

# 08

## DVCS, parte II

### Input-Output e Reflection

Danilo Pianini  
Giovanni Ciatto, Angelo Croatti, Mirko Viroli

Ingegneria e Scienze Informatiche  
ALMA MATER STUDIORUM—Università di Bologna, Cesena

29 ottobre 2017



- 1 Decentralized Version Control Systems
  - Risolvere i merge conflict
  - Lavorare in remoto: clone, fetch, pull, push
  - Hosting e Bitbucket
  - Plugin Eclipse (eGit)
  - Features avanzate e prossimo episodio
  
- 2 Testing con JUnit



## 1 Decentralized Version Control Systems

- Risolvere i merge conflict
- Lavorare in remoto: clone, fetch, pull, push
- Hosting e Bitbucket
- Plugin Eclipse (eGit)
- Features avanzate e prossimo episodio

## 2 Testing con JUnit



## 1 Decentralized Version Control Systems

- Risolvere i merge conflict
- Lavorare in remoto: clone, fetch, pull, push
- Hosting e Bitbucket
- Plugin Eclipse (eGit)
- Features avanzate e prossimo episodio

## 2 Testing con JUnit



# Conflicted files

## In generale

Abbiamo visto nell'ultima lezione come sia possibile riunire due flussi di lavoro diversi. Abbiamo anche visto che, nel caso in cui due linee di sviluppo abbiano modificato concorrentemente un file nello stesso punto, il merge non è banale da effettuare ma occorre risolvere manualmente un *merge conflict*.



# Esempio di conflict file

```
public final class HelloWorld {  
  
    private static final String AUTHOR = "Danilo Pianini";  
  
    public static void main(final String[] args) {  
<<<<<< HEAD  
        System.out.println("This program is running in a PC with " + procNumber() + " logic processors!");  
    }  
  
    public static int procNumber() {  
        return Runtime.getRuntime().availableProcessors();  
    }  
    =====  
    System.out.println("This program has been realised by " + AUTHOR);  
>>>>>> feature  
    }  
}
```



# Risoluzione del conflitto

- Modifica dei file che confliggono, fino a portarli allo stato desiderato.
- Aggiunta dei file alla staging area (con `git add`)
- Salvataggio delle modifiche tramite `git commit`
  - ▶ Il messaggio di commit di default nel caso di merge viene auto-generato da Git e può essere mantenuto



## Errore frequente: bad tracking

Cosa succede, se per errore abbiamo messo in tracking rigenerabili che non avremmo dovuto mettere?

### Un altro buon motivo per stare attenti a cosa si mette in tracking

- Se abbiamo dei file binari in tracking (ad esempio dei class files che vengono ricompilati ogni volta), ad **ogni** merge avremo dei conflitti.
- Ci sarà un conflitto per ogni risorsa modificata. Potenzialmente, centinaia ad ogni merge.
- I conflitti su binari sono difficili da risolvere: il file può essere solo ispezionato tramite editor binari, e buona fortuna a capirne il contenuto.
- Di norma si risolve cancellando tutti i file, rigenerandoli, marcando tutti i conflitti come risolti e facendo un commit.
- **ESPLODE** la dimensione del repository
- State attenti a cosa mettete in tracking!



## 1 Decentralized Version Control Systems

- Risolvere i merge conflict
- Lavorare in remoto: clone, fetch, pull, push
- Hosting e Bitbucket
- Plugin Eclipse (eGit)
- Features avanzate e prossimo episodio

## 2 Testing con JUnit



# Decentralizzazione

## Decentralizzazione totale...

Nei DVCS, non esiste un punto centrale che fa repository “ufficiale”: tutti i repository ricevono l’intera storia ed hanno pari importanza.

Il design dei DVCS è ideale per un mondo P2P

## ... nella realtà della struttura di Internet

Per via della struttura attuale di Internet, della presenza di NAT e simili strutture di rete, il modello client/server è spesso difficile da aggirare. È spesso anche sconveniente aggirarlo: tutto sommato, il cloud spesso è utile... Esistono servizi di hosting che consentono di avere in un punto sempre accessibile dei repository in rete: chi ha bisogno di collaborare non deve preoccuparsi di problematiche di rete diverse da “andare sul web”.



# Clone

## In generale

Occorre un meccanismo che consenta di fare copie di repository esistenti, al fine di cominciare a lavorare su qualcosa di già esistente, o di effettuare copie di sicurezza del proprio lavoro.

## In Git

- `git clone URI localfolder`

Scarica l'intera storia del repository conservato in URI all'interno di `localfolder`, che diventa un repository Git. Esempi di URI:

- `/home/user/repository/` — URI locale (\*nix)
- `C:\Users\Username\repository\` — URI locale (Windows)
- `ssh://username@server/repository/` — URI Secure Shell (raccomandata per chi lavora in remoto con server e/o client Unix)
- `https://user@server/repository/` — URI HTTPS, multiplatforma

# Remote repositories I

## In generale

Vogliamo conservare un riferimento agli URI ai quale si trovano i repository dai quali vogliamo leggere o sui quali vogliamo scrivere.

In caso di operazioni remote, infatti, è comodo avere un nome simbolico invece di specificare sempre una URI.



# Remote repositories II

## In Git

Il sottocomando `git remote` consente di gestire repository remoti. Ogni repository remoto ha un nome ed una URL.

- `git remote -v`
  - ▶ Elenca i repository remoti configurati
- `git remote add name url`
  - ▶ Aggiunge un nuovo repository remoto di nome `name` che punta ad `uri`
- `git remote rm name`
  - ▶ Rimuove il repository remoto `name`



# Remote repositories III

## Branch upstream

È possibile configurare, per ciascun branch, un uri di un repository remoto dove si andrà a leggere e scrivere a meno di diversa specifica

- Si usa l'opzione `-u` (`--set-upstream-to`)
  - ▶ `git branch -u remoteName/remoteBranch`
  - ▶ D'ora in poi, tutte le operazioni accesso remoto lanciate dal branch corrente verranno mappate sul branch `remoteBranch` del repository remoto `remoteName`

Nel caso in cui il repository sia stato clonato e non inizializzato (ossia, il primo comando dato è stato `git clone` e non `git init`), un riferimento al repository di origine viene automaticamente inserito fra i remote e configurato come upstream per tutti i branch col nome di `origin`.



