Distributed Version Control Systems II
Branching e merging
Programmazione ad Oggetti – Lab05

Docenti: Danilo **Pianini**, Roberto **Casadei**
Tutor: Simone **Costanzi**, Luca **Tremamunno**

C.D.S. Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Campus di Cesena

25 ottobre 2021

Outline

Outline

Concetti basilari e terminologia I

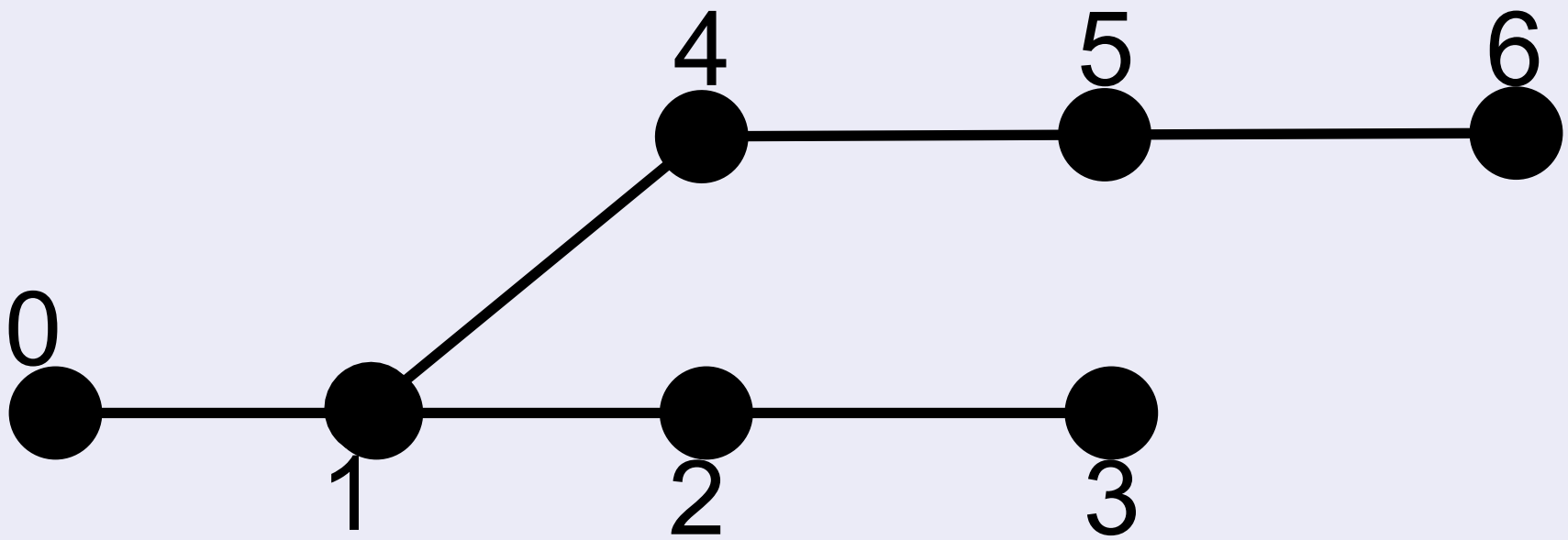
Navigazione della storia

Possibilità di tornare ad un qualunque commit (salvataggio) precedente o successivo

Concetti basilari e terminologia II

Branch

Linea di sviluppo (ossia sequenza di commit successivi). Dal momento in cui si può tornare indietro nella storia dei salvataggi e ripartire a sviluppare, si può creare una nuova linea di sviluppo, che si diparte da quella originale e procede parallelamente.



Concetti basilari e terminologia III

HEAD

Identifica la posizione corrente all'interno della storia del repository. È in sostanza un riferimento ad uno specifico commit. In una normale sessione di lavoro, la `HEAD` è posizionata alla fine di un branch. Quando si naviga nella storia del repository, lo si fa spostando la `HEAD`.

