Git

AN INTRODUCTION

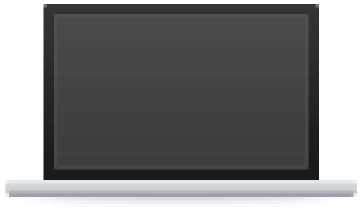
Introduction to Git as a version control system: concepts, main features and practical aspects.

POLITECNICO
DI TORINO

e-Lite

How do you share and save data?

- "I'm working solo… and I only have one computer"

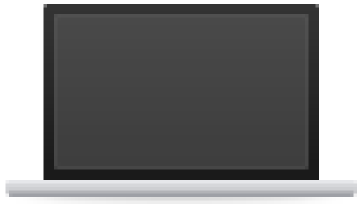


What I need:
- backup;
- different saved versions;
- early and frequently saving.

What I can use:
- external hard drives;
- dedicated folder;
- Dropbox folder;
- …

How do you share and save data?

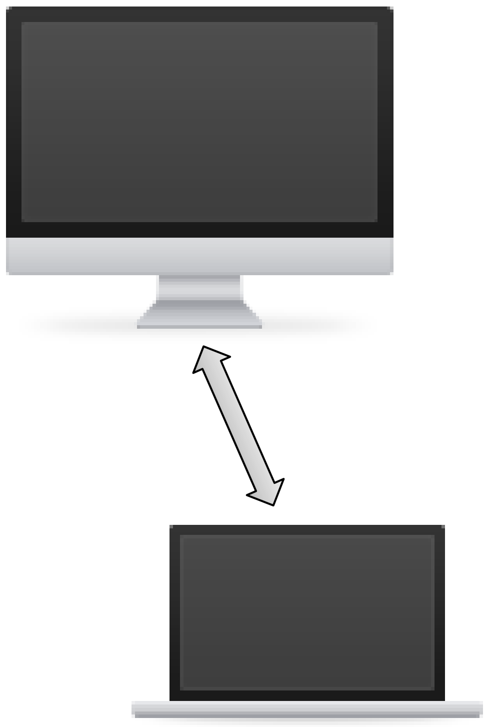
- "I'm working solo… and I only have one computer"



What if…
- … I forget to save a specific version and then I need it?
- … I delete/loose a previous version?

How do you share and save data?

- "I'm working solo… and I have more than one computer"

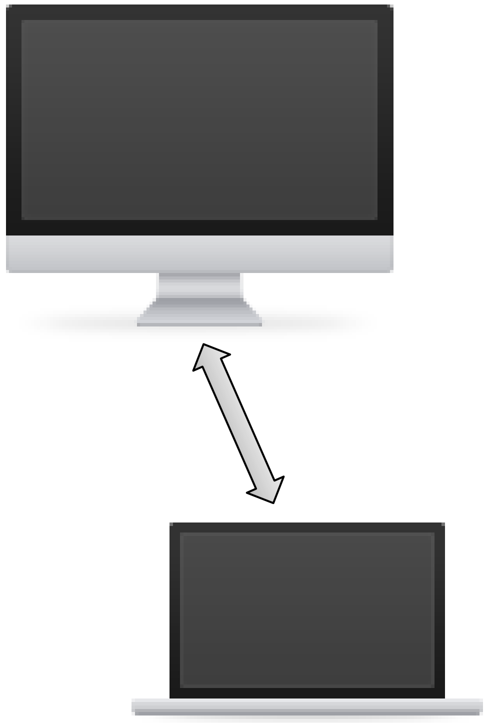


What I need:
- backup;
- different saved versions;
- early and frequently saving;
- conventions on file names.

What I can use:
- USB memory sticks;
- external hard drives;
- Dropbox folder;
- shared folder;
- …

How do you share and save data?

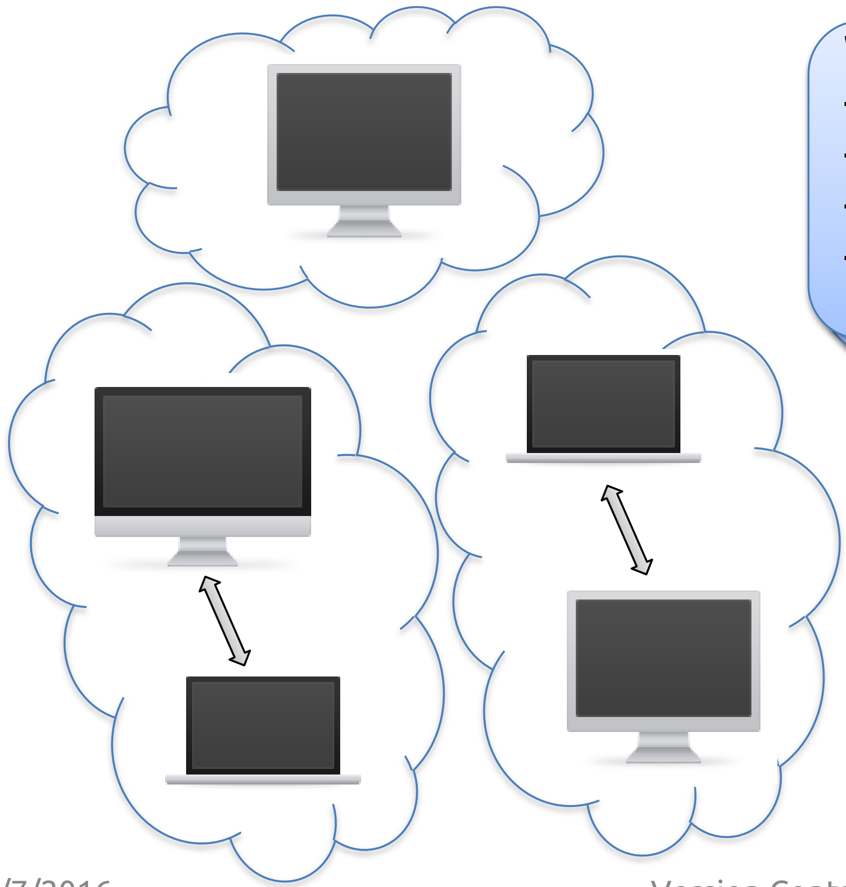
- "I'm working solo… and I have more than one computer"



What if…
- … I forget to save a specific version and then I need it?
- … I delete/loose a previous version?
- … I have different projects in the "shared" workspace?
- … I forget to copy one version between computers?

How do you share and save data?

- "I'm working in team"

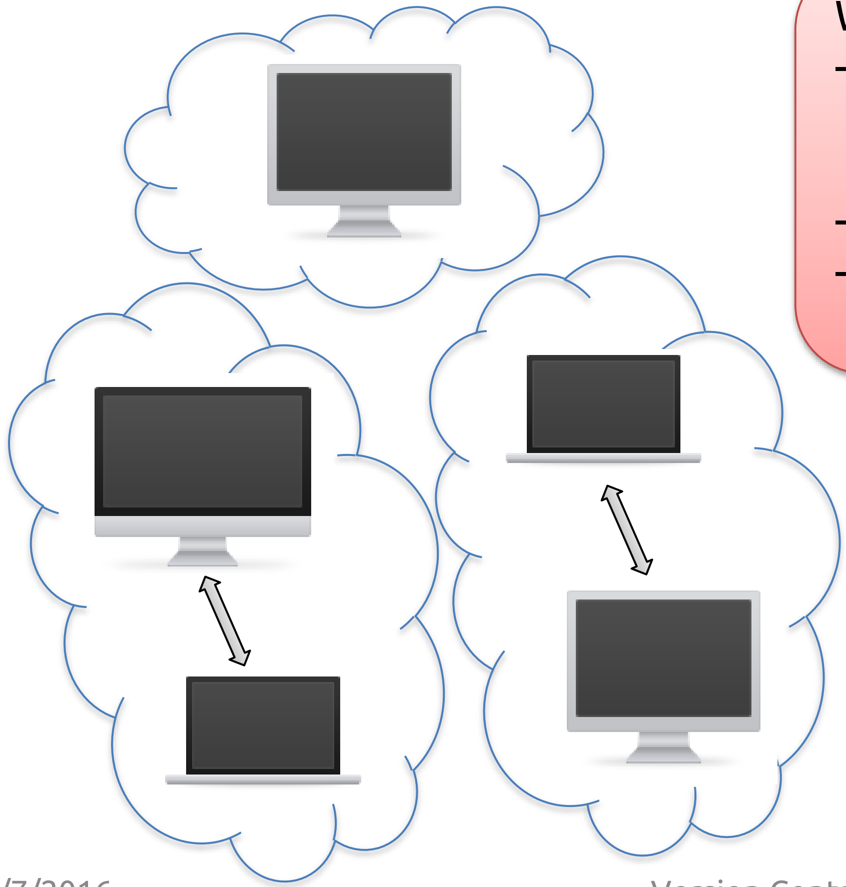


What I need:
- backup;
- different saved versions;
- early and frequently saving;
- shared conventions on file names.

