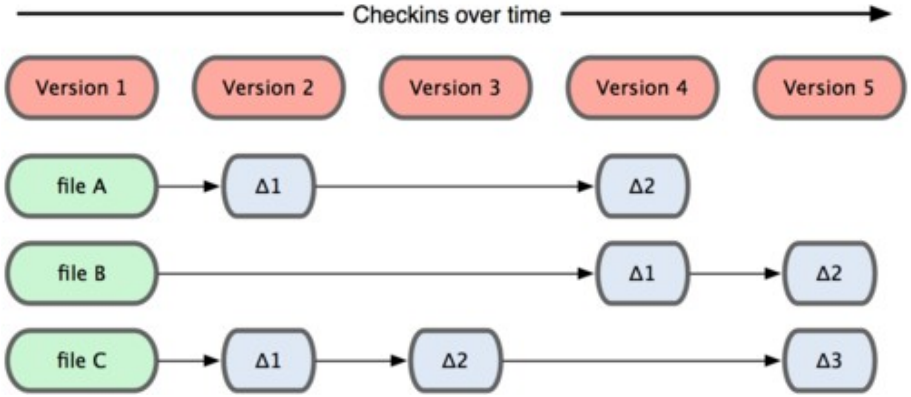


Short History of GIT

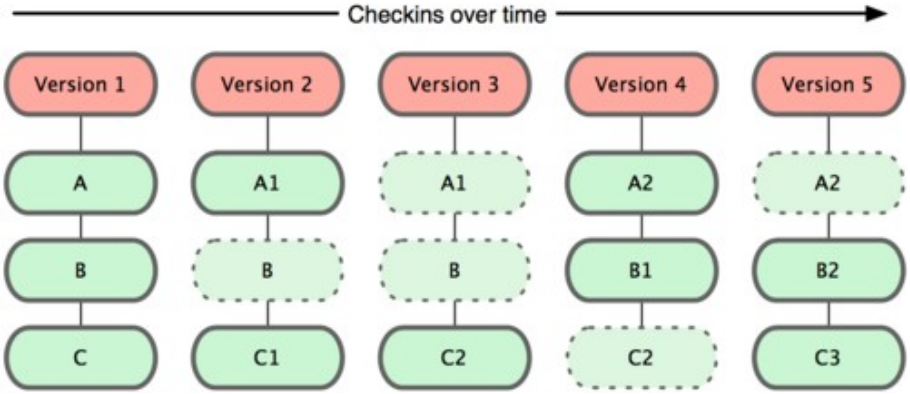
- Developed since 2005 from Linux development community (and in particular Linus Torvalds, the creator of Linux)
- the goals of the new system were:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

Git Basics

- other systems: store information as a list of file-based changes



- Git: its data more like a set of snapshots of a mini filesystem



Git Has Integrity

- everything is check-summed before it is stored and is then referred to by that checksum
- checksumming using a SHA-1 hash
- is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure in Git
- something like this:
  - 24b9da6552252987aa493b52f8696cd6d3b00373
- Git stores everything not by file name but in the Git database addressable by the hash value of its contents