## Visualizzazione dei branch remoti

Al momento del clone, Git scaricherà solo il branch “primario” (normalmente, `master`). È possibile comunque visualizzare tutti i branch disponibili in remoto:

- `git branch -a`
  - ▶ Elenca tutti i branch, inclusi quelli remoti.



## Importazione di branch remoti

Qualora si voglia importare in locale un branch remoto per lavorarci, occorre creare un “tracking branch” locale:

- `git checkout -b localBranchName remoteName/remoteBranchName`
  - ▶ Crea un nuovo branch di nome `localBranchName`, che avrà i contenuti di `remoteName/remoteBranchName`
  - ▶ Sovente si dà al branch locale lo stesso nome del branch remoto
    - `git checkout -b develop origin/develop`





## In generale

Vogliamo una operazione che ci consenta di acquisire nuovi commit da una fonte, in maniera tale da acquisire il lavoro fatto da altri. Tale operazione si chiama fetch, e *teoricamente* è l'unica operazione necessaria a rendere completamente distribuito un DVCS.



# Fetch e pull II

## In Git

Il sottocomando `fetch` scarica i commit da un repository remoto

- `git fetch URI branchName`
  - ▶ Scarica da URI tutti i commit del branch `branchName`
- `git fetch remoteName branchName`
  - ▶ Scarica dal remote `remoteName` tutti i commit del branch `branchName`
- `git fetch`
  - ▶ Scarica dal remote di default (se presente) tutti i commit

Dopo la `fetch`, è necessario utilizzare `git merge` per unire i commit scaricati al branch corrente.



# Fetch e pull III

## Pull

Il sottocomando `pull` (stessa sintassi di `fetch`) esegue una `fetch` seguita da un `merge`, ed è quindi conveniente da usare nel caso in cui si vogliono eseguire entrambe le operazioni.

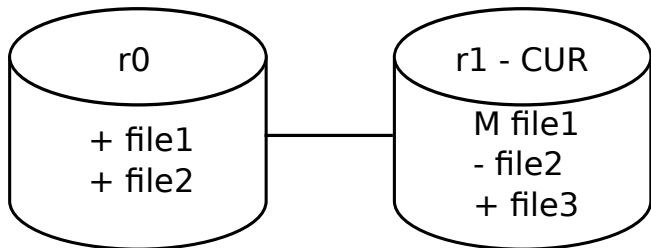
- `git pull URI branchName`
  - ▶ Scarica da `URI` tutti i commit del branch `branchName` e prova a mergerli nel branch corrente
- `git pull remoteName branchName`
  - ▶ Scarica dal remote `remoteName` tutti i commit del branch `branchName` e prova a mergerli nel branch corrente
- `git pull`
  - ▶ Scarica dal remote di default (se presente) tutti i commit e prova a mergerli nel branch corrente



# Esempio con clone e pull

Situazione iniziale

REMOTE  
REPO



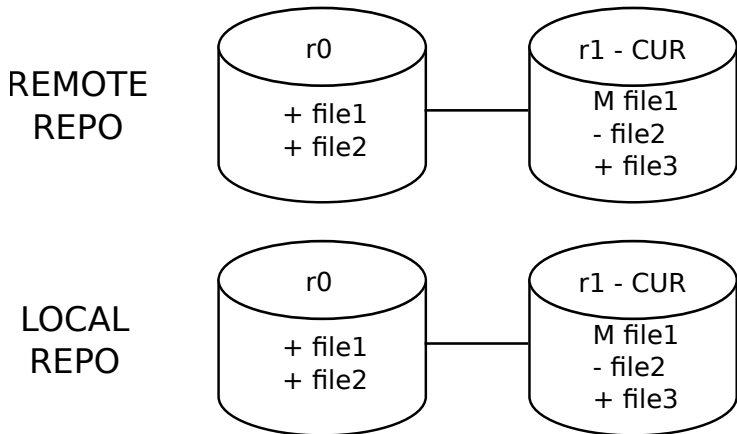
LOCAL  
REPO



## Esempio con clone e pull

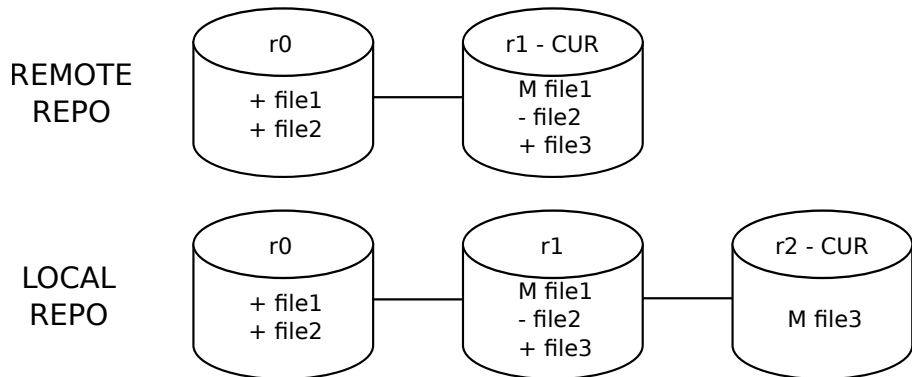
Local esegue:

```
git clone indirizzo_di_remote_repo percorso_di_local_repo
```



