



UNIVERSITÀ DEGLI STUDI
DI MILANO

Ingegneria del software

Validare le impressioni

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 30, NO. 4, APRIL 2004

An Empirical Study of Open-Source and Closed-Source Software Products

James W. Paulson, Member, IEEE, Giancarlo Succi, Member, IEEE Computer Society, and Armin Eberlein, Member, IEEE Computer Society

Current Research:	Reference:	Metric:
Open source development fosters faster system growth.	Wheeler [13] O'Reilly [9] Raymond [12] Mockus <i>et al.</i> [7]	Overall project growth in functions over time.
	O'Reilly [9] Dalle and Jullien [2]	Overall project growth in LOC over time.
		Overall project growth in complexity over time.
Open source projects foster more creativity.	Wheeler [13] O'Reilly [9] Raymond [12] Mockus <i>et al.</i> [7] O'Reilly [9] Dalle and Jullien [2]	Functions added over time.
Open source projects succeed because of their simplicity.	Raymond [12]	Overall project complexity.
		Average complexity of all of the functions.
		Average complexity of the functions added.
Open source projects generally have fewer defects than closed source projects, as defects are found and fixed rapidly.	Mockus <i>et al.</i> [7] O'Reilly [9]	Functions modified over time.
		Functions modified as a percentage of total functions.
Open source projects are more modular than closed source projects.	Mockus <i>et al.</i> [7] O'Reilly [9]	Correlation between functions added and functions modified.

Current Research:	Metric:	Our Results:
Open source development fosters faster system growth.	Overall project growth in functions over time.	Does not support hypothesis. The project growth rates over time were similar for all of the projects in our analysis, open source projects did not exhibit significantly higher growth rates than closed source projects.
	Overall project growth in LOC over time.	
Open source projects foster more creativity.	Functions added over time.	Supports hypothesis. The growing rate of the open source projects in our analysis was greater than the closed source projects.
	Overall project complexity	
	Average complexity of all of the functions	
Open source projects succeed because of their simplicity.	Average complexity of the functions added.	Does not support hypothesis. The closed source projects in our analysis were generally simpler than the open source projects analyzed.
Open source projects generally have fewer defects than closed source projects, as defects are found and fixed rapidly.	Functions modified over time.	Supports hypothesis. The y-intercept of the linear approximation of the functions modified over time was higher in open source projects. The number of functions modified as a percentage of the total functions was generally higher in open source projects.
	Functions modified as a percentage of total functions.	
Open source projects are more modular than closed source projects.	Correlation between functions added and functions modified.	Does not support hypothesis. There was strong significant correlation between the growing rate and the changing rate in open source projects, but little to no correlation in closed source projects.



Vecchie sfide che si amplificano

- integrazione del software
 - modello a cascata
 - è una fase a sé stante
 - modello microsoft
 - stabilize & synchronize
 - modello XP
 - più volte al giorno, escludendosi a vicenda
 - F/LOSS
 - continuamente e senza coordinazione a priori



Nuove sfide

- Il team si sfalda
 - come comunicare?
 - Come tenersi uniti?
 - Come coordinarsi?
 - Come ottenere collaborazioni?

Strumenti di supporto

- Comunicazione
 - Internet
 - Forum
 - mantenere la *community* unita
 - rispondere ai dubbi delle *new entry*
- Sincronizzazione del lavoro e versioning
 - Sul codice
 - Sulla documentazione
- Automatizzazione della build
- Bug tracking

SCM: scenari

- gestisce un posto dove mettere i lavori mentre sono in evoluzione, permettendo di richiamare velocemente una qualunque versione memorizzata degli stessi
- Permette la condivisione di tali lavori con altri gestendo accessi contemporanei ed aiutando a gestire i conflitti
- permette tracciabilità