Concetti basilari e terminologia IV

Nomi di branch

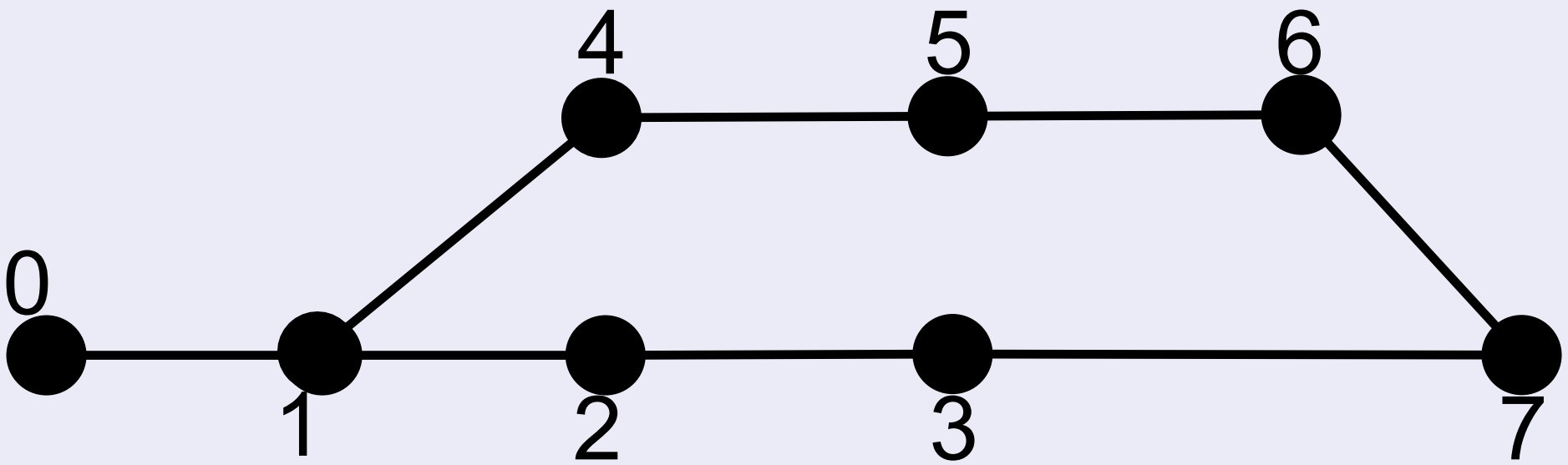
Per identificare i branch, viene loro assegnato un nome

- In git ogni branch ha un nome (assegnato manualmente)
- Se non viene specificato, viene creato un branch di default
- Tradizionalmente, il nome di default è `master`
  - Deriva da BitKeeper, che usava quel nome
  - Il termine, a partire dal 2020, è considerato offensivo da alcuni
    - `https://sfconservancy.org/news/2020/jun/23/gitbranchname/`
  - Versioni recenti di git consigliano la configurazione del default branch name con un warning al primo `init`
  - Il nome di default è settabile con
    - `git config --global init.defaultBranch <BRANCH-NAME>`
- Alla creazione di nuovi branch, va assegnato loro un nome

Concetti basilari e terminologia V

Merge

Fusione di due branch (linee di sviluppo) in una sola.



Outline

Visualizzazione della storia I

In generale

Visualizzare l'elenco dei commit effettuati, chi li ha eseguiti, quando, ed il loro message commit

Visualizzazione della storia II

In Git

Git offre il sottocomando `log`

- `git log`
  - Visualizza tutti i commit della linea di sviluppo corrente
  - Se l'output è troppo lungo crea una visualizzazione scorrevole (si vedano i comandi Unix `less` e `more`)
  - Per uscire dalla visualizzazione scorrevole, si usa il tasto Q
- `git log --graph`
  - Come sopra, con visualizzazione grafica dell'evoluzione sulla sinistra
- `git log --graph --oneline`
  - Come sopra, visualizzazione compatta con un commit per linea