What I can use:
- USB memory sticks;
- external hard drives;
- Dropbox folder;
- e-mails;
- …

How do you share and save data?

- "I'm working in team"



What if…
- … a team member forgets to save a specific version and then someone else needs it?
- … someone deletes/looses a version?
- … someone forgets to share a specific version of the projects?

Other issues:
- who has the latest version?
- who has the right to edit?
- how to ensure that everyone sees up-to-date versions of everything?
- how to handle conflicts?

Version Control Systems

Record changes to a file or a set of files over time so that you can recall specific versions later

Three generations:

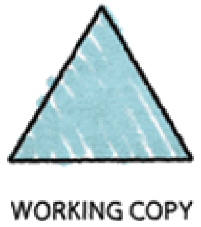
1. Local (RCS, SCCS)
2. Centralized (CVS, Subversion, Team Foundation Server)
3. Distributed (Git, Mercurial) ← NOW

Basic Concepts

Repository

- place where you store all your work
- contains every version of your work that has ever existed
  - files
  - directories layout
  - history
- can be shared by the whole team

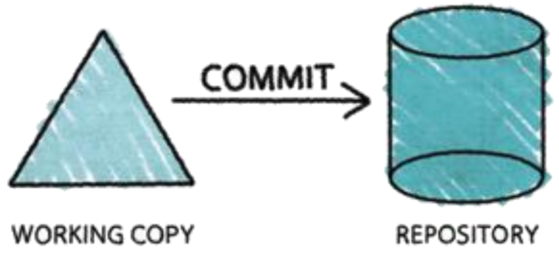
Basic Concepts



Working copy

- a snapshot of the repository used for… working
- the place where changes happens
- private, not shared by the team
- it also contains some metadata so that it can keep track of the state of things
  - has a file been modified?
  - is this file new?
  - has a file been deleted?

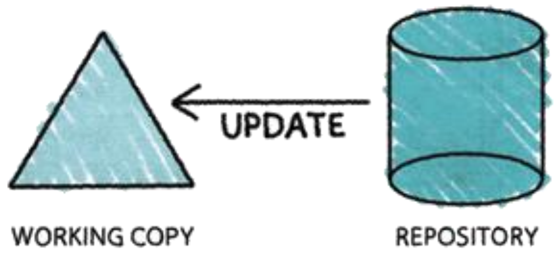
Basic Concepts



Commit

- the operation that modifies the repository
- atomically performed by modern version control tools
  - the integrity of the repository is assured
- it is typical to provide a log message (or comment) when you commit
  - to explain the changes you have made
  - the message becomes part of the history of the repository

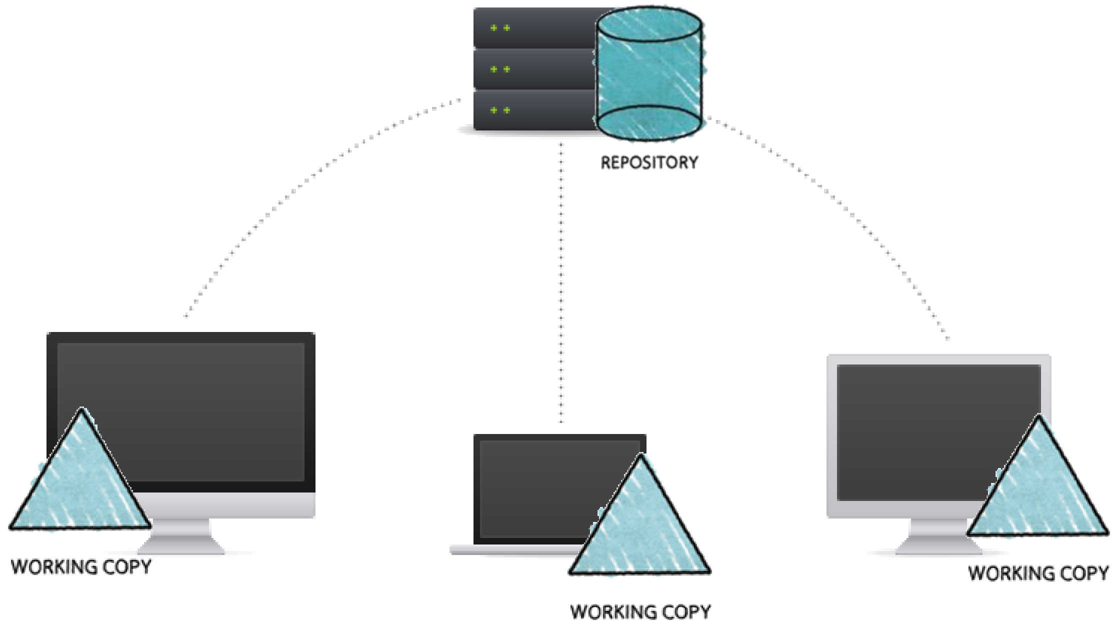
Basic Concepts



Update

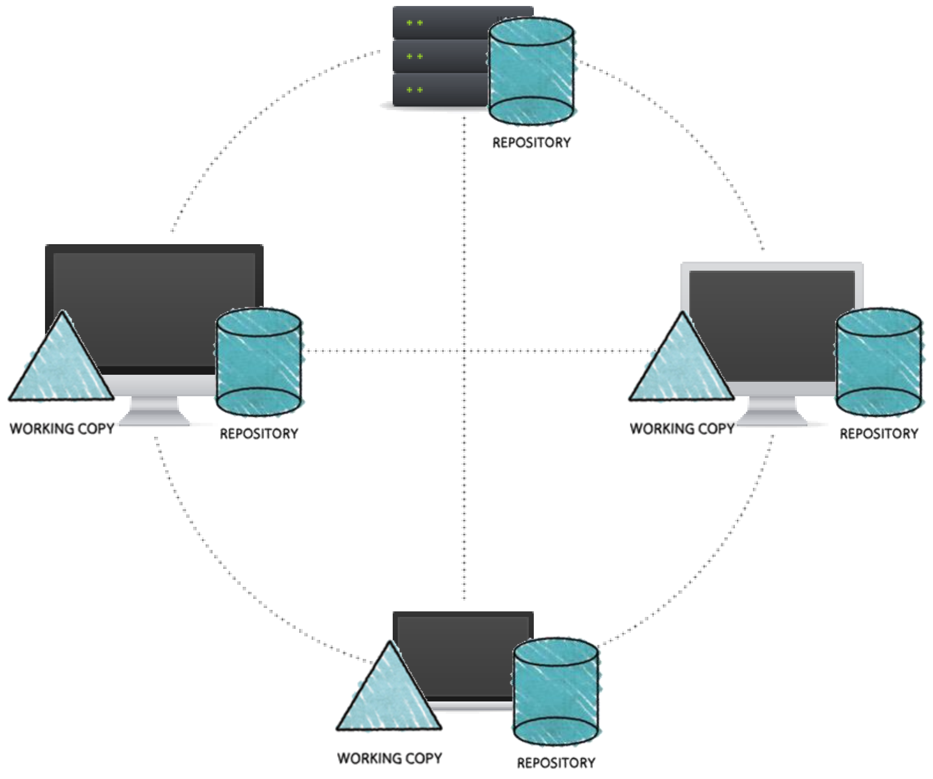
- update the working copy with respect to the repository
  - apply changes from the repository
  - merge such changes with the ones you have made to your working copy, if necessary