Collaborazioni



Software Configuration Management

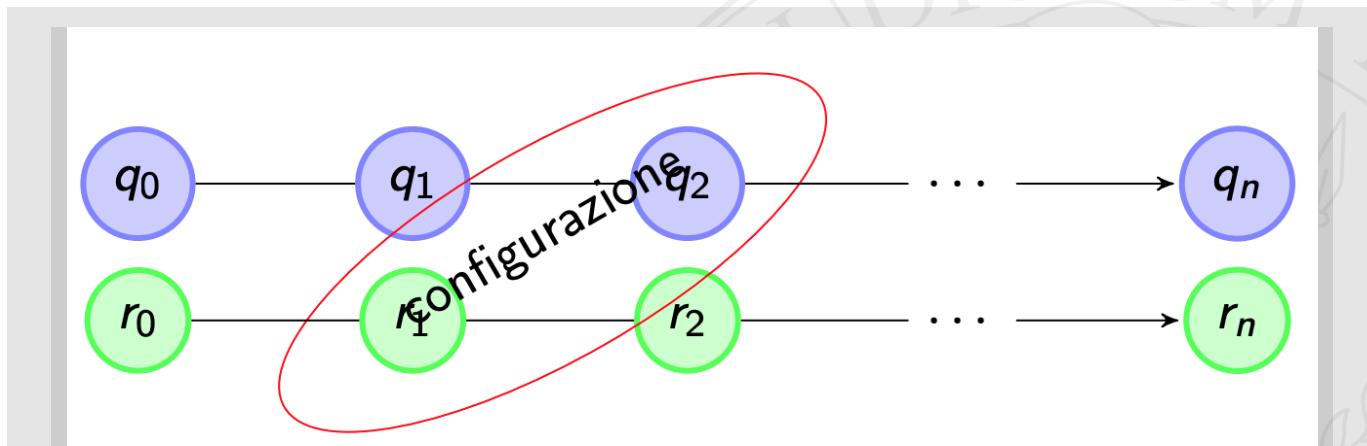
- Il Configuration Management nasce nell'industria aerospaziale negli anni '50. Alla fine degli anni '70 inizia a essere applicato nella produzione del software.

Pratiche che hanno l'obiettivo di rendere sistematico il processo di sviluppo, **tenendo traccia dei cambiamenti** in modo che il prodotto sia in ogni istante in uno stato (configurazione) ben definito.

- Gli “oggetti” di cui si controlla l’evoluzione sono detti *configuration item* o (in ambito sw) *artifact*.

Di cosa si occupano

- Gli *artifacts* classicamente sono file
- l'SCM permette di tracciare/controllare le revisioni degli *artifact* e le versioni delle risultanti configurazioni
- a volte fornisce supporto per la generazione del prodotto a partire da una ben determinata configurazione



Gli SCM sono per lo più indipendenti da linguaggi di programmazione e applicazioni (una notevole eccezione è Monticello di Smalltalk): lavorano genericamente su file, preferibilmente fatti di **righe di testo**

- anni '80: strumenti locali (SCCS, rcs, ...)
- anni '90: strumenti client-server centralizzati (CVS, subversion, ...)
- anni 2000: strumenti distribuiti *peer-to-peer* (git, mercurial, bazaar, ...)

Cosa viene tracciato?

- Qualunque sistema si usi, occorre prendere due decisioni importanti, che influenzano la **replicabilità** della produzione
 1. Si traccia l'evoluzione anche di componenti fuori dal nostro controllo (library, compilatori, ecc.) ?
 2. Si archiviano i file che costituiscono il prodotto?

In entrambi i casi la risposta più comune è **NO**, perché molto *costoso* e *poco pratico* ma in questo caso la perfetta replicabilità è perduta