Visualizzazione della storia III

Esercizio

- Si riparta dal repository contenente l'esercizio eseguito nella lezione precedente
  - Nel caso in cui non fosse stato completato, esso è reso disponibile assieme agli esercizi
- Si visualizzi l'attuale storia del repository, corredata di grafico

Visualizzazione della storia IV

Output atteso

```
* commit 47b5f2fb9f5300dc8bc530ce45d37a86a0436755
| Author: Danilo Pianini <danilo.pianini@unibo.it>
| Date:   Wed Oct 19 16:46:21 2016 +0200
|
|     Remove the trash
|
* commit 4d086a9b0d2139f0cd300d329f532a2c464304c7
| Author: Danilo Pianini <danilo.pianini@unibo.it>
| Date:   Wed Oct 19 16:45:28 2016 +0200
|
|     move junk to trash
|
* commit 844aebd840e6f3d2b034312e9fa37677f64b9a15
| Author: Danilo Pianini <danilo.pianini@unibo.it>
| Date:   Wed Oct 19 16:43:48 2016 +0200
|
|     Add junk
|
* commit 3ae84225f45afdfa02c268c6079e0f6c96695c1f
| Author: Danilo Pianini <danilo.pianini@unibo.it>
| Date:   Wed Oct 19 16:26:30 2016 +0200
|
|     Create .gitignore
|
* commit 19aa252373d1e44897233bf5b733cf82019cd5bf
  Author: Danilo Pianini <danilo.pianini@unibo.it>
  Date:   Wed Oct 19 15:51:10 2016 +0200

      Create HelloWorld
```

Riferirsi a commit I

In git esistono vari modi di riferirsi ad un commit. Un riferimento valido è chiamato `<tree-ish>`.

Riferimenti validi

- Hash del commit, e.g., `b82f7567961ba13b1794566dde97dda1e501cf88`
- Versione accorciata dello hash (almeno 8 caratteri se non ambiguo), e.g., `b82f7567`
- Nomi di branch: puntano sempre all'ultimo commit appartenente ad un branch
- `HEAD`: riferimento al commit corrente

Riferirsi a commit II

Riferimenti relativi

Si possono indicare i commit precedenti ad un `<tree-ish>` usando `~` seguito dal numero di "passi indietro" da fare.

- `HEAD~1` fa riferimento al commit precedente a quello corrente
- `HEAD~2` fa riferimento a due commit precedenti al corrente
- `HEAD~N` fa riferimento al N-esimo commit effettuato prima dell'ultimo
- `b82f7567~1` fa riferimento al parent commit di `b82f7567`

Visualizzazione delle differenze I

In generale

Vogliamo poter controllare quali modifiche sono state introdotte da un commit o da una serie di commit

Visualizzazione delle differenze II

In Git

Git mostra le modifiche intercorse fra due commit col sottocomando `diff`

- `git diff`
  - Differenze fra working tree ultimo commit *escluso il contenuto in stage*
- `git diff --staged`
  - Come sopra, ma *include il contenuto in stage*
- `git diff <tree-ish>`
  - Mostra le differenze fra `<tree-ish>` e il working tree
  - `--staged` per includere il contenuto in stage
- `git diff <FROM> <TO>`
  - Dove `<FROM>` e `<TO>` due `<tree-ish>`
  - Mostra le differenze fra `<FROM>` e `<TO>`

Il formato dell'output è lo stesso del comando Unix `diff`, è interpretabile da quest'ultimo e può essere utilizzato per creare delle "patch".

Visualizzazione delle differenze III

Esercizio

- Osservare le differenze fra la staging area e i tre commit precedenti (`HEAD~2`)

Output atteso

```
diff --git a/junk.txt b/junk.txt
deleted file mode 100644
index de0a8c8..0000000
--- a/junk.txt
+++ /dev/null
@@ -1 +0,0 @@
-some trash
```

Visualizzazione delle differenze IV

Spiegazione dell'output

- È stato cancellato un file con permessi ottali 644
- è stato spostato `junk.txt` dentro `/dev/null`
  - Ossia cancellato
  - `/dev/null` è uno speciale device file Unix (Io vedrete in Sistemi Operativi)
- Una modifica ha rimosso una linea a partire dalla riga 0, colonna 0
- Il contenuto della riga rimossa è `some trash`