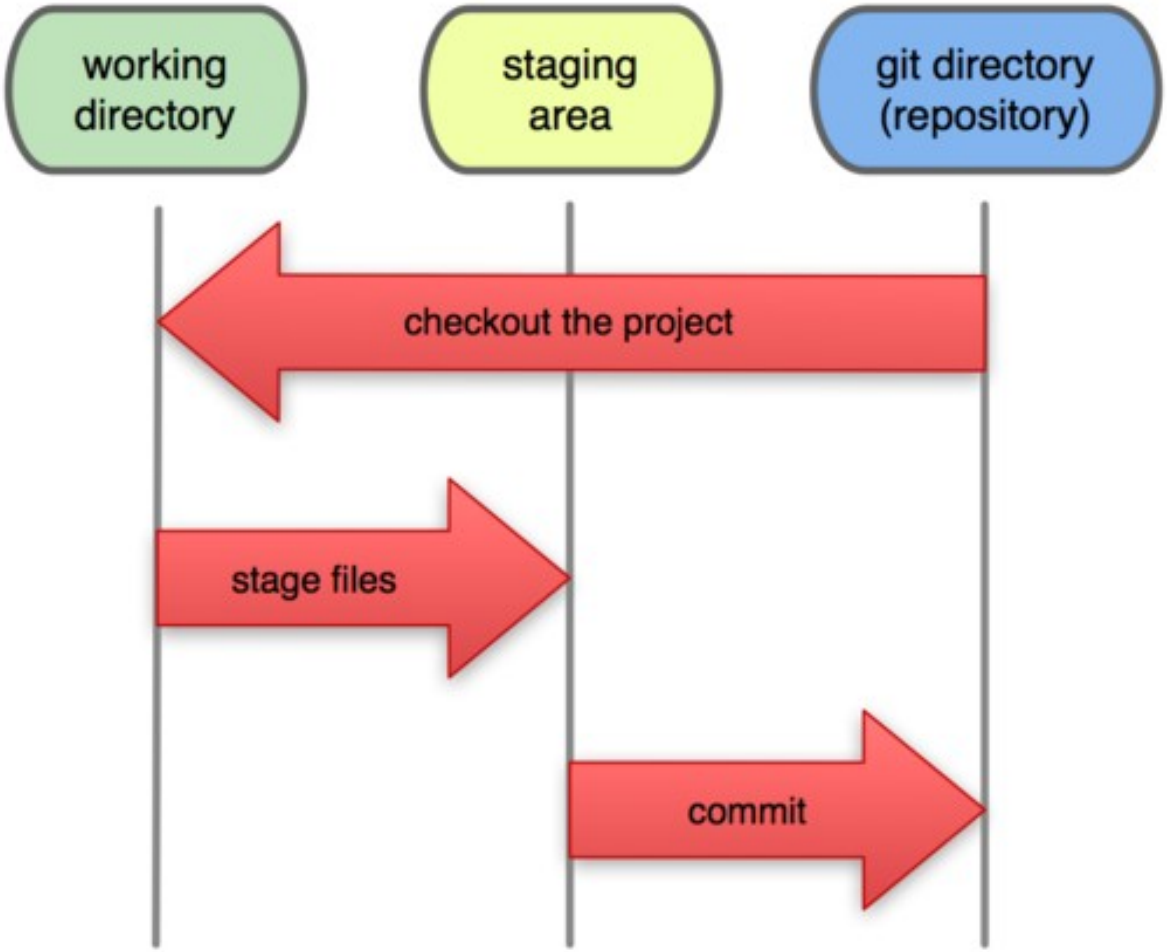
Three state

- Git has three main states that your files can reside in:
  - committed, modified, and staged.
- **Committed** means that the data is safely stored in your local database
- **Modified** means that you have changed the file but have not committed it to your database yet
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot

Three main area

- **Git directory**
  - is where Git stores the metadata and object database for your project.
  - the most important part of Git, and it is what is copied when you clone a repository from another computer
- **working directory**
  - is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify
- **staging area (index)**
  - is a simple file, generally contained in your Git directory, that stores information about what will go into your next commit

Local operations



Basic Git Workflow

1) You modify files in your working directory

2) You stage the files

   adding snapshots of them to your staging area

3) You do a commit

   which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

- If a particular version of a file:
  - is in the git directory, it's considered **committed**
  - If it's modified but has been added to the staging area, it is **staged**
  - if it was changed since it was checked out but has not been staged, it is **modified**

Installing Git

- On a **Debian-based** distribution like ***Ubuntu***
  - $ apt-get install git

- On **Mac**, there are two easy ways to install Git
  - use the graphical Git installer
    - http://code.google.com/p/git-osx-installer
  - via MacPorts (http://www.macports.org)
    - $ sudo port install git-core +svn +doc +bash_completion +gitweb

- On **Windows**, the msysGit project has one of the easier installation procedures (download the installer exe file)
  - http://code.google.com/p/msysgit

$ git config

- get and set configuration variables
  - /etc/gitconfig
    - contains values for every user on the system and all their repositories
    - if you pass the option --system to git config, it reads and writes from this file specifically
  - ~/.gitconfig
    - specific to your user. Use –-global option to read and write to this file
  - .git/config
    - config file in the git directory, specific to that single repository

First time Git setup

- Your Identity
  - $ git config --global user.name "Cristian Lucchesi"
  - $ git config --global user.email cristian.lucchesi@iit.cnr.it
- Your Editor
  - $ git config --global core.editor emacs
- Your Diff Tool
  - $ git config --global merge.tool vimdiff
- Checking Your Settings
  - $ git config –list
    user.name=Cristian Lucchesi
    user.email=cristian.lucchesi@iit.cnr.it
    .....