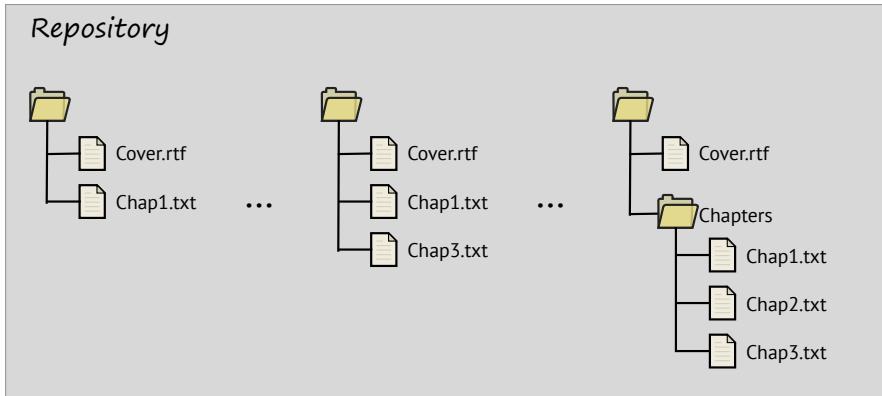
Meccanismo base

- Il meccanismo di base per controllare l'evoluzione delle revisioni è che ogni cambiamento è regolato da:
 - **check-out** dichiara la volontà di lavorare partendo da una particolare revisione di un *artifact* (o una configurazione di diversi artifacts)
 - **check-in** (o **commit**) dichiara la volontà di registrarne una nuova (spesso chiamata **change-set**)

Queste operazioni vengono attivate rispetto a un *repository*

Cioè producono spostamenti tra il *repository* (che contiene tutte le configurazioni) e il *workspace* (l'ambiente su cui si lavora nel filesystem)

Repository



- La repository mantiene informazioni comprese date, etichette (tag), versioni, diramazioni (branches), etc
- La repository può salvare solo le differenze tra una versione e l'altra
- Può essere centralizzata o distribuita

Regolazione del lavoro concorrente

- Quando il *repository* è condiviso da un gruppo di lavoro, nasce il problema di gestirne l'accesso concorrente:
 - modello “pessimistico” (RCS): il sistema gestisce l'accesso agli *artifact* in mutua esclusione attivando un **lock** al check-out
 - modello “ottimistico” (CVS): il sistema si *disinteressa* del problema e fornisce supporto per le attività di **merge** di *change-set* paralleli potenzialmente conflittuali

Il modello ottimistico può però essere parzialmente regolato tramite i rami paralleli di sviluppo (**branch**)

SCM ... distribuito?

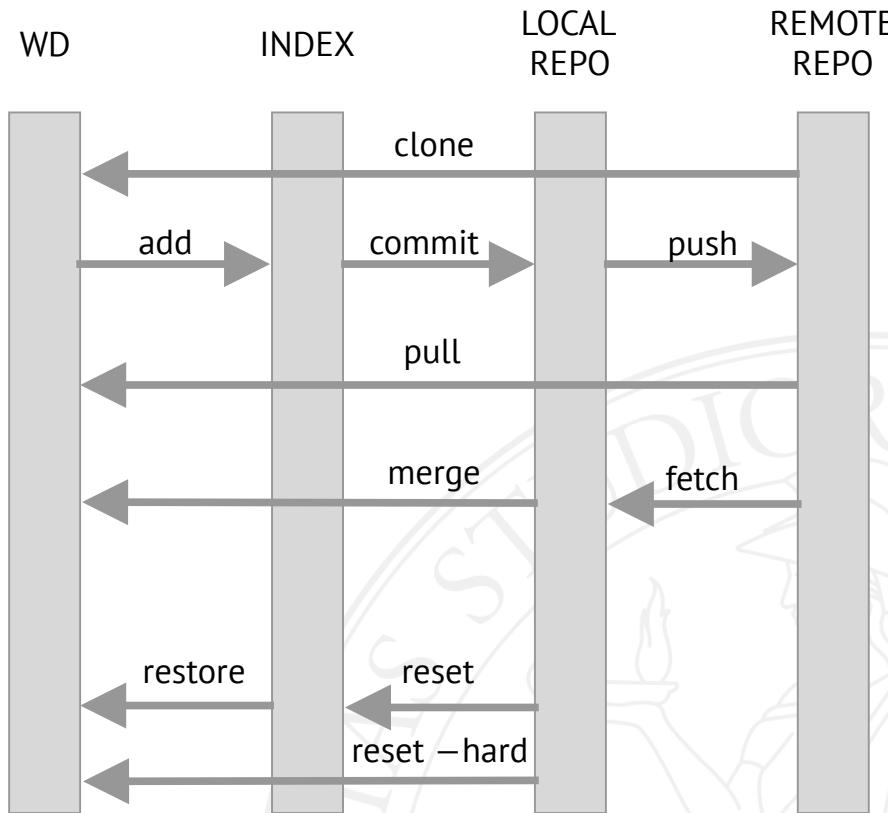
- Ogni peer ha un *repository* e non c'è sincronizzazione automatica
 - Comandi esplicativi per fare “merge” con un altro *repository*