Centralized Version Control



- one central repository
- client-server relationship

Distributed Version Control

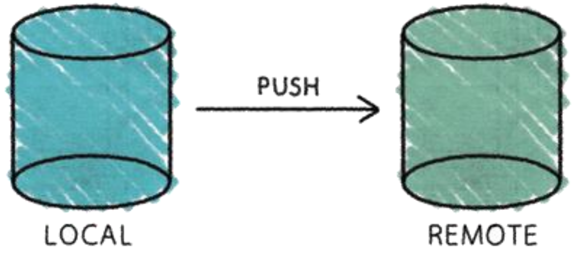


- clients and server have the full copy of the repository
  - local repositories clone a remote repository
- it is possible to have more than one server

More Basic Concepts

Push

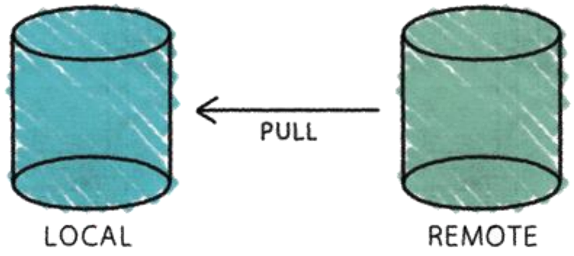
- copy changesets from a local repository instance to a remote one
  - synchronization between two repository instances



More Basic Concepts

Pull

- copy changesets from a remote repository instance to a local one
  - synchronization between two repository instances



Introducing… Git

- Distributed Version Control System
- Born
  - on 2005 for the Linux kernel project
  - to be used via command line
- Website: http://git-scm.com
- Highlights:
  - free and open source
  - strong support for non-linear development
  - fully distributed
  - efficient handling of large projects
  - cryptographic authentication of history

Who uses Git?

Google facebook Microsoft

LinkedIn NETFLIX GNOME

RAILS Qt

eclipse

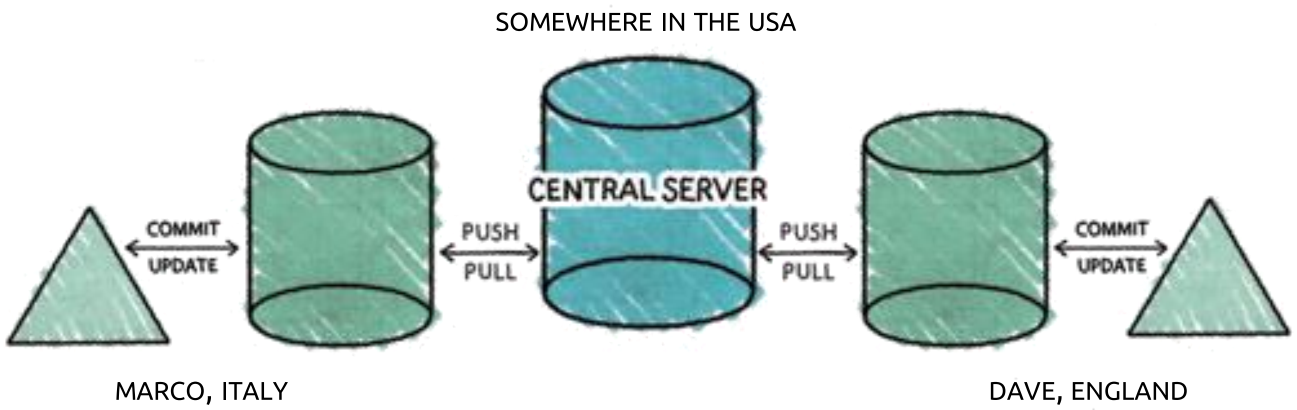
Getting started with Git

- Standard installations
  - http://git-scm.com/downloads
- Available for all the platform
- Git Graphical Applications
  - http://git-scm.com/downloads/guis
- For this course, Git is
  - integrated in PyCharm
  - already installed on the LADISPE computers

Installing Git

- Windows
  - download and install Git from http://git-scm.com/downloads
- Linux
  - check if it is already installed
    - open a terminal and type “git”
  - otherwise, install it from your package manager or via http://git-scm.com/downloads
- Mac
  - check if it is already installed
    - open a terminal and type “git”
  - otherwise, install it from http://git-scm.com/downloads

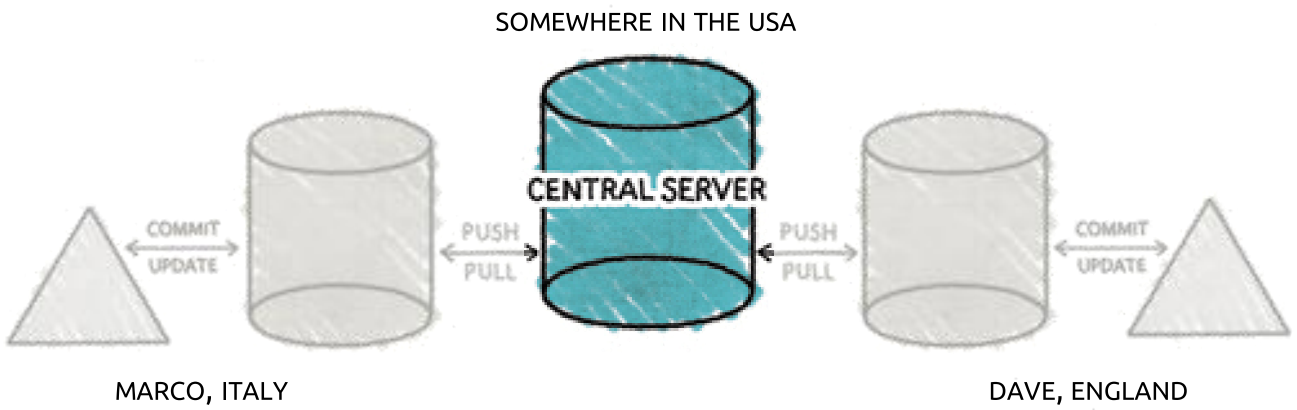
Git by Example



Marco and Dave work for the same company, but in two different countries.

They have to realize a new software project, so they decided to make it in Python and to use Git for version control.

Git by Example



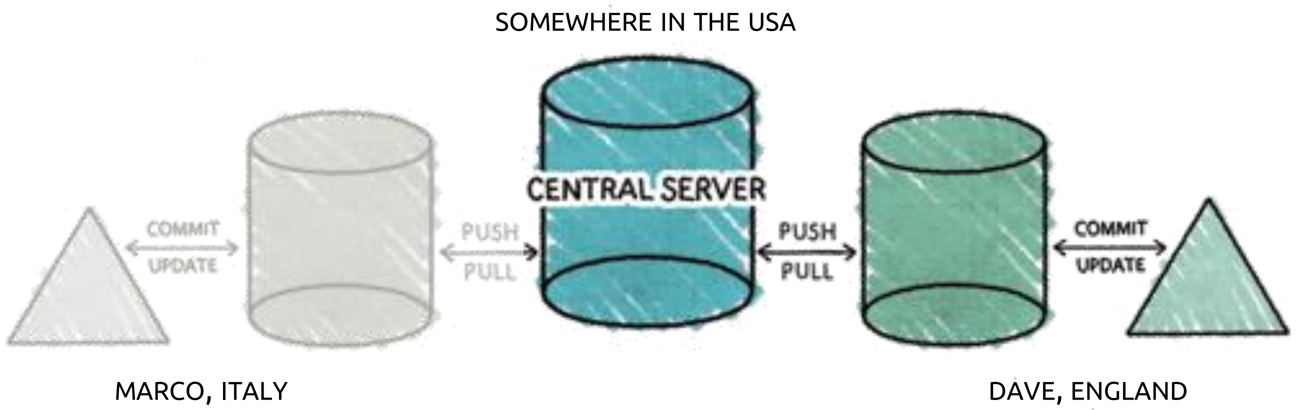
Marco starts the project by creating a new Git repository on the central server.

He goes into the project directory and types:

```
git init --bare myproject.git
```

to initialize an empty Git repository for the project.