# Esempio con clone e pull

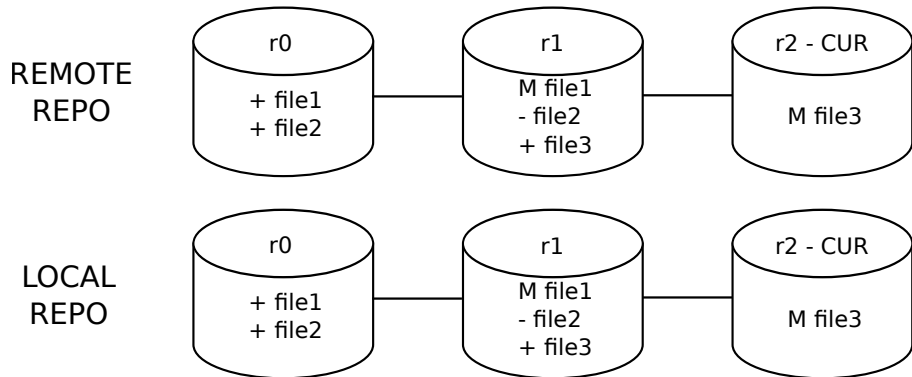
Local esegue:  
modifica di file3  
commit



# Esempio con clone e pull

Remote esegue:

```
git pull indirizzo_di_local_repo
```



## In generale

Nella realtà dei fatti le operazioni di `fetch` e `pull` non sono sufficienti. L'operazione duale della `pull` si chiama `push`, ed è più delicata: chi la esegue deve avere i diritti di scrittura verso la destinazione.

## In Git

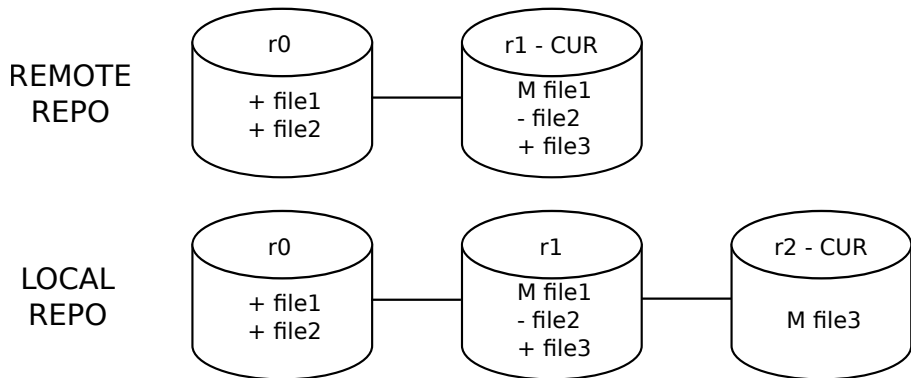
- `git push URI branchName`
  - ▶ Carica sul branch `branchName` `URI` tutti i commit del branch
- `git push remoteName branchName`
  - ▶ Carica sul remote `remoteName` tutti i commit del branch `branchName`
- `git push`
  - ▶ Carica dal remote di default (se presente) tutti i commit del branch corrente

Nel caso in cui ci fossero commit sul branch destinazione del repository remoto non presenti in quello locale, la `push` **fallirebbe**. In questo caso, infatti, è necessario effettuare prima una `pull`



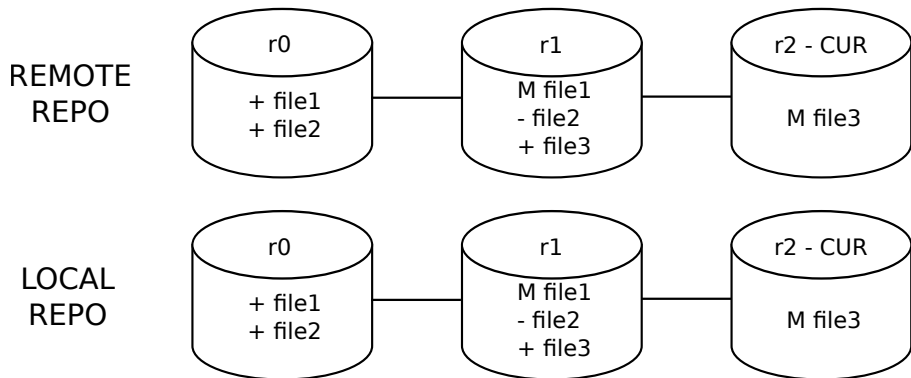
# Esempio con push I

Situazione iniziale:



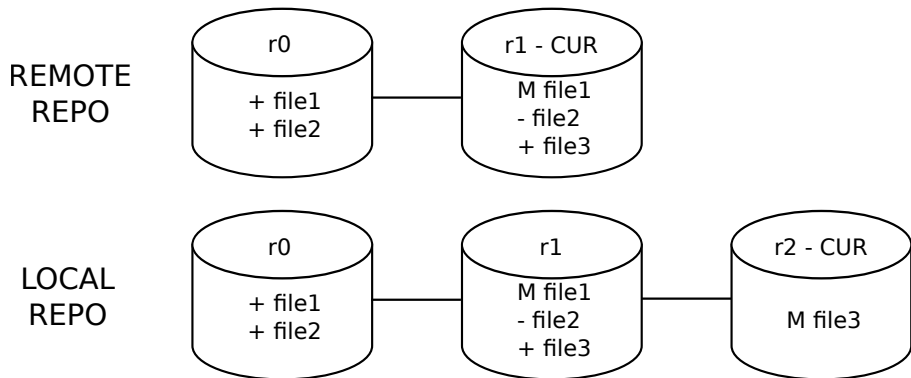
## Esempio con push II

Local esegue: `git push indirizzo_di_remote_repo`



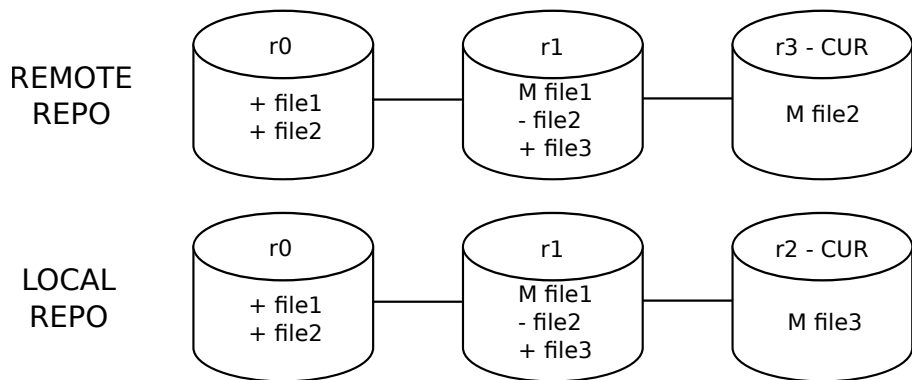
# Lavorare in parallelo: esempio I

Situazione iniziale:



## Lavorare in parallelo: esempio II

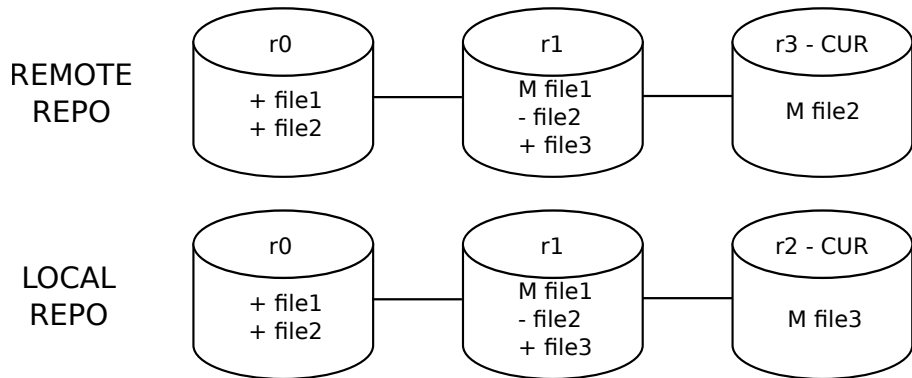
Remote esegue:  
Modifica di file2  
git commit



## Lavorare in parallelo: esempio III

Local esegue:

```
git push indirizzo_di_remote_repo
```



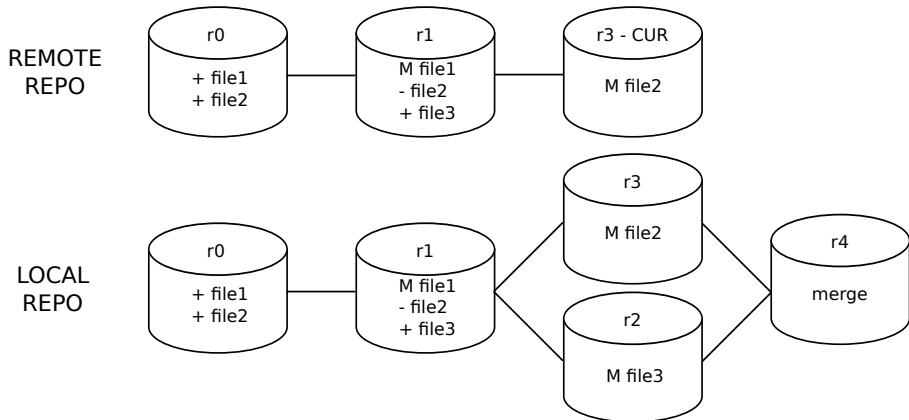
La push viene rifiutata: la radice dei due repository è diversa!



## Lavorare in parallelo: esempio IV

Local esegue:

```
git pull indirizzo_di_remote_repo
```



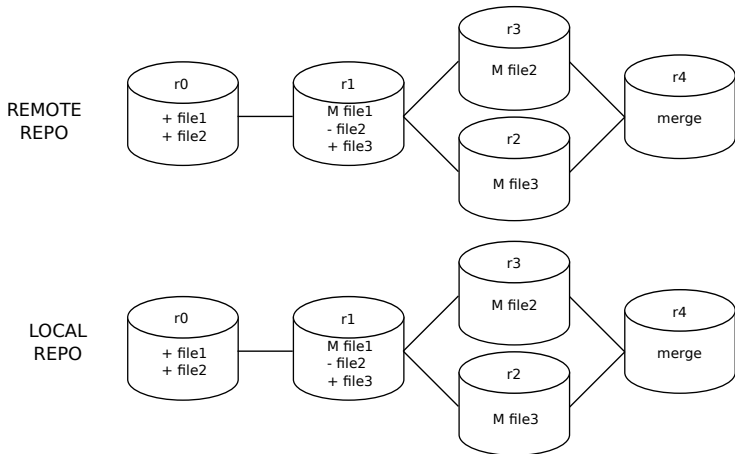
Ora è possibile effettuare la push!



# Lavorare in parallelo: esempio V

Local esegue:

```
git push indirizzo_di_remote_repo
```



## 1 Decentralized Version Control Systems

- Risolvere i merge conflict
- Lavorare in remoto: clone, fetch, pull, push
- **Hosting e Bitbucket**
- Plugin Eclipse (eGit)
- Features avanzate e prossimo episodio


## 2 Testing con JUnit





# Bitbucket Overview

Bitbucket Dashboard Teams Repositories Create owner/repository

 **Danilo Pianini** (danyusk)  
<http://danilopianini.org/>  
Cesena  
Member since November 2011

Manage account

Overview Followers 11 Following 13

Type: All Public Private Language

Find repositories

**Recent activity**

**4 commits**  
Pushed to danyusk/Courses - 2014 - OOP Laboratory








- d18311e Start work on Lab08. Create pers...
- 0c60cd0 Move images from Lab 07 to Lab...
- 8eb983a Kill ti with fire, burn it, BUUUUUU...
- 6a4222a BURN IN HELL F\*\*\*\*\*G AMPER...

Danilo Pianini · 14 minutes ago

**5 commits**  
Pushed to danyusk/Test

- a3ea9dd Merge commit
- 1329755 Change HelloWorld to print autho...
- 25f7ada Change HelloWorld in such a wa...
- 144baba Create ,hgignore
- 12c7966 Create HelloWorld

Danilo Pianini · 17 hours ago

	Courses - 2014 - OOP Laboratory	Updated 14 minutes ago
	Test	Updated 17 hours ago
	Courses - OOP - Merge Conflict Test	Updated 17 hours ago
	Paper - 2014 - SAC	Updated 2014-11-04
	OOP13 - Foschi Davide, Gondolini Monica, Righi Lorenzo - Greenki...	Updated 2014-10-23
	OOP13 - Aprile Anna Chiara, Sasso Greta - TubbyHamster	Updated 2014-10-21
	Alchemist	Updated 2014-10-20



# Bitbucket Overview

Bitbucket Dashboard Teams Repositories Create owner/repository

danyisk Courses - 2014 - OOP Labo...