VANTAGGI

- possibile work offline
- velocità
- Supporta diversi modi di lavorare:
 - simil centralizzato: un repository viene considerato "di riferimento"
 - due peer che collaborano direttamente
 - gerarchico a più livelli



Git common command/operation



escape a git mess, step-by-step»
discover character entities at &what»

GIT CHEATSHEET

de | en | es | fr | it | pt | 한국어 | vn | 简中 | 繁中

AN INTERACTION FROM NDP SOFTWARE

STASH

WORKSPACE

INDEX

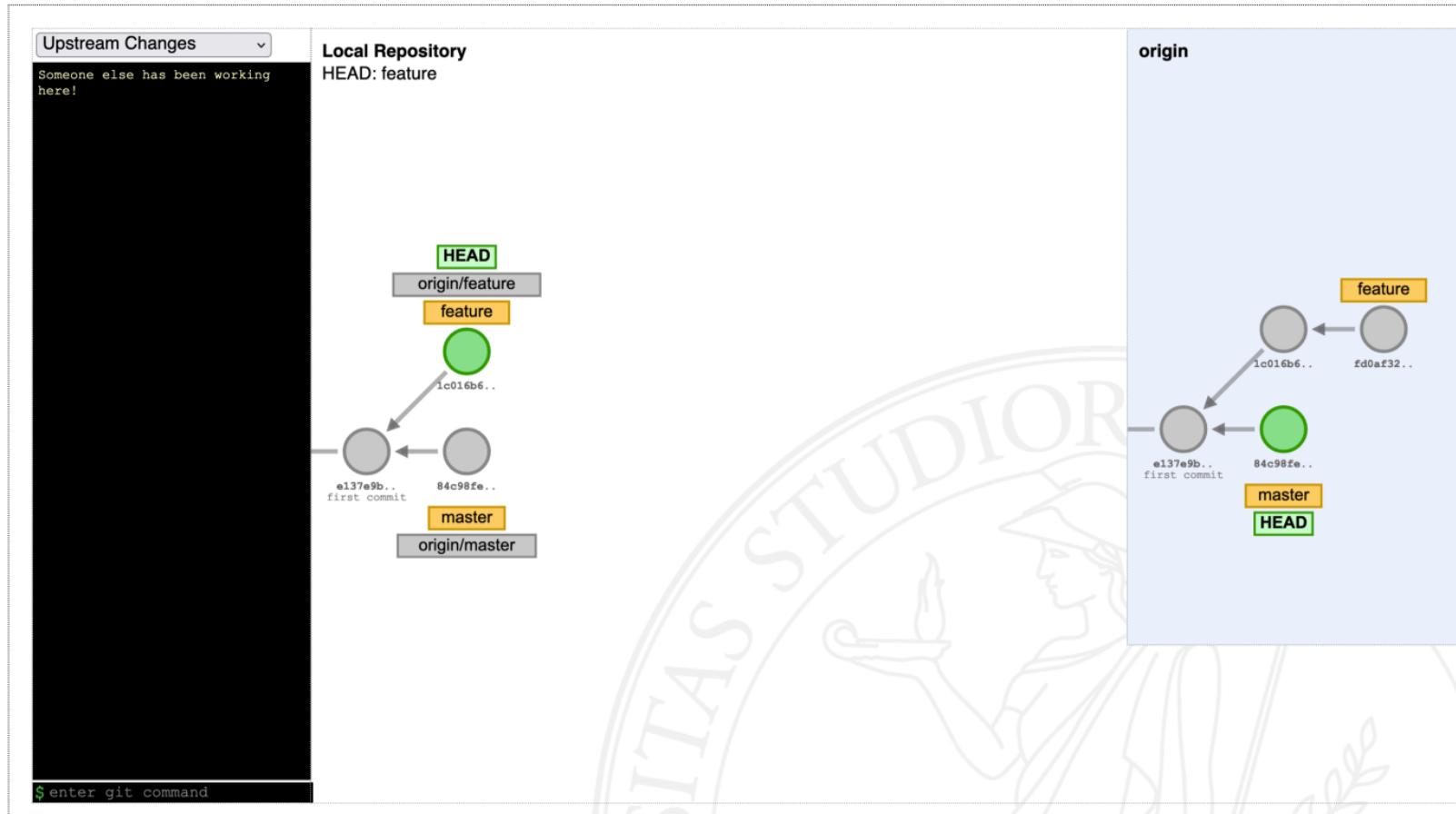
REPOSITORY LOCALE

REPOSITORY REMOTO

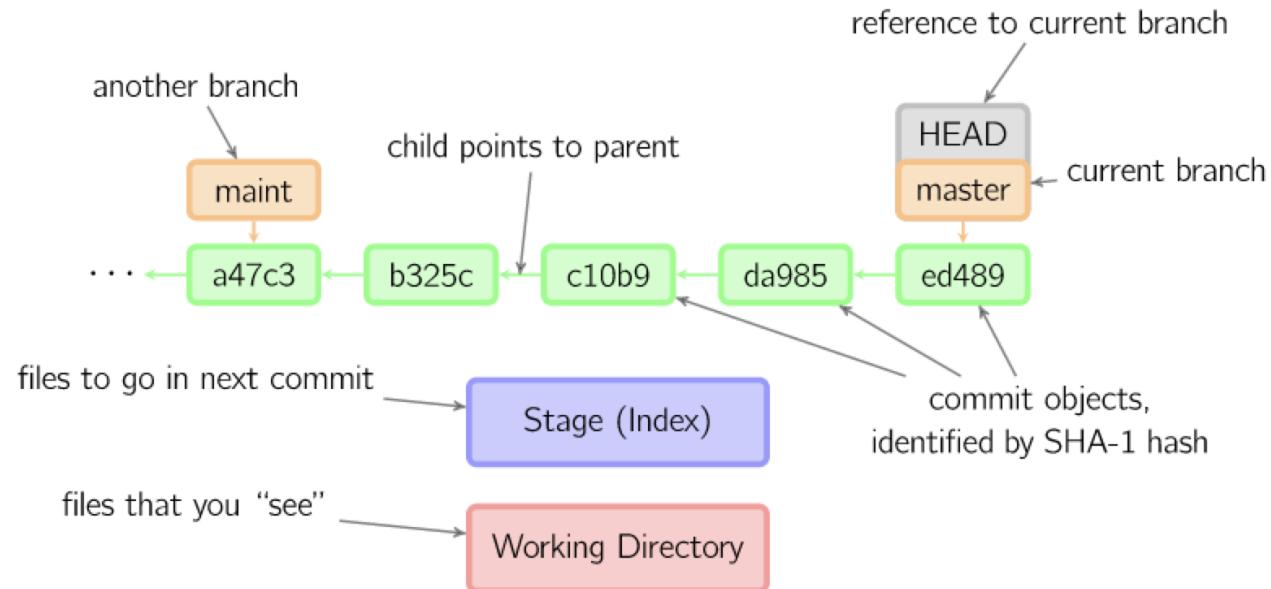
```
status  
diff  
add file... or dir...  
add -u  
rm file(s)...  
mv file(s)...  
(checkout file(s)... or dir)
```

```
reset --hard file(s)...  
reset --soft HEAD^  
diff --cached (commit)  
commit [-m 'msg']  
commit --amend
```