Outline

Navigazione della linea di sviluppo I

In generale

Vogliamo poter ritornare a qualunque salvataggio della nostra linea di sviluppo

Navigazione della linea di sviluppo II

In Git

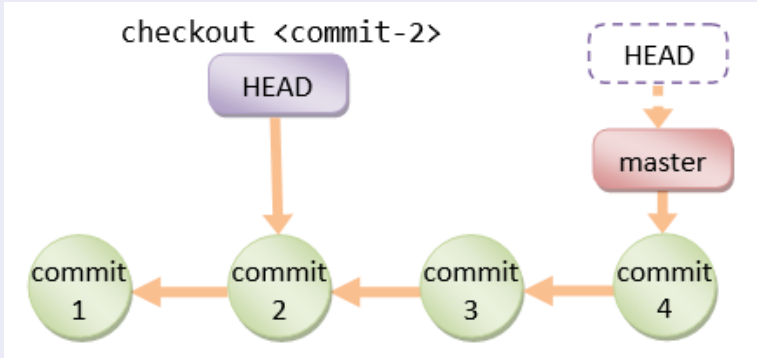
Il sottocomando `checkout` consente di ripristinare una versione precedente di un file o dell'intero repository

- `git checkout <tree-ish>`
  - Dove `<tree-ish>` è lo hash di un commit o un suo riferimento, ad esempio:
    - `HEAD`
    - `master`, o altro nome di branch
  - Se non ci sono modifiche che rischiano di essere perse, torna al salvataggio `<tree-ish>`
- `git checkout <tree-ish> -- FILENAME`
  - Ripristina il file `FILENAME` prendendolo dal commit `<tree-ish>`

Navigazione della linea di sviluppo III

La modalità detached HEAD

Quando si torna indietro nella storia, Git entra in modalità "detached HEAD": la "testa", ossia il commit a cui ci troviamo, è staccato dalla "cima" della linea di sviluppo.



- I commit effettuati in questa modalità verranno scartati
- Per tornare alla modalità "attached", è necessario effettuare un checkout con il nome del branch
  - Il nome del branch punta sempre all'ultimo commit su quella linea di sviluppo
  - Ad esempio: `git checkout master`

Navigazione della linea di sviluppo IV

Esercizio

Premessa: si osservi lo stato del repository con `git status` **prima e dopo ogni operazione**, assicurandosi di capire appieno l'output fornito da Git

- Si recuperi il file `junk.txt` da tre commit fa (`HEAD~2`)
- Si vada al primo commit, usando il suo hash
  - Potete ottenerlo usando `git log`
- Si torni alla cima della linea di sviluppo (`master`)
- Si rimuova il file `junk.txt` dall'area di staging (con `reset`)
- Si elimini il file `junk.txt`

Output atteso

```
On branch master
nothing to commit, working tree clean
```

Navigazione della linea di sviluppo V

Output atteso

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   junk.txt
```

Navigazione della linea di sviluppo VI

Output atteso

```
A       junk.txt
Note: checking out '19aa252373d1e44897233bf5b733cf82019cd5bf'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 19aa252... Create HelloWorld
```

Navigazione della linea di sviluppo VII

Output atteso

```
HEAD detached at 19aa252
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   junk.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        bin/
```

Output atteso

```
A       junk.txt
Previous HEAD position was 19aa252... Create HelloWorld
Switched to branch 'master
```

Navigazione della linea di sviluppo VIII

Output atteso

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   junk.txt
```

Output atteso

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        junk.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Navigazione della linea di sviluppo IX

Output atteso

```
On branch master
nothing to commit, working tree clean
```

Spiegazione dell'output

- Si noti che Git cerca di non cancellare le modifiche non salvate in un commit quando si naviga la storia (il file `junk.txt` non viene cancellato)
- In caso di conflitti, si rifiuterebbe di cambiare commit
- Si risolve cancellando le modifiche fatte o creando un nuovo commit, in modo da tornare allo stato di "`working tree clean`"