Git by Example



When the central repository is ready, Dave create a folder named *myproject* on his computer and initializes a repository in it:

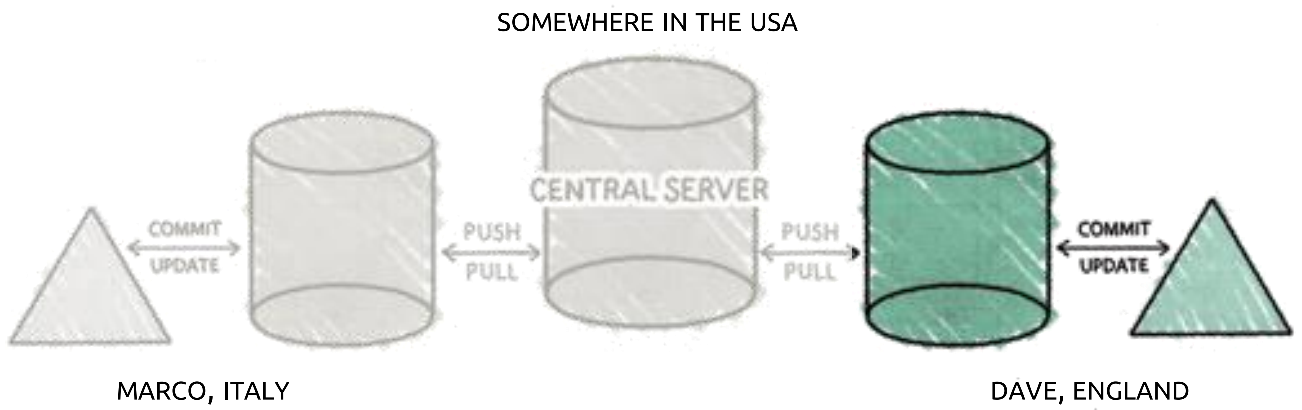
```
git init
```

Git by Example

git init

- initializes an empty Git repository inside an existing folder
- creates a .git directory inside it
- without parameters, typically
  - on the server, it is used with the *--bare* parameter

Git by Example

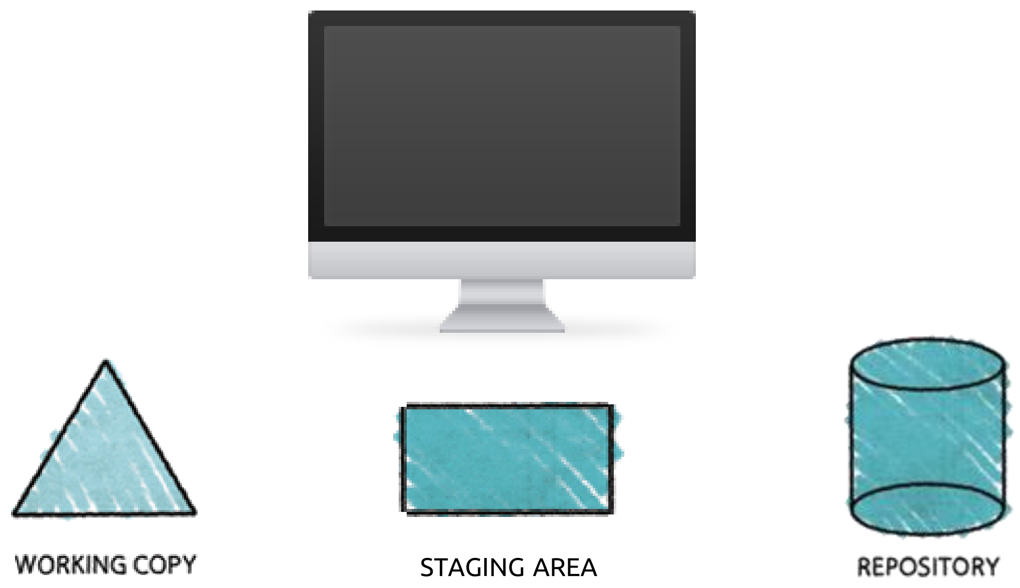


Dave writes some code in the *myproject* folder.

Before committing, Dave needs to really put the created file under version control, by adding the file to the Staging area:

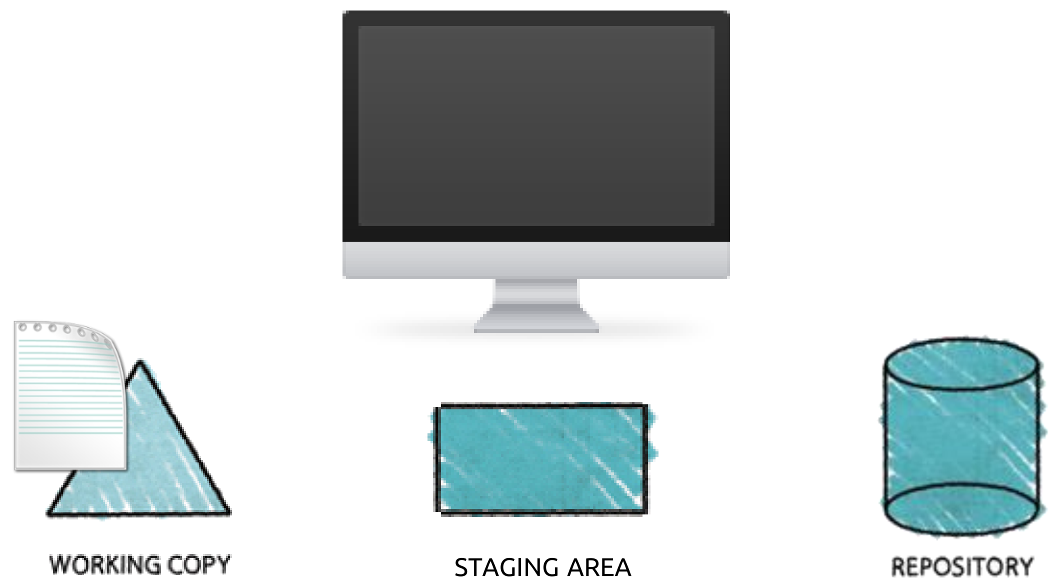
```
git add main.py
```

The Staging Area



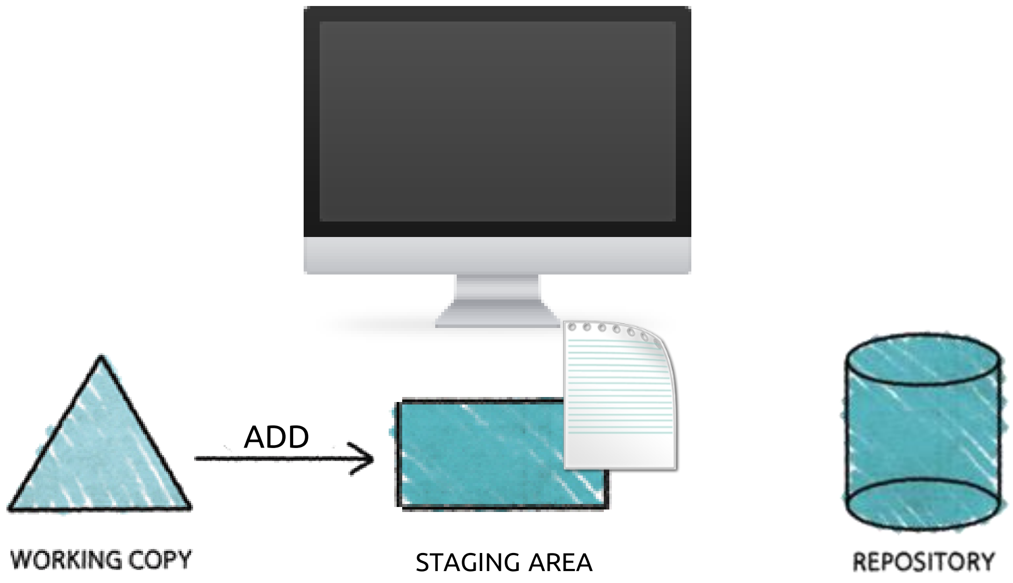
- A sort of loading dock
- It can contain things that are neither in the working copy nor in the repository instance
- Also called “index”