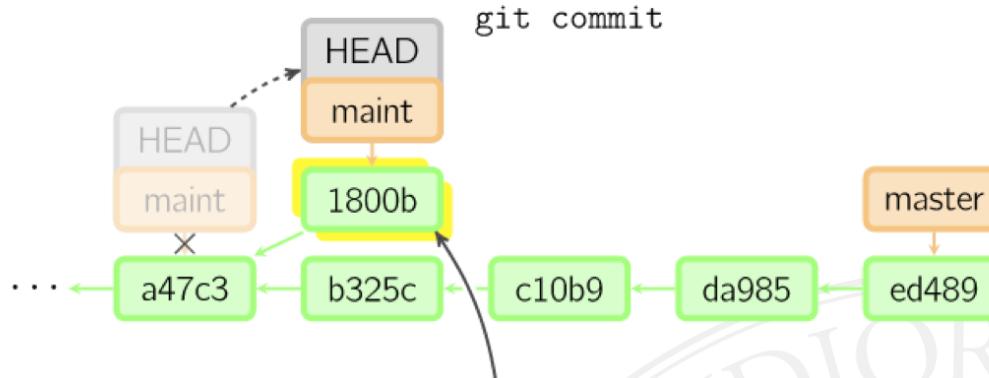




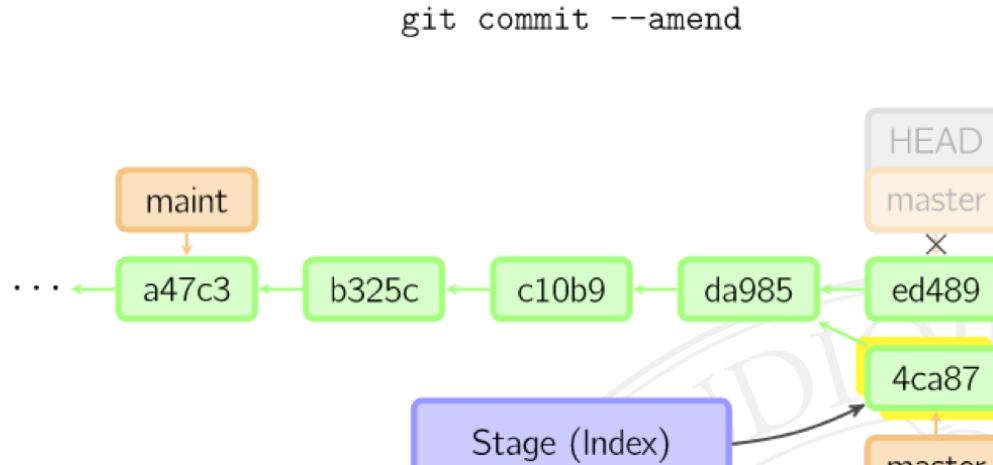
git



git commit

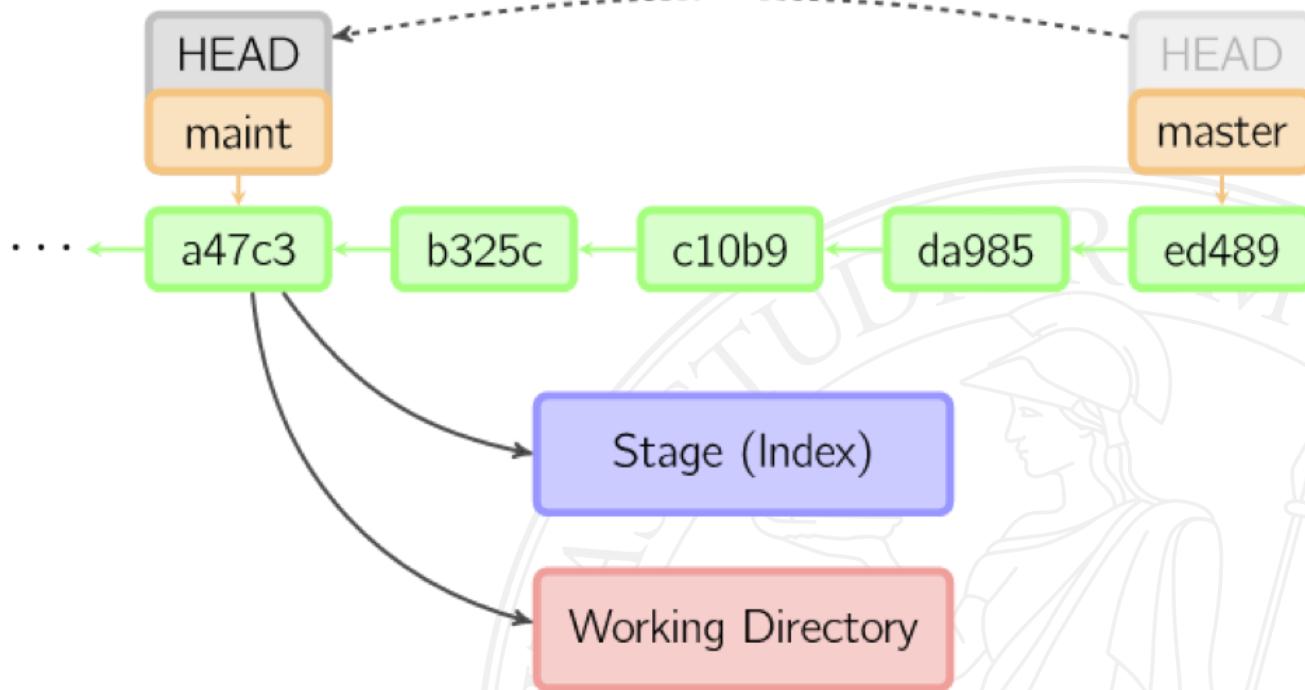