Outline

Creazione di nuove linee di sviluppo (branching) I

In generale

Vogliamo poter sviluppare su più linee

- Ad esempio perché stiamo per sviluppare una funzionalità che non sapremo se e quando completeremo, ma nel frattempo il nostro software va comunque manutenuto
- Ad esempio perché vogliamo sviluppare qualcosa a partire da una versione più vecchia (ossia, salvare i commit effettuati in modalità "detached `HEAD`"

Creazione di nuove linee di sviluppo (branching) II

In Git

Il sottocomando `branch` consente di creare e cancellare branch

- `git branch`
  - Stampa i branch, mostrando con `*` quello corrente.
- `git branch <branchname>`
  - Crea un nuovo branch di nome `<branchname>`
  - Non sposta `HEAD`
  - Di solito viene quindi seguito da un `checkout`
    - `git branch <branchname> && git checkout <branchname>`
- `git checkout -b <branchname>`
  - Scorciatoia per il comando composto visto sopra

L'opzione `--all` di `git log` visualizza la storia per tutti i branch

- `git log --all --graph`

Creazione di nuove linee di sviluppo (branching) III

Esercizio

Premessa: si osservi e comprenda lo stato del repository con `git status` **prima e dopo ogni operazione** di modifica di file o `commit`

- Si visualizzino i branch disponibili
- Si crei un nuovo branch di nome `feature/readme`
- Si visualizzino i branch disponibili
- Si passi al branch `feature/readme`
- Si visualizzino i branch disponibili
- Si crei un nuovo file `README.md`, con del testo semplice
- Si aggiunga `README.md` alla staging area
- Si effettui il commit
- Si passi al branch `master`
- Si modifichi la stampa di `HelloWorld.java`
- Si aggiunga `HelloWorld.java` alla staging area
- Si effettui il commit
- Si visualizzi la storia dei commit su tutti i branch

Unione di più linee di sviluppo (merge) I

In generale

Vogliamo poter unire due linee di sviluppo in una sola

- Abbiamo sviluppato in un branch separato una nuova funzionalità, ora è pronta e vogliamo unirla al resto

Unione di più linee di sviluppo (merge) II

In Git

Il sottocomando `merge` consente di unire due branch

- `git merge branchname`
  - Tenta di unire le modifiche di `branchname` al branch corrente
  - Prima di effettuare il merge, è necessario spostarsi sul branch destinazione (con `checkout`)
  - Se non ci sono conflitti, tutti i commit di `branchname` vengono aggiunti al branch corrente
  - Viene creato un nuovo commit (merge commit)
  - È accettabile non cambiare il messaggio di commit predefinito
  - La risoluzione dei conflitti sarà uno dei temi del prossimo laboratorio
- `git branch -d branchname`
  - Elimina il branch `branchname`
  - Se un branch non dovesse più servire, ad esempio perché tutte le sue modifiche sono state merse in un altro branch, è possibile rimuoverlo.

Unione di più linee di sviluppo (merge) III

Esercizio

Premessa: si osservi e comprenda lo stato del repository con `git status` **prima e dopo ogni operazione** di modifica di file o `merge`

- Si visualizzino i branch disponibili, ci si assicuri di essere su `master`
- Si faccia il merge di `feature/readme`
- Si visualizzino i file del repository con `ls -ahl` (Unix) o `dir` (Windows)
- Si visualizzi la storia dei commit su tutti i branch
- Si elimini il branch `feature/readme`
- Si visualizzino i branch disponibili
- Si visualizzi la storia dei commit su tutti i branch

Output atteso

```
  feature/readme
* master
```

Unione di più linee di sviluppo (merge) IV

Output atteso

```
On branch master
nothing to commit, working tree clean
```

Output atteso

```
Merge made by the 'recursive' strategy.
 README.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Output atteso