The Staging Area: an example



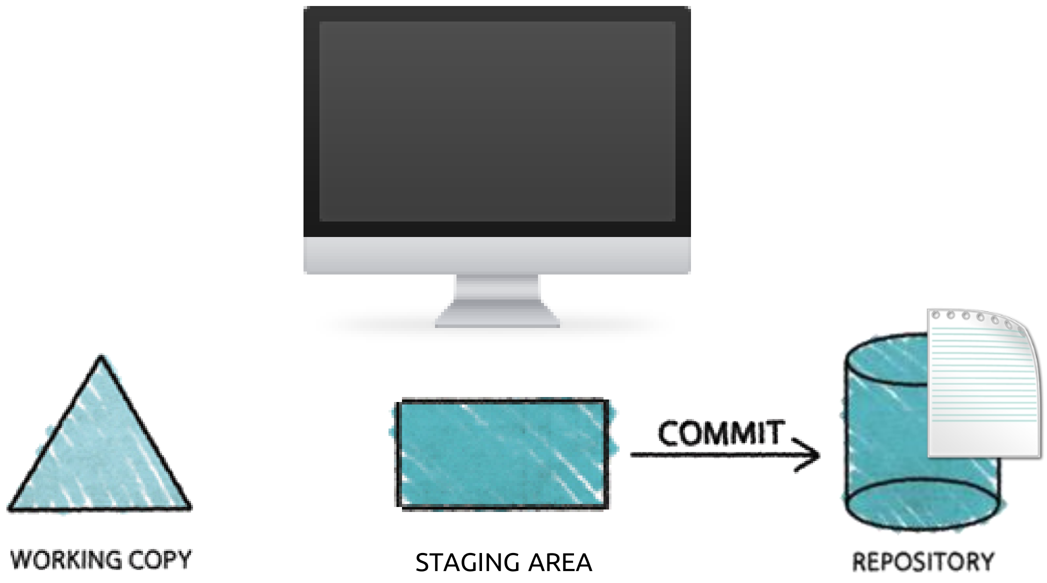
- Imagine to modify an existing file in the working copy
- The file is marked as “modified”

The Staging Area: an example



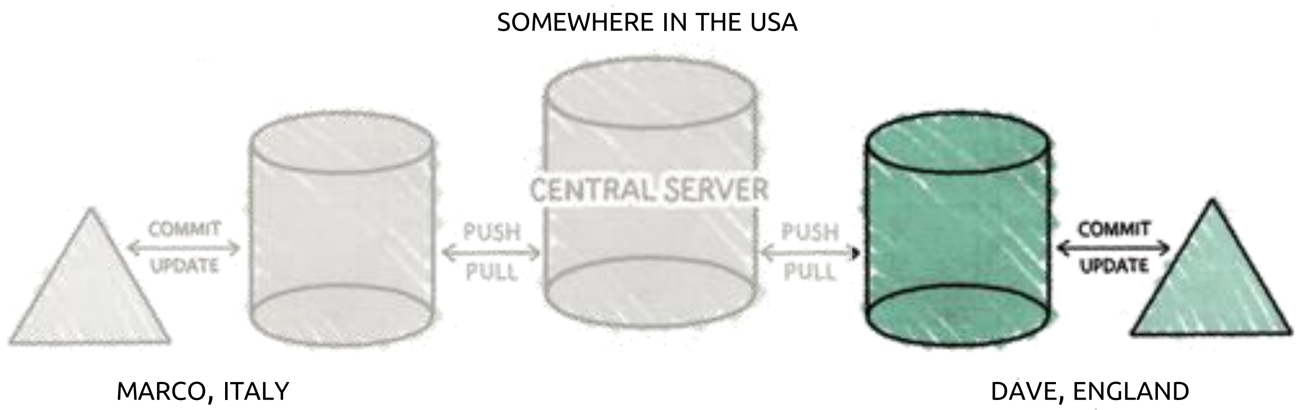
- Before committing, the modified file needs to be "staged"
  - i.e., add a snapshot of it in the staging area
- Modified data has been marked in its current version to go into the next commit snapshot

The Staging Area: an example



- Then, changes can be “committed”
  - i.e., take the file from the staging area and store permanently the snapshot in the local repository

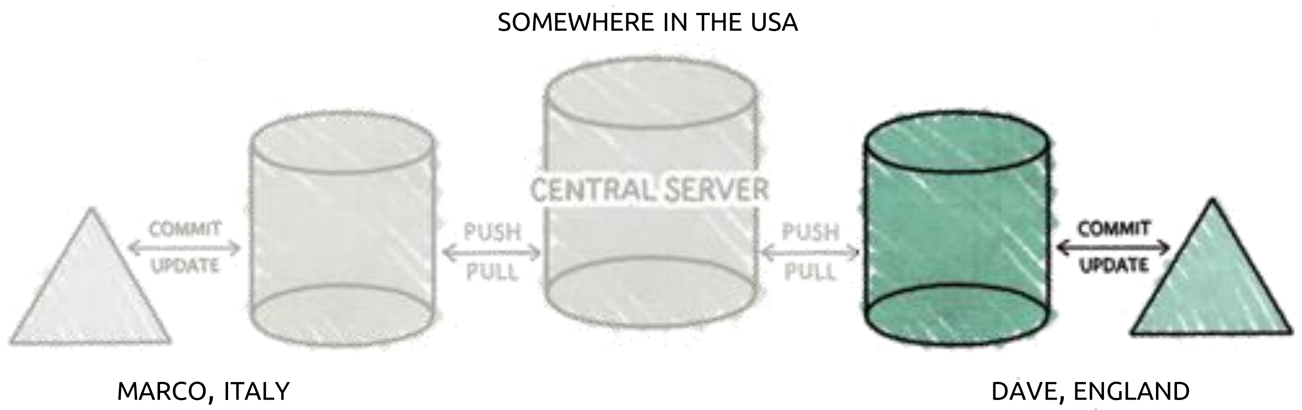
Git by Example



If Dave wants permanently to exclude from version control some files in the project folder, he can add them in the *.gitignore* file

– such files and folders will not be staged

Git by Example



After adding *main.py* to the Staging area, Dave can commit the file to the local repository:

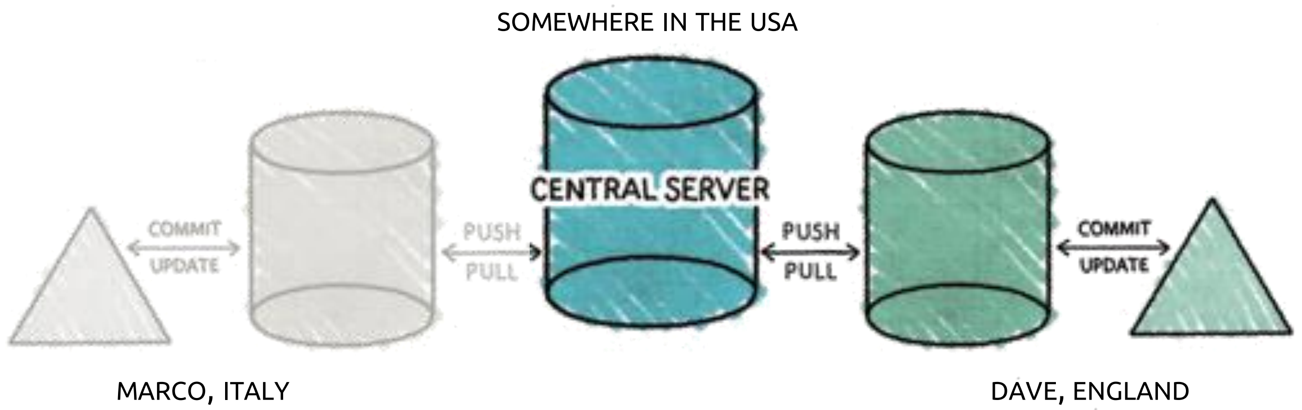
```
git commit -m “initial commit”
```

Git by Example

```
git commit -m "initial commit"
```

- store the current snapshot in the local repository
- -m "put a message here"
  - perform the commit with a log message
  - useful to know what you have committed
- - a
  - a useful parameter
  - it performs an add for modified files, too
    - useless at this point

Git by Example



Dave is ready to load the data to the remote repository. He has to specify what is the remote server to use:

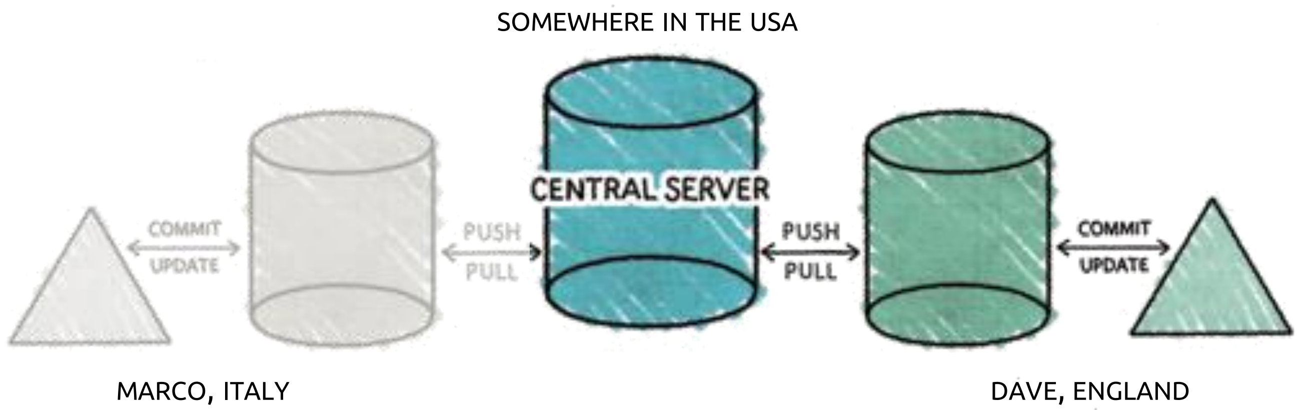
```
git remote add origin http://centralserver.com/myproject.git
```

Git by Example

```
git remote add origin http://centralserver.com/myproject.git
```

- add a new remote repository
  - multiple remotes can exists
- for each remote, a name and an address is specified
  - *origin* is the “standard” name for indicating the principal remote
  - the address can be in the format http(s):// or git://
- remotes can also be removed, renamed, etc.

Git by Example



Now Dave can push the data to the remote repository:

```
git push -u origin master
```

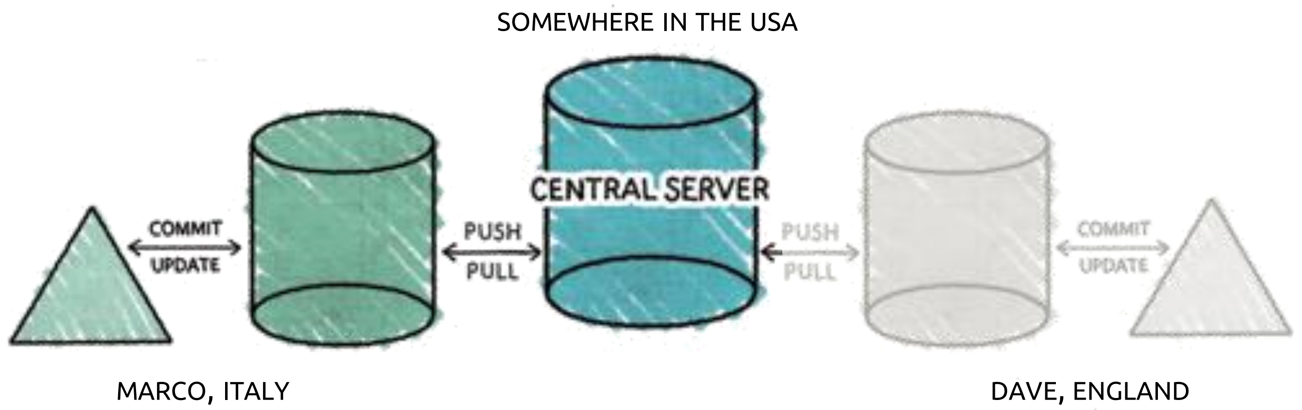
where *origin* is the remote name and *master* is the default branch name

Git by Example

```
git push -u origin master
```

- Git pushes only to matching branches
  - for every branch that exists on the local side, the remote side is updated if a branch of the same name already exists there
    - this behavior will change with Git 2.0
  - you have to push the branch explicitly the first time
- -u
  - set other information useful for other commands (e.g., pull)
- After the first time, you can simply use:
  - git push

Git by Example



Marco starts working on the project and clones the remote repository on his computer:

```
git clone http://centralserver.com/myproject.git
```

Now Marco has the code!

Git by Example

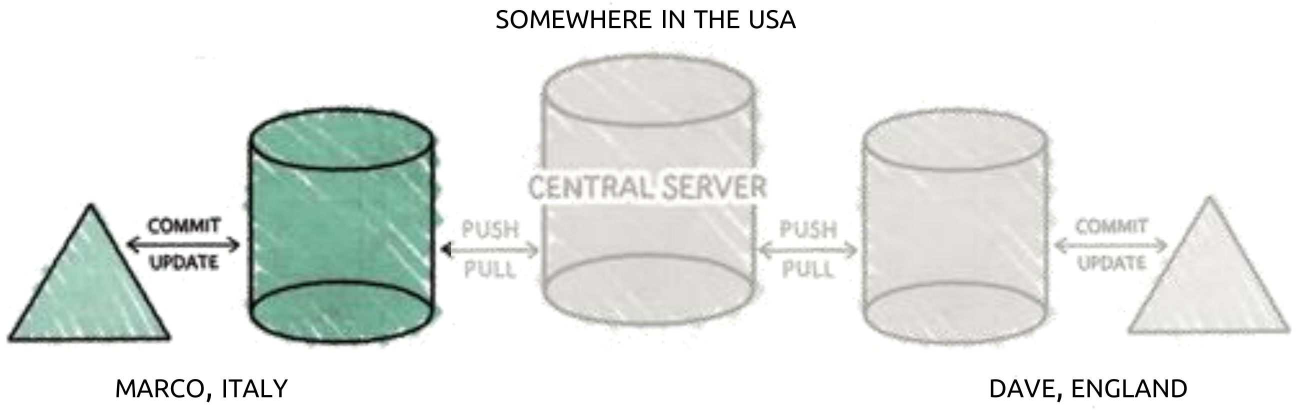
```
git clone http://centralserver.com/myproject.git
```

- creates a directory named *myproject*
- initializes a .git directory inside it
- pulls down all the data for that repository
- checks out a working copy of the latest version

If you want to clone the repository into a directory with another name, you can specify that as:

```
git clone http://centralserver.com/myproject.git first_project
```

Git by Example



Marco wants to see the details of what Dave did:

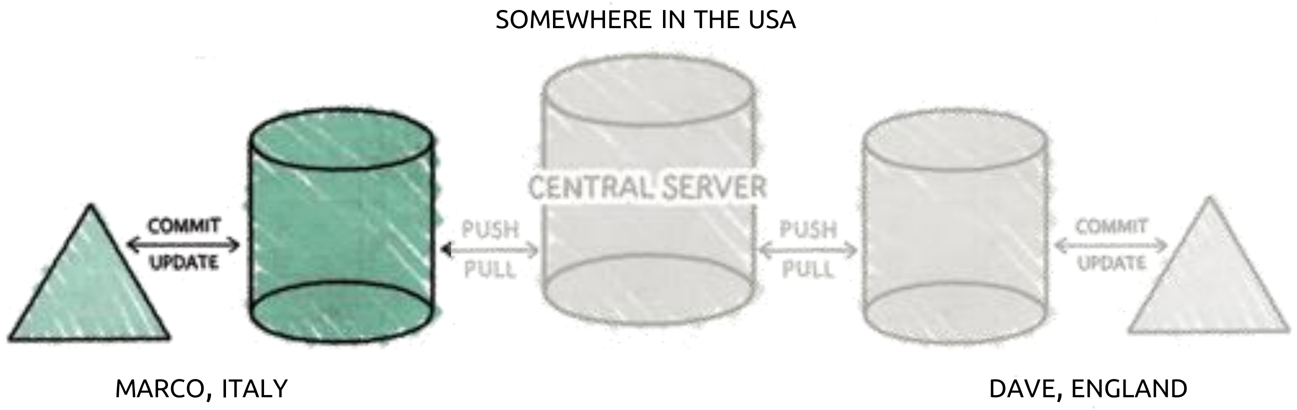
```
git log
```

The result will be something like:

```
commit bcb39bee268a92a6d2930cc8a27ec3402ebecf0d
Author: Dave <dave@email.co.uk>
Date:   Wed Mar 23 10:06:13 2015

initial commit
```