## Overview

SSH ssh://hg@bitbucket.org/danyisk/course Share

Last updated	15 minutes ago	1 Branch	8 Tags
Fork of	mcasadei/oop2...		
Language	Other		
Access level	Admin	1 Fork	1 Watcher

### Invite users to this repo

Send invitation

This fork is 2 commits behind mcasadei/oop2014-lab. Sync now.

### Recent activity

4 commits Pushed to danyisk/Courses - 2014 - OOP Laboratory

- d18311e Start work on Lab08. Create pers...
- 8c60cd0 Move images from Lab 07 to Lab...
- 8eb983a Kill ti with fire, burn it, BUUUUUU...
- 6a4222a BURN IN HELL F\*\*\*\*\*G AMPER...

Danilo Pianini - 15 minutes ago

4 commits Pushed to danyisk/Courses - 2014 - OOP Laboratory

**THERE ISN'T A README YET**  
Create one and tell people where to start and how to contribute.  
[Create a README](#)

Clone  
Create branch  
Create pull request  
Compare  
Fork

Overview  
Source  
Commits  
Branches  
Pull requests  
Downloads  
Settings



# Bitbucket Overview

The screenshot displays the Bitbucket web interface for a repository named "Courses - 2014 - OOP Laboratory". The top navigation bar includes "Dashboard", "Teams", "Repositories", and a "Create" button. The user profile "owner/repository" is visible in the top right. The left sidebar contains "ACTIONS" (Clone, Create branch, Create pull request, Compare, Fork) and "NAVIGATION" (Overview, Source, Commits, Branches, Pull requests, Downloads, Settings). The main content area, titled "Source", shows a file tree for "Courses - 2014 - OOP Laboratory /" with folders numbered 01 through 12, and subfolders "Esercitazione C#", "LAB\_CODE/OOP1415-LAB03", and "OOP-Exam-Exercises". A "New file" button is located in the top right of the source view.



# Bitbucket Overview

Bitbucket Dashboard Teams Repositories Create owner/repository ? ?

danyk Courses - 2014 - OOP Labo...

ACTIONS

- Clone
- Create branch
- Create pull request
- Compare
- Fork

NAVIGATION

- Overview
- Source
- Commits
- Branches
- Pull requests
- Downloads
- Settings

## Commits

All branches Find commits

Author	Commit	Message	Date
Danilo Pianini	d18311e	Start work on Lab08. Create personalised .sty in root folder	17 minutes ago
Danilo Pianini	0c60cd0	Move images from Lab 07 to Lab 08	17 hours ago
Danilo Pianini	8eb983a	Kill ti with fire, burn it, BUUUUUUUURNhg status! (and also correct a typo	20 hours ago
Danilo Pianini	6a4222a	BURN IN HELL F*****G AMPERSAND	20 hours ago
Danilo Pianini	986cf66	Merge Matteo's	20 hours ago
Danilo Pianini	8bff18e	Minor fixes to Lab 07	21 hours ago
Danilo Pianini	59fabd9	Ignore Eclipse metadata and .directory files	21 hours ago
Danilo Pianini	3b861a6	Add PMD default file	21 hours ago
mcasadei	cde8ce9	Commit first revision of lab 07 sol: still to be checked againg with checkstyle	22 hours ago
mcasadei	ef8dcdd	code refactoring	yesterday
mcasadei	00ea576	Added tag 7.0 for changeset e44c4eb8461c	4 days ago
mcasadei	e44c4eb	Delete exceptions on code for es 04	4 days ago
mcasadei	71e0625	Added tag 7.0 for changeset 96f0ddd9aa0a	4 days ago
mcasadei	96f0ddd	finalized lab 07	4 days ago



# Bitbucket Overview

The screenshot displays the Bitbucket web interface. At the top, a dark blue navigation bar contains the Bitbucket logo, a menu icon, and navigation links for 'Dashboard', 'Teams', 'Repositories', and 'Create'. On the right side of this bar, there is a search field with the text 'owner/repository', a help icon, and a user profile icon.

On the left side, a sidebar menu is visible. It includes a profile icon for 'danyisk' and the repository name 'Courses - 2014 - OOP Labo...'. Below this, there are two sections: 'ACTIONS' with links for 'Clone', 'Create branch', 'Create pull request', 'Compare', and 'Fork'; and 'NAVIGATION' with links for 'Overview', 'Source', 'Commits', 'Branches', 'Pull requests', 'Downloads', and 'Settings'.

The main content area is titled 'Pull requests' and 'Create a pull request'. It features a visual representation of the pull request process: a source repository 'danyisk / Courses - 2014 - OOP Laboratory' (created 2014-10-01, updated 17 minutes ago) with a 'default' branch is shown on the left, an arrow points to the right, and a target repository 'mcasadei/oop2014-lab' with a 'default' branch is shown on the right.

Below this, the 'Title' field is empty. The 'Description' field has a rich text editor toolbar with icons for bold, italic, link, and other formatting options, and a 'Preview' button. The 'Reviewers' field contains the text 'Start typing to search for a user'. Below it, the 'Add:' field shows two user avatars: 'mviroli' and 'mcasadei'. At the bottom, there is a 'Close branch' section with a checkbox labeled 'Close default after the pull request is merged'.



# Bitbucket Overview

Bitbucket Dashboard Teams Repositories Create owner/repository

## Fork danysk / Courses - 2014 - OOP Laboratory

You can also [create a patch queue](#)

Name\*

Description

It's encouraged to write a little about why you are forking.

Access level  This is a private repository

Permissions  Inherit repository user/group permissions

Project management  Issue tracking  
 Wiki


### Repository integrations

HipChat  Enable HipChat notifications

Fork at

### What is a fork?

Forking is a great way to contribute to a project even though you don't have write access. Check out our documentation for more details.