git commit --amend

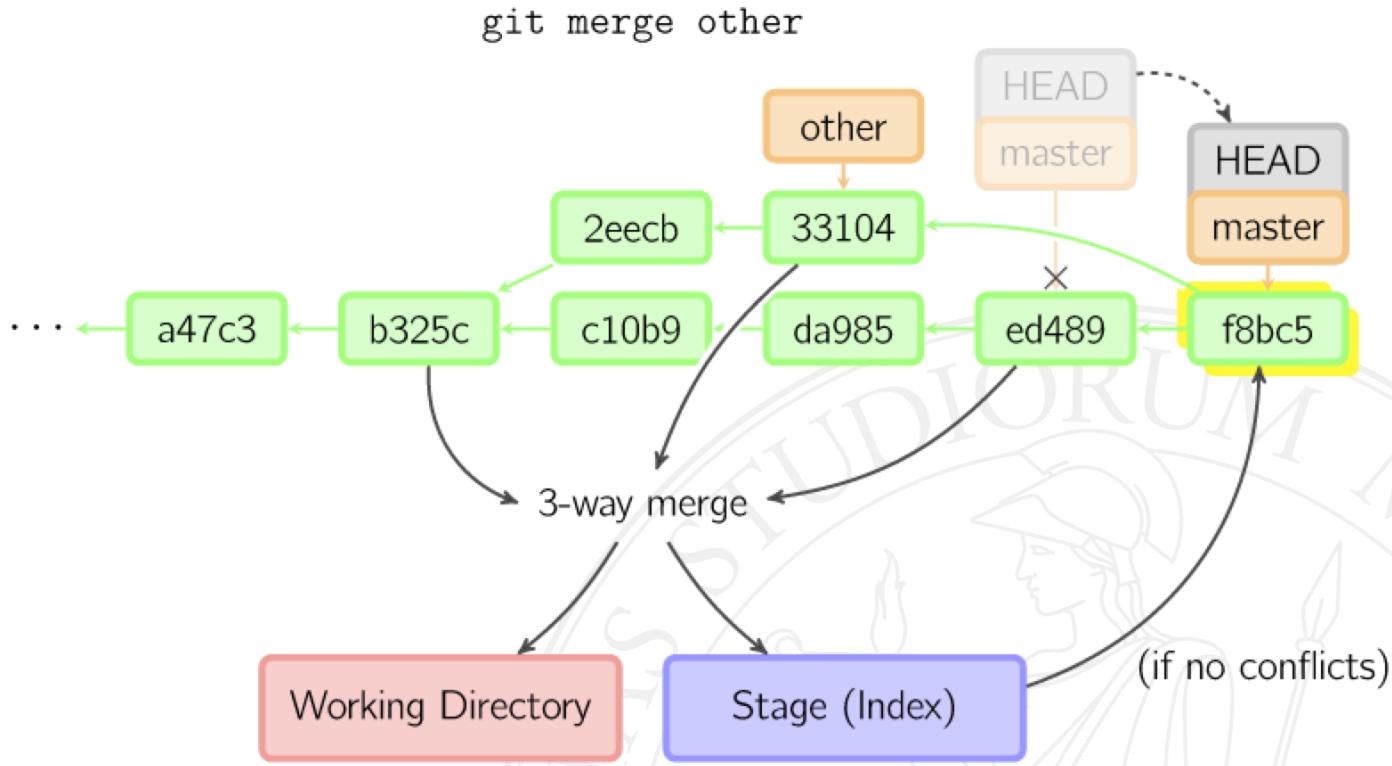


git checkout

git checkout maint

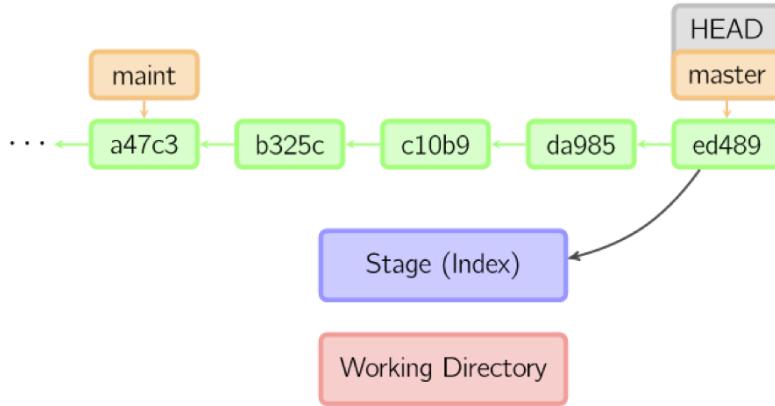