Git by Example



Marco wants to see the details of what Dave did:

```
git log
```

The result will be something like:

```
commi  bcb39bee268a92a6d2930cc8a27ec3402ebecf0d
Author
Date:

initial commit
```

SHA-1 hash for data integrity

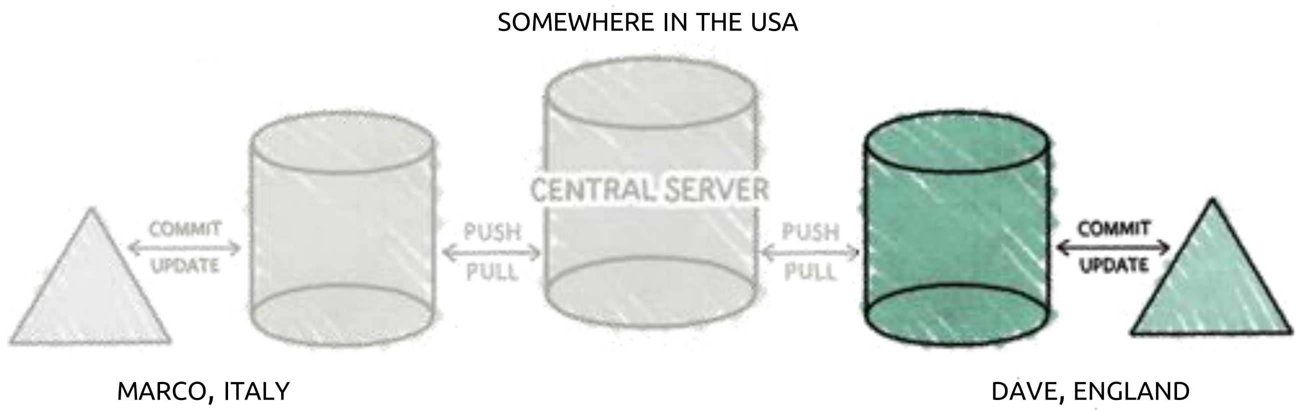
Git by Example

- At this point, Marco edits the source code and saves
- To see the pending changes, he can use:
  - git status
- To see the difference between his version and the previous one, he can use:
  - git diff (--cached, to include staged files)
- Marco decides to commit and to push his work

```
git commit -a -m "added new functionalities"
```

```
git push
```

Git by Example



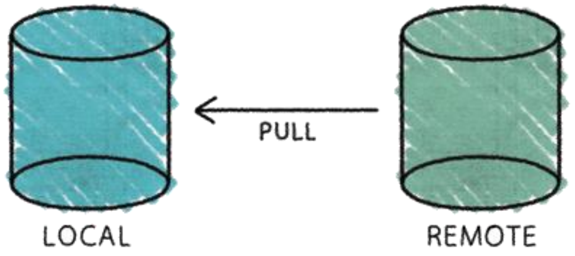
Meanwhile, Dave found some bugs in the code. He looks for update on the central server and get it (if any):

```
git pull
```

Pull and Fetch in Git

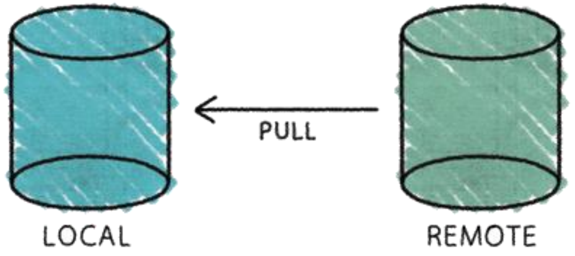
Fetch

- copy changesets from a remote repository instance to a local one
- previously, we called it "pull"

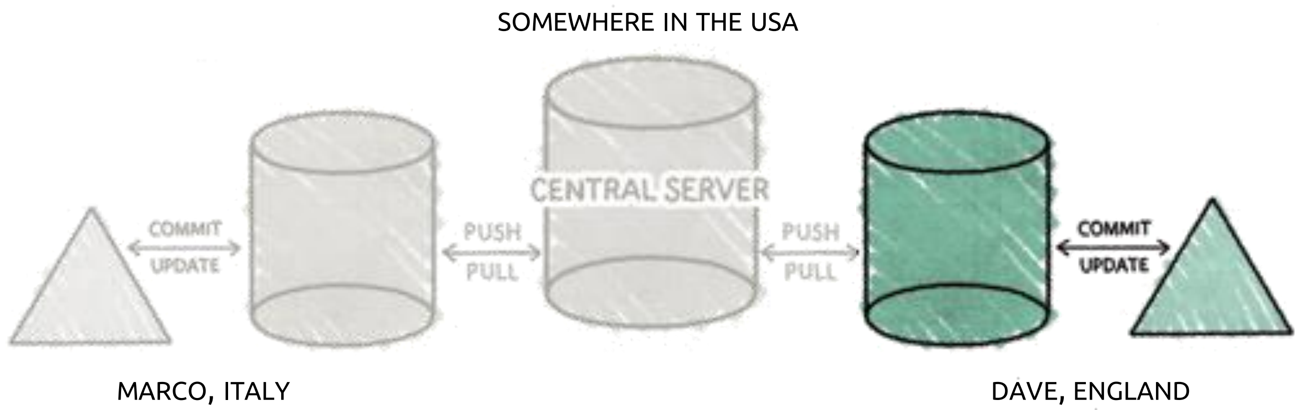


Pull

- perform fetch
- update the working copy



Git by Example

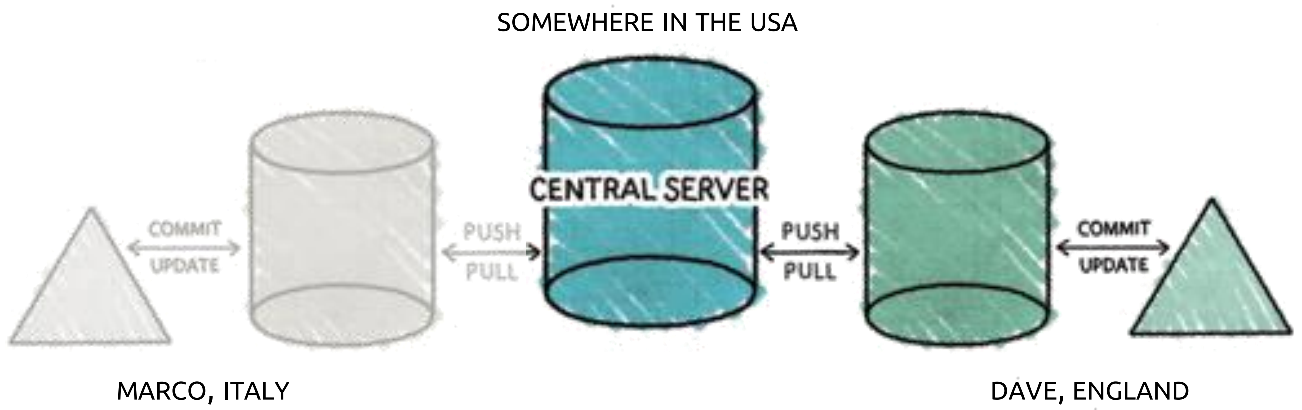


However, no new data is available since Marco has not yet pushed his changes.

So, Dave fixes the bugs, and commits:

```
git commit -a -m “bug fixing”
```

Git by Example



After some time, Dave tries to push his changes

```
git push
```

but something goes wrong

```
To http://centralserver.com/myproject
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to ‘http://centralserver.com/myproject’
```

Git by Example

- What happens?
  - Git is not allowing Dave to push his changes because Marco has already pushed something to the master branch
- Solution:
  - Dave has to do a pull, to bring in changes before pushing his modifications
- Two possible scenarios:
  - merging of files goes smoothly;
  - merging of files generates conflicts.