# Bitbucket Overview

Bitbucket Dashboard Teams Repositories Create owner/repository

danyisk Courses - 2014 - OOP Labo...

## Settings

**Repository details**

Name\* Courses - 2014 - OOP Laboratory

Size 39.9 MB

Description

Access level  This is a private repository

Phases  This is a non-publishing repository

Allow draft changesets to be pushed to this repository. For more information, see [hg help phases](#).

Landing page\* Overview

Website

Language Other

Main branch default

Change logo

**ACTIONS**

- Clone
- Create branch
- Create pull request
- Compare
- Fork

**NAVIGATION**

- Overview
- Source
- Commits
- Branches
- Pull requests
- Downloads
- Settings

**GENERAL**

- Repository details
- Access management
- Branch management
- Username aliases
- Deployment keys
- Strip changesets
- Transfer repository
- Delete repository

**INTEGRATIONS**

- HipChat integration
- Hooks
- Links

**ISSUES**

- Issue tracker settings

**WIKI**



# Bitbucket Overview

The screenshot shows the Bitbucket web interface. At the top is a dark blue navigation bar with the Bitbucket logo, 'Dashboard', 'Teams', 'Repositories', and a 'Create' button. On the right of the bar is a search bar containing 'owner/repository', a help icon, and a user profile icon.

The main content area is titled 'Settings' and is divided into three columns:

- Left Column (Navigation):** Contains a sidebar with a 'danyks' profile and repository name 'Courses - 2014 - OOP Labo...'. Below this are sections for 'ACTIONS' (Clone, Create branch, Create pull request, Compare, Fork) and 'NAVIGATION' (Overview, Source, Commits, Branches, Pull requests, Downloads, Settings).
- Middle Column (Menu):** Lists various settings categories: GENERAL (Repository details, Access management, Branch management, Username aliases, Deployment keys, Strip changesets, Transfer repository, Delete repository), INTEGRATIONS (HipChat integration, Hooks, Links), ISSUES (Issue tracker settings), and WIKI.
- Right Column (Access management):** Titled 'Access management', it shows a 'Users' table with columns for 'Username or email address' and 'Permissions'. The table lists three users: 'Danilo Pianini' (owner), 'mcsadei' (READ, WRITE, ADMIN), and 'mviroli' (READ, WRITE, ADMIN). Below the table is a 'Groups' section with a 'Select a group' dropdown, a 'Read' dropdown, and an 'Add' button.





# Bitbucket Overview

Bitbucket Dashboard Teams Repositories Create owner/repository

Manage Danilo Pianini

### GENERAL

- Account settings**
- Email addresses
- Notifications
- Custom domain
- Change username
- Delete account

### PLANS AND BILLING

- Plan details
- Billing details

### ACCESS MANAGEMENT

- User groups
- OAuth
- HipChat integration

### SECURITY

- Change password

## Account settings

Username **danyisk** [\(change\)](#)

First name

Last name

Website


Location

**BETA** Language

Help translate Bitbucket into your language.

- Keyboard shortcuts
- Private profile

[Save settings](#) [Cancel](#)



[Change avatar](#)  
Current avatar via [Gravatar](#).



# Bitbucket Overview

Bitbucket Dashboard Teams Repositories Create owner/repository

Manage Danilo Pianini

**GENERAL**  
Account settings  
Email addresses  
Notifications  
Custom domain  
Change username  
Delete account

**PLANS AND BILLING**  
**Plan details**  
Billing details

**ACCESS MANAGEMENT**  
User groups  
OAuth  
HipChat integration

**SECURITY**  
Change password

**Current plan**  
Plan owner: Danilo Pianini  
Plan level: Academic  
[Change plan](#)

**Working in a team?**  
Bring all your repositories together in a Bitbucket Team account.  
[Create a team.](#)


**Users on this account (19 / Unlimited)**  
The number of users (including you) with any level of access to one or more of your private repos.


**Access via groups**

Alberto Rosi 1	<a href="#">Remove</a>
Andrea Leardini 1	<a href="#">Remove</a>
Bernhard Anzengruber 1	<a href="#">Remove</a>
Davide Ensini 1	<a href="#">Remove</a>
Gabriella Castelli 1	<a href="#">Remove</a>
Giovanni Ciatto 1	<a href="#">Remove</a>



# Bitbucket Overview

☰ **Bitbucket** Dashboard ▾ Teams ▾ Repositories ▾ Create owner/repository ? 

**Manage**  Danilo Pianini ▾

**GENERAL**

- [Account settings](#)
- [Email addresses](#)
- [Notifications](#)
- [Custom domain](#)
- [Change username](#)
- [Delete account](#)

**PLANS AND BILLING**

- [Plan details](#)
- [Billing details](#)

**ACCESS MANAGEMENT**

- [User groups](#)
- [OAuth](#)
- [HipChat integration](#)

**SECURITY**

- [Change password](#)

## SSH keys

Use [SSH](#) to avoid password prompts when you push code to Bitbucket. Learn how to [generate a SSH key](#).

[Add key](#)

Key	Added
APICe	Not recorded <a href="#">Edit</a> <span>✕</span>
ASUS N53S DEIS	Not recorded <a href="#">Edit</a> <span>✕</span>
Desktop Cesena	Not recorded <a href="#">Edit</a> <span>✕</span>
PC Scavolino	Not recorded <a href="#">Edit</a> <span>✕</span>
Server Maven Alchemist	2013-08-23 <a href="#">Edit</a> <span>✕</span>



# Bitbucket Overview

Bitbucket Dashboard Teams Repositories Create owner/repository

Manage Danilo Pianini

### Add SSH key

Label

Key\*

**Already have a key?**  
Copy your key to your clipboard with: `xclip -sel clip < ~/.ssh/id_rsa.pub`

Add key Cancel

Su sistemi \*nix, si utilizzi il comando `ssh-keygen -t rsa` per generare una coppia di chiavi pubblica e privata. Non si inserisca alcuna password e si mantengano le opzioni suggerite dal sistema. A questo punto, all'interno della cartella `~/.ssh/`, si troveranno i file delle chiavi. Si incolli nel campo key il contenuto del file `~/.ssh/id_rsa.pub`



## 1 Decentralized Version Control Systems

- Risolvere i merge conflict
- Lavorare in remoto: clone, fetch, pull, push
- Hosting e Bitbucket
- **Plugin Eclipse (eGit)**
- Features avanzate e prossimo episodio

## 2 Testing con JUnit



# Plugin eclipse

## Installazione

Il plugin è preinstallato dentro la distribuzione di Eclipse.