```
On branch master
nothing to commit, working tree clean
```

Unione di più linee di sviluppo (merge) V

Output atteso

```
total 2.0M
drwxr-xr-x 5 danysk users 4.0K Oct 20 16:15 .
drwxr-xr-x 6 danysk users 2.0M Oct 20 15:56 ..
drwxr-xr-x 2 danysk users 4.0K Oct 19 16:22 bin
drwxr-xr-x 8 danysk users 4.0K Oct 20 16:15 .git
-rw-r--r-- 1 danysk users    5 Oct 19 18:55 .gitignore
-rw-r--r-- 1 danysk users   25 Oct 20 16:15 README.md
drwxr-xr-x 2 danysk users 4.0K Oct 19 16:00 src
```

Unione di più linee di sviluppo (merge) VI

Output atteso

```
*   commit b68c8a48dc24bd1d948c2ac4409d8d91a000b8b6
|\  Merge: 56aa7aa 9261ec2
| | Author: Danilo Pianini <danilo.pianini@unibo.it>
| | Date:   Thu Oct 20 16:15:47 2016 +0200
| |
| |     Merge branch 'feature/readme'
| |
| * commit 9261ec24cfb56d7cd7ecc67d4eaa8add9c44b3ff
| | Author: Danilo Pianini <danilo.pianini@unibo.it>
| | Date:   Thu Oct 20 15:43:50 2016 +0200
| |
| |     Add README.md file
| |
* | commit 56aa7aaad47026911c2aa2b026f33293c3b4fe31
|/  Author: Danilo Pianini <danilo.pianini@unibo.it>
|   Date:   Thu Oct 20 15:47:57 2016 +0200
|
|       Modify HelloWorld
|
* commit 47b5f2fb9f5300dc8bc530ce45d37a86a0436755
| Author: Danilo Pianini <danilo.pianini@unibo.it>
| Date:   Wed Oct 19 16:46:21 2016 +0200
|
|     Remove the trash
|

...CONTINUA!
```

Unione di più linee di sviluppo (merge) VII

Output atteso

```
Deleted branch feature/readme (was 9261ec2).
```

Output atteso

```
* master
```

Unione di più linee di sviluppo (merge) VIII

Output atteso

```
*   commit b68c8a48dc24bd1d948c2ac4409d8d91a000b8b6
|\  Merge: 56aa7aa 9261ec2
| | Author: Danilo Pianini <danilo.pianini@unibo.it>
| | Date:   Thu Oct 20 16:15:47 2016 +0200
| |
| |     Merge branch 'feature/readme'
| |
| * commit 9261ec24cfb56d7cd7ecc67d4eaa8add9c44b3ff
| | Author: Danilo Pianini <danilo.pianini@unibo.it>
| | Date:   Thu Oct 20 15:43:50 2016 +0200
| |
| |     Add README.md file
| |
* | commit 56aa7aaad47026911c2aa2b026f33293c3b4fe31
|/  Author: Danilo Pianini <danilo.pianini@unibo.it>
|   Date:   Thu Oct 20 15:47:57 2016 +0200
|
|       Modify HelloWorld
|
* commit 47b5f2fb9f5300dc8bc530ce45d37a86a0436755
| Author: Danilo Pianini <danilo.pianini@unibo.it>
| Date:   Wed Oct 19 16:46:21 2016 +0200
|
|     Remove the trash
|

...CONTINUA!
```

Conclusioni

- Avete in mano uno strumento molto potente
- Usatelo sempre d'ora in poi
- A partire da questo laboratorio!
  - Effettuate almeno un commit alla fine di ogni esercizio prima di chiamarci per correggere
  - Se ve ne servono di più, fatene di più!

Nei prossimi laboratori

- Impareremo qualche strategia per il lavoro in efficace in team
- Impareremo a scambiarci commit via Internet
- Lo useremo per ottenere le esercitazioni (bye bye zip files)

Nel progetto d'esame

- Andrà consegnato sotto forma di repository Git
- L'uso corretti di Git sarà parte della valutazione