Getting help

- In 3 different ways
  - $ git help <verb>
  - $ git <verb> --help
  - $ man git-<verb>

- For example
  - $ git help config

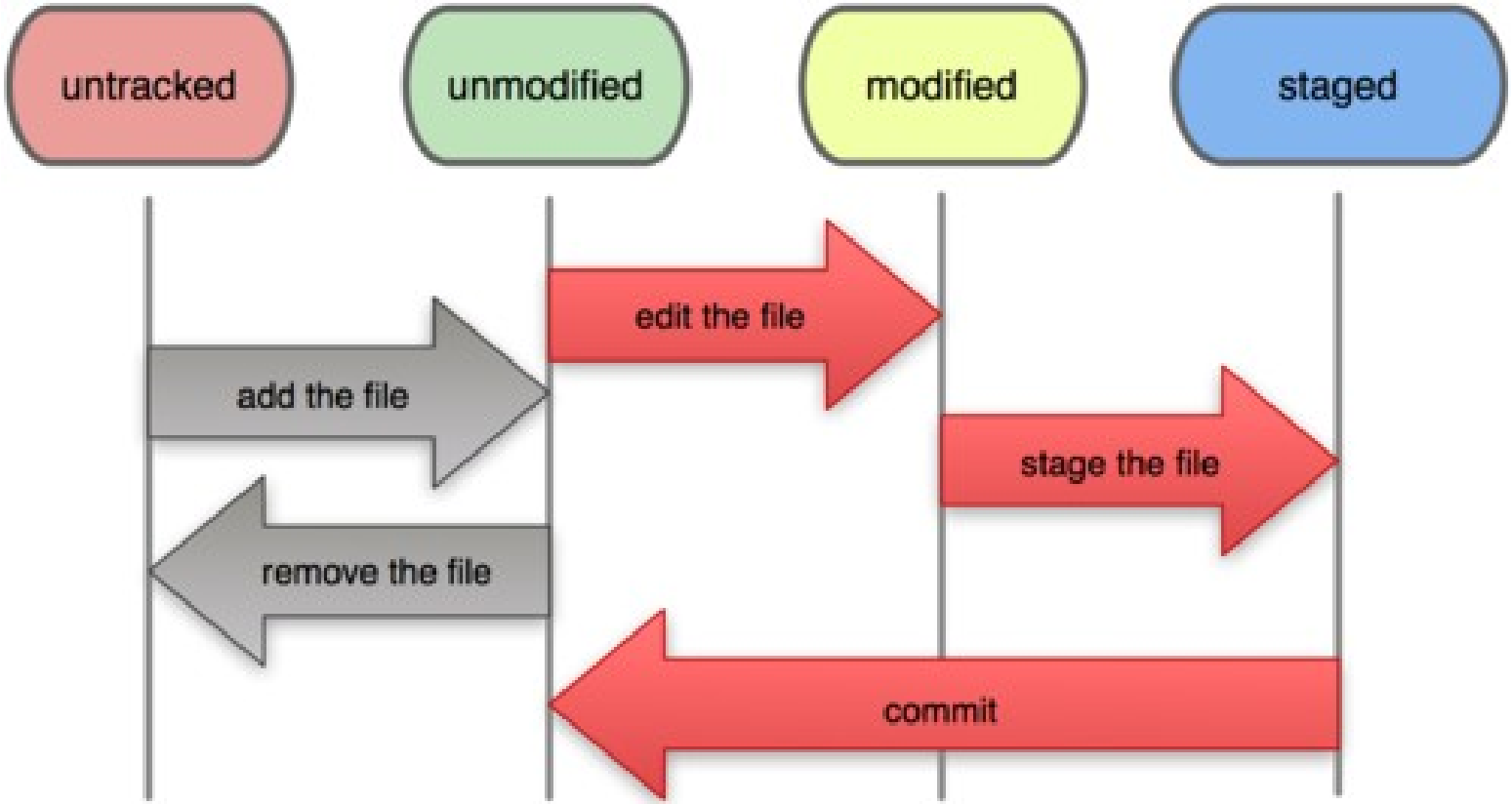
Getting a Git repository

- take an existing project or directory and import it
  - $ git init
    - creates .git dir containing a repository skeleton
    - nothing in your project is tracked yet
    - $ git add *.py to start tracking files
- clone an existing Git repository from another server.
  - $ git clone git://github.com/schacon/grit.git
    - creates a directory named grit
    - initializes a .git directory inside it
    - pulls down all the data for that repository
    - checks out a working copy of the latest version

Recording changes

- You need to make some changes and commit snapshots of those changes into your repository each time the project reaches a state you want to record.
- each file in your working directory can be in one of two states: **tracked** or **untracked**
- tracked:
  - files that were in the last snapshot;
  - they can be **unmodified**, **modified**, or **staged**
- untracked:
  - any files in your working directory that
    - were not in your last snapshot
    - are not in your staging area

File status lifecicle



Hands on Git

Undoing Things

- Changing Your Last Commit
  - $ git commit –amend
    - as an example, if you commit and then realize you forgot to stage the changes in a file you wanted to add to this commit:
      - $ git commit -m 'initial commit'
      - $ git add forgotten_file
      - $ git commit --amend

Undoing Things (cont)

- Unstaging a Staged File
  - git reset HEAD <file>...
    - or example, let’s say you’ve changed two files and want to commit them as two separate changes, but you accidentally type git add * and stage them both
      - $ git reset HEAD benchmarks.py
        - » the benchmarks.py file is modified but once again unstaged.