## Funzionalità

Il plugin mostrerà le proprie funzionalità dal menu “Team”, ed esporrà le stesse funzionalità del tool a linea di comando, **mostrando anche quelle che non abbiamo trattato, e che non dovete utilizzare se non avete cristallino il loro funzionamento da terminale.**

Appare anche una nuova opzione per l'import di un progetto tramite clone di un repository Git.

È bene (come per tutte le utility con interfaccia grafica) fare un utilizzo **consapevole** del plugin Eclipse.

Vi consigliamo di preferire l'uso di Git da terminale!



## 1 Decentralized Version Control Systems

- Risolvere i merge conflict
- Lavorare in remoto: clone, fetch, pull, push
- Hosting e Bitbucket
- Plugin Eclipse (eGit)
- Features avanzate e prossimo episodio

## 2 Testing con JUnit



# Aspetti avanzati (cenni)

## Rebasing

Procedura alternativa al merge, in cui i due commit, invece di essere fusi, vengono messi in sequenza. Simula una storia “lineare” anche dove è in realtà parallela.

## Cherry picking

Pull di un singolo commit o di un piccolo gruppo di commit. Spesso utilizzato quando si desidera avere un bugfix che si trova in un altro branch, ma non tutto il resto. Molto comune nel backporting delle patch di sicurezza.

## Bisection

Strategia per scoprire bug in maniera automatizzata, testando il software a varie versioni (ricerca dicotomica). Una volta trovata la prima versione dove il bug si verifica, si controllano commit message e le differenze.



## Nel prossimo episodio (verso fine corso)

- Abbiamo in mano uno strumento formidabile per la gestione di progetti
- Non ci resta che trovare una strategia efficace per massimizzarne l'efficacia
- Vedremo una strategia di lavoro che promuoverà e faciliterà il lavoro in parallelo



- 1 Decentralized Version Control Systems
  - Risolvere i merge conflict
  - Lavorare in remoto: clone, fetch, pull, push
  - Hosting e Bitbucket
  - Plugin Eclipse (eGit)
  - Features avanzate e prossimo episodio
- 2 Testing con JUnit



# Importanza del testing

(Tutte cose che discuterete meglio a lezione)

- Il testing è importante quanto il sistema stesso
- Si dovrebbero scrivere **prima** i test, e solo dopo realizzare il sistema
  - ▶ Consente di dare una specifica formale del comportamento delle entità che si stanno modellando
  - ▶ Consente di intercettare immediatamente eventuali problemi in fase di sviluppo
  - ▶ Fornisce una base per effettuare “regression testing”, ossia per evitare che nuove funzionalità introducano dei bug
  - ▶ È vero per tutte le branche dell'ingegneria: prima ancora di realizzare un motore, si prepara il suo banco prova
- Esistono degli strumenti appositi che facilitano il testing!



## Unit testing

- Scopo ristretto: testa “piccole parti di codice”
  - ▶ idealmente, una sola “unità”, ad esempio una sola implementazione di una sola interfaccia
- Servono a supportare lo sviluppatore
  - ▶ L'utente ne beneficia indirettamente, ottenendo software con meno difetti
- Non hanno dipendenze da sistemi esterni a quello in test
- Testano la consistenza interna



# Unit vs integration II

## Integration testing

- Scopo generale: verifica che diverse parti del sistema lavorino bene insieme
- Hanno frequentemente dipendenze da sistemi esterni
- Molto più impegnativi da realizzare
  - ▶ Possono richiedere anche risorse esterne
    - Database
    - Hardware apposito
    - Macchine virtuali
- Testano il funzionamento globale del sistema
  - ▶ Tanto che a volte si fanno dei veri e propri dimostratori...



## Generalità

JUnit è il framework per lo unit testing più usato nel mondo Java

- Probabilmente, la libreria Java (escluso il JDK) più usata in assoluto
- Sfrutta alcune funzionalità di Java che conoscerete solo a fine corso
  - ▶ Reflection e annotazioni



## Funzionamento

- Si crea una classe di test
  - ▶ Normalissima classe Java, come tutte le altre
- Si creano uno o più metodi con:
  - ▶ Modificatore di visibilità `public`
  - ▶ Tipo di ritorno `void`
  - ▶ Nome a piacere (formato suggerito: `testThingYouAreTesting`)
  - ▶ Nessun argomento
  - ▶ Annotazione `@Test`
    - Ossia, si scrive `@Test` prima di `public`
    - L'annotazione va importata
- Nel corpo dei metodi:
  - ▶ Si creano e usano le entità che si vogliono testare
  - ▶ Si verificano le proprietà usando le *asserzioni* fornite da JUnit
- I metodi annotati saranno eseguiti senza necessità di alcun `main`

## Assertzioni

Sono particolari metodi che verificano una certa condizione, e fanno fallire il test se non è rispettata

- Metodi statici della classe `org.junit.Assert`
- `assertTrue(boolean)` — verifica che la condizione passata sia `true`
- `assertFalse(boolean)` — verifica che la condizione passata sia `false`
- `assertEquals(Object, Object)` — verifica che i due argomenti passati siano uguali (chiama la `equals`)
- `assertEquals(double, double, double)` — verifica che il modulo della differenza fra i primi due argomenti sia inferiore al terzo
  - ▶ Non si deve mai fare `==` fra numeri in virgola mobile
- `assertNotEquals` — reciproco di `assertEquals`
- `fail()` — Fa fallire il test immediatamente
  - ▶ Ad esempio se non viene generata una eccezione che ci si sarebbe attesa