Git by Example

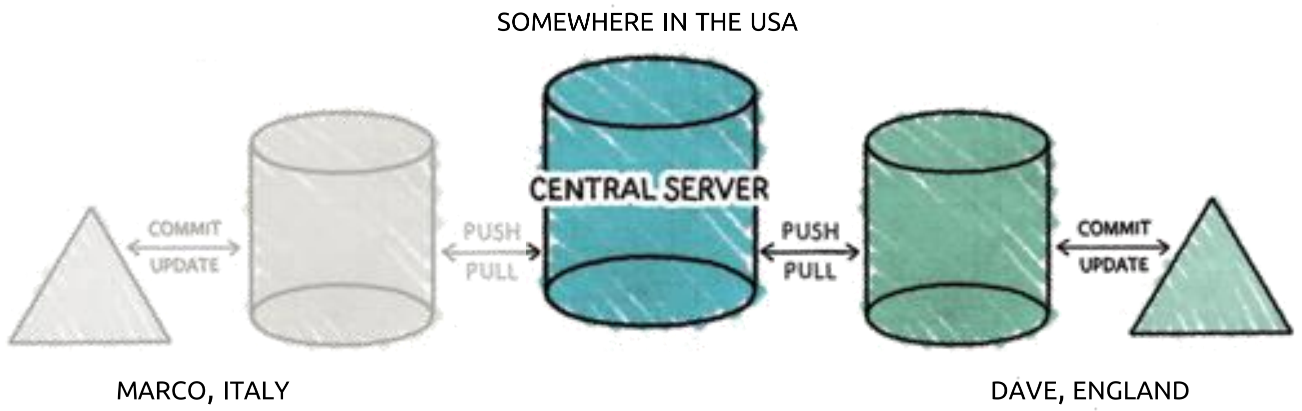
- Merge with conflicts

```
From http://centralserver.com/myproject
  b19f36c..b77378f  master -> origin/master
Auto-merging main.py
CONFLICT (content): Merge conflict in main.py
Automatic merge failed; fix conflicts and then commit the result.
```

- Git includes both Marco’s code and Dave’s code with conflict markers to delimit things

```
<<<<<<< HEAD
   # Marco’s code here
=======
   # Dave’s code here
>>>>>>> b77378f6eb0af44468be36a085c3fe06a80e0322
```

Git by Example



After (manually) resolving these conflicts, Dave is able to push the changes:

```
git push
```

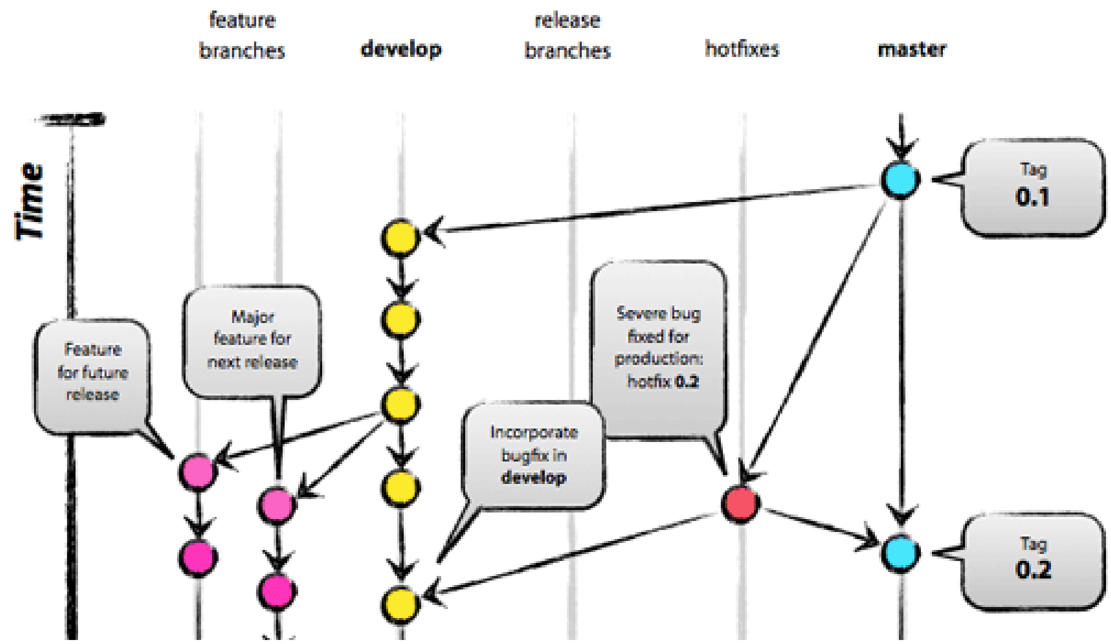
Other Useful Commands

- Operations on files
  - Remove: *git rm [filename]*
  - Move/rename: *git mv [file-from] [file-to]*
  - Unstage some staged files: *git reset HEAD [filename-list]*
  - Unmodify a modified file: *git checkout -- [filename]*
- Change the last commit
  - *git commit --amend*

Other Useful Commands

- Operations on remotes
  - List: *git remote* (-v, to show the URLs)
  - Add: *git remote add [shortname] [url]*
  - Inspect: *git remote show [remote-name]*
  - Rename: *git remote rename [old-name] [new-name]*
  - Remove: *git remote rm [remote-name]*

Tags and Branches in a Nutshell


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2

- Local and remote
- Do not push automatically

[Image from http://nvie.com/posts/a-successful-git-branching-model/]

Tags… in brief

- useful to mark **release points**
- two types:
  - lightweight
  - annotated (more complete)
- commands:
  - *git tag*, shows the available existing tags
  - *git tag [tag-name]*, creates a lightweight tag
  - *git tag -a [tag-name] -m [message]*, creates an annotated tag
  - *tag show [tag-name]*, shows the tag data

Branches… in brief

- used to develop features isolated from each other
- the *master* branch is the "default" branch when you create a repository
  - you should use other branches for development and merge them back to the master branch upon completion
- really lightweight in Git
- commands:
  - *git branch [branch-name]*, create a new branch
  - *git branch*, lists all existing branches
  - *git checkout [branch-name]*, switches to the selected branch
  - *git branch -d [branch-name]*, removes the selected branch

Hosted Git

- To have (at least) one remote repository
  - alternative: set up your own Git server!
- Most popular:
  - GitHub, https://github.com/
  - Bitbucket, https://bitbucket.org/
  - GitLab, https://about.gitlab.com/gitlab-com/
  - Sourceforge, http://sourceforge.net/
  - CodePlex (by Microsoft), https://www.codeplex.com/

GitHub

- Slightly different than other code-hosting sites
  - instead of being primarily based on the project, it is user-centric
  - social coding
- A commercial company
  - charges for accounts that maintain private repository
  - free account to host as many open source project as you want
  - free Micro plan for students
    - 5 private repositories, unlimited public repositories
    - https://education.github.com

Bitbucket

- Similar to GitHub
- Less used than GitHub, right now
- Mercurial support
- A commercial company
  - free private and public repositories for small team (up to 5 private collaborators)
  - charges for project involving bigger team
  - free for academia (also for students)
    - unlimited public and private repositories
    - unlimited users for single projects

GitHub Pages

- Website for your (GitHub) repository
  - https://pages.github.com/
- We will use it for hosting your project website
  - deliverables, video, etc.
  - your website will be reachable from *http://ami-2016.github.io/your-project-name*
- FAQ
  - https://help.github.com/categories/github-pages-basics/

Homework(s)

- Create a personal GitHub account
  - you can also ask for a “student discount” at https://education.github.com
  - we will require your username to set up your team repositories for final projects
    - **report your GitHub username on the shared Google doc!**
- Try Git!
  - http://try.github.io/
  - 15 minutes tutorial

References

- Git Reference
  - http://gitref.org/
- Git - the simple guide
  - http://rogerdudler.github.io/git-guide/
- Git Documentation
  - http://git-scm.com/docs
- Pro Git (online book)
  - http://git-scm.com/book
- Version Control by Example (online book)
  - http://www.ericsink.com/vcbe/

References

- Try Git!
  - http://try.github.io/
- Various Git resources
  - https://help.github.com/articles/what-are-other-good-resources-for-learning-git-and-github
- A successful Git branching model
  - http://nvie.com/posts/a-successful-git-branching-model/
- Some Git (graphical) clients
  - http://git-scm.com/downloads/guis

Questions?

01QZP Ambient Intelligence

Luigi De Russis
luigi.derussis@polito.it

License