git merge

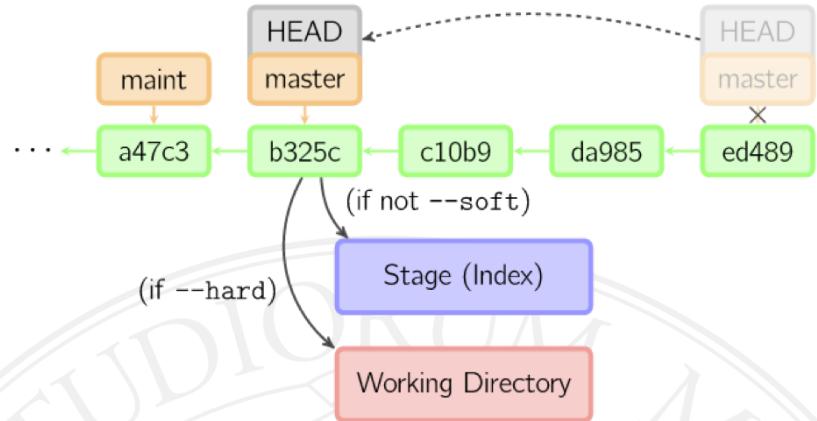


git reset

`git reset -- files`



`git reset HEAD~3`



git checkout

git checkout HEAD~ files