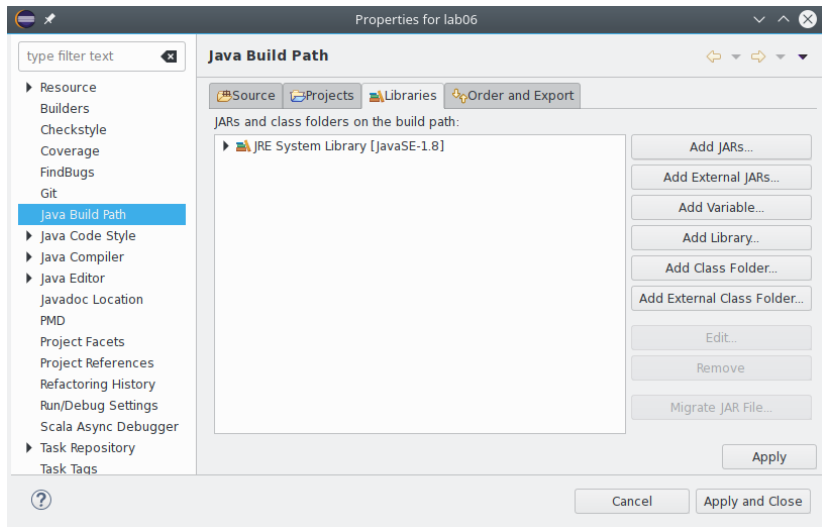


# JUnit in Eclipse I

Per usare una libreria in Eclipse, occorre segnalare all'IDE che vogliamo inserirla nel classpath. Per JUnit, esiste un percorso “di favore” (viene fornita con Eclipse, non occorre installare alcunché).



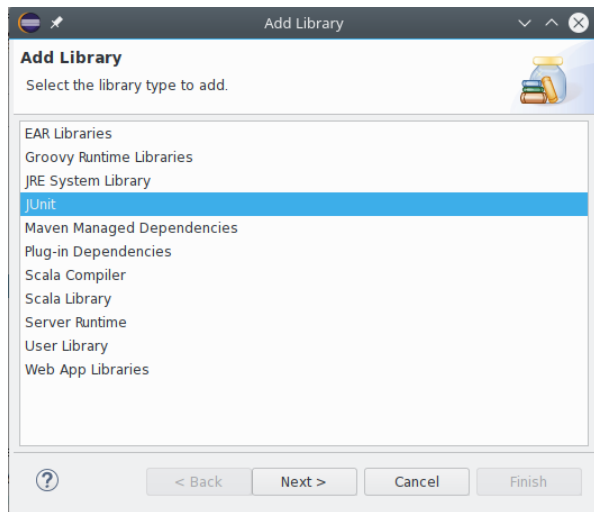
# Junit in Eclipse II



click su “Add library”



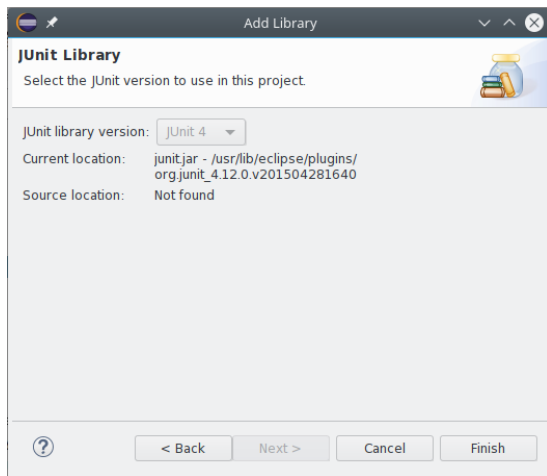
# JUnit in Eclipse III



click su "Next"



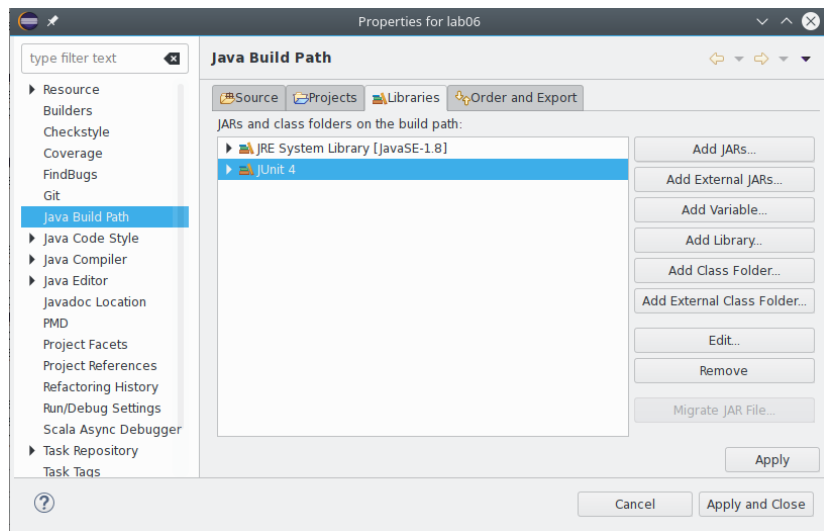
# JUnit in Eclipse IV



click su "Finish"



# Junit in Eclipse V



La libreria appare nel classpath. Click su “Apply and Close”



# Junit in Eclipse VI

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays a project structure with several packages and files. The file 'BaseRobotTest.java' is selected and highlighted in blue. A context menu is open over this file, listing various actions such as 'New', 'Open', 'Copy', 'Paste', 'Delete', 'Build Path', 'Source', 'Refactor', 'Import...', 'Export...', 'References', 'Declarations', 'Find Bugs', 'Refresh', 'Assign Working Sets...', 'Coverage As', 'Run As', 'Debug As', 'Profile As', and 'Validate'. The 'Run As' option is highlighted, and its sub-menu is visible, showing 'Run on Server' and 'JUnit Test' as options. The 'JUnit Test' option is also highlighted in blue. In the background, a snippet of Java code is visible, showing a class 'BaseRobotTest' with a 'test' method that calls 'move' and 'throwException' methods.

Per eseguire, si usa “Run As” – “JUnit Test”