Undoing Things (cont)

- Unmodifying a Modified File
  - git checkout -- <file>...
    - can you easily unmodify a file revert it back to what it looked like when:
      - you last committed
      - or initially cloned
      - or however you got it into your working directory
    - git status tells you how to do that, too

Working with Remotes

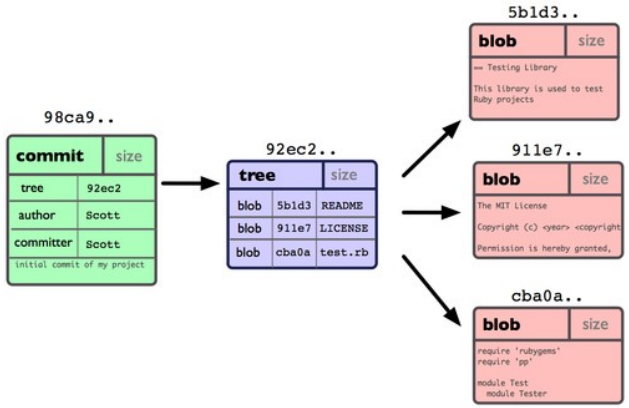
- Remote repositories are versions of your project that are hosted on the Internet or network somewhere
- you can have several of them
- $ git remote

  origin
  - **origin** is the default name Git gives to the server you cloned from
- $ git remote -v

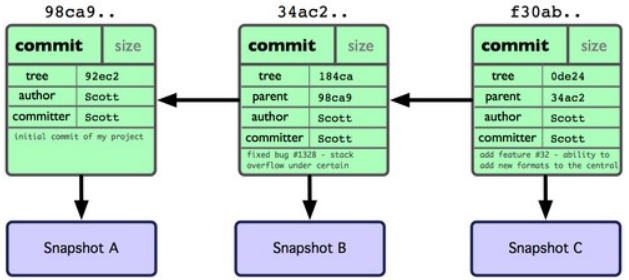
  origin git://github.com/schacon/ticgit.git

Git repository objects

- single commit repository

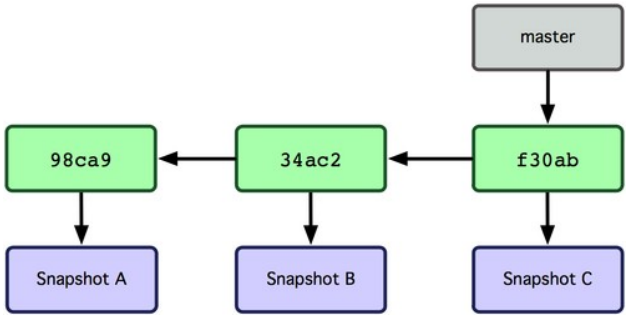


- Git object data for multiple commits

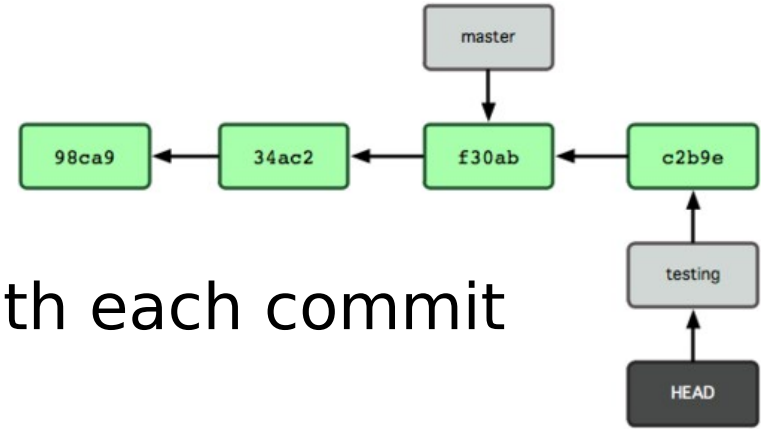


Git branching

- Branch pointing into the commit data's history



- the branch that HEAD points to moves forward with each commit



GIT basic HTTP Auth

- at least on Linux the HTTP Basic Auth doesn't work well
- Workaround:
  - create a .netrc file in your home dir

```
$ cat ~/.netrc

machine wiki.iit.cnr.it
login your-username
password your-password
```

This tutorial is on github

http://bit.ly/UJNcI1