□□□-□□□

1-□□□□□□□□□□ □□

□□□□□□□□

□□□□. □□□□□□□□ □□□□□□□□□

□□ □□□ □□ □□□□□?

Quando un progetto software diventa complesso, anche la sua gestione è complicata.

Il controllo revisioni permette di gestire tutte le modifiche apportate ai documenti del progetto, archiviando tutte le versioni del progetto sin dal suo inizio

È uno strumento essenziale nello sviluppo moderno, e non usarlo può solo portare a conseguenze spiacevoli ...

No VCS?

□□□ □□□ □□□!

1.

□□ □□□□□□□□□

□□□□□□□ □□ □□□ □□□□□□□ □□□□□□□

□□□□□□□□□□□□ □□ □□□□ □□□□□□

- □□□□□□□□□□ □□□□□□□□□□□□ □□ □□□□□□□□□ □□□□ □□□□□□□□ □□□□□□□□□□ □□□□□□ □□□□□.
- □□□□□□□□□□ □□□□□□□□□□ □□ □□□□□□□□□□□□ □□□□ □□ □□□□□□□□□ □□ □□□□ □□□□□□□□ □□□□□.

□□□□□

```
#□□□□□□□□<□□□□□.□>
□□□ □□□□□()
{
    □□□ □;
}
```

□□□□.□

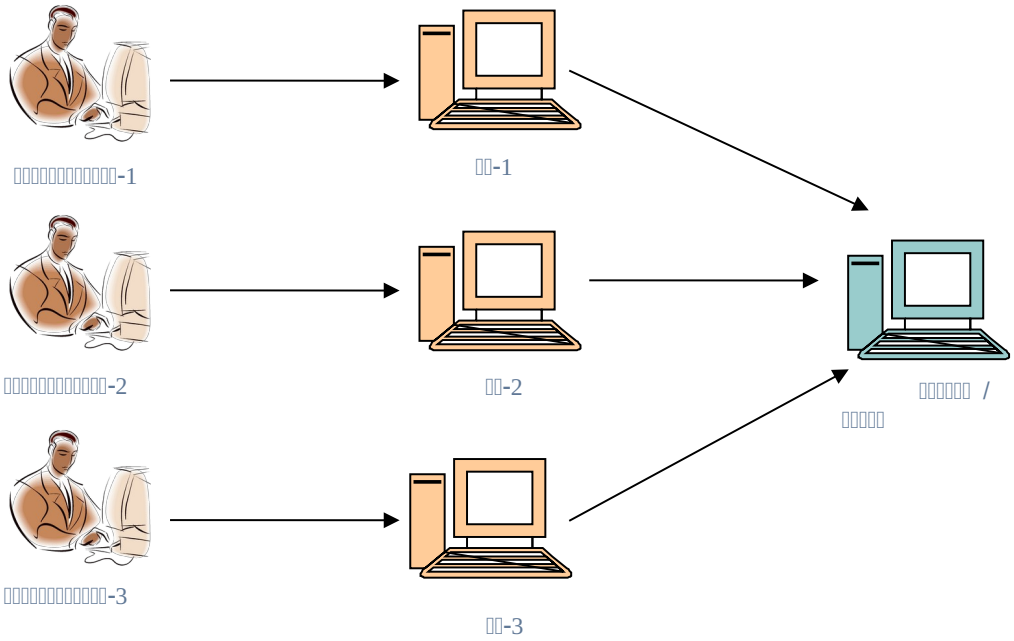
□□□□□

```
#□□□□□□□□<□□□□□.□>
□□□ □□□□□()
{
    □□□ □;
    □□□ □;
}
```

□□□□.□

□rog□□mmi□g □□ the □arg□/ma□y

- Non c'è modo di recuperare le versioni precedenti
- Non c'è modo di vedere le differenze tra le versioni.
- Processo di fusione versioni manuale, lento e di dubbio successo.
- Sovrascritture accidentali



[illegible] [illegible] System

Detto anche Versioning, o Revision Control System o Source control management.

[illegible] [illegible] ([illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible] ([illegible]) [illegible] [illegible] ([illegible])) [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible], [illegible], [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

□□ □□□□□□□, vc□ ci dà ...

- Un posto dove memorizzare le varie versioni del codice (e tutti i documenti correlati), un backup multiplo e flessibile.
- Possibilità di “tornare indietro” in caso di errore e tracciare le modifiche
- Sviluppo parallelo facilitato, potendo lavorare su più versioni contemporaneamente

2.

□□□

“You stupid git!”

RCS

RCS is created by Walter Tichy

1982

BITKEEPER + SVN

BitKeeper is created by Bitmover Inc and SVN is created by Collabnet Inc

2000

1972

SCCS

SCSS is created by Marc Rochkind

1986

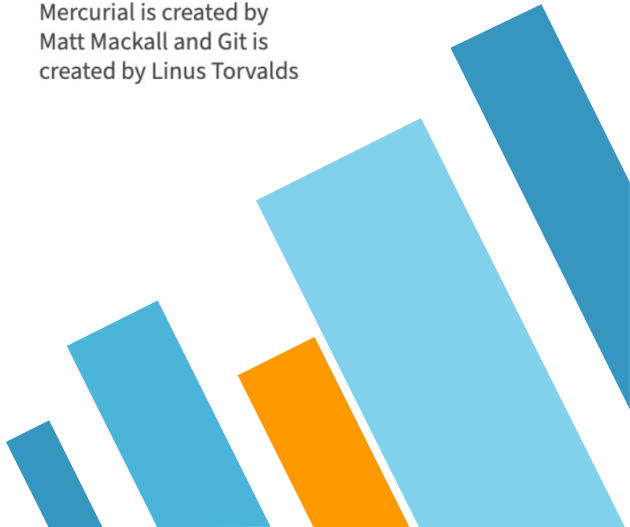
CVS

CVS is created by Dick Grune

2005

MERCURIAL + GIT

Mercurial is created by Matt Mackall and Git is created by Linus Torvalds



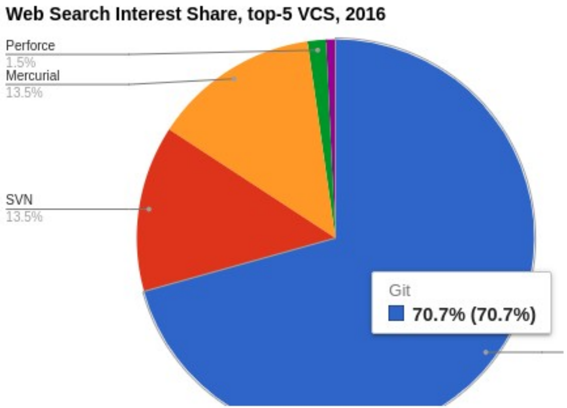
□□□□□□□□□□

□□□□□ & □□□ (‘82)

□□□ (‘86)

□□□□□□□□□□□ (2000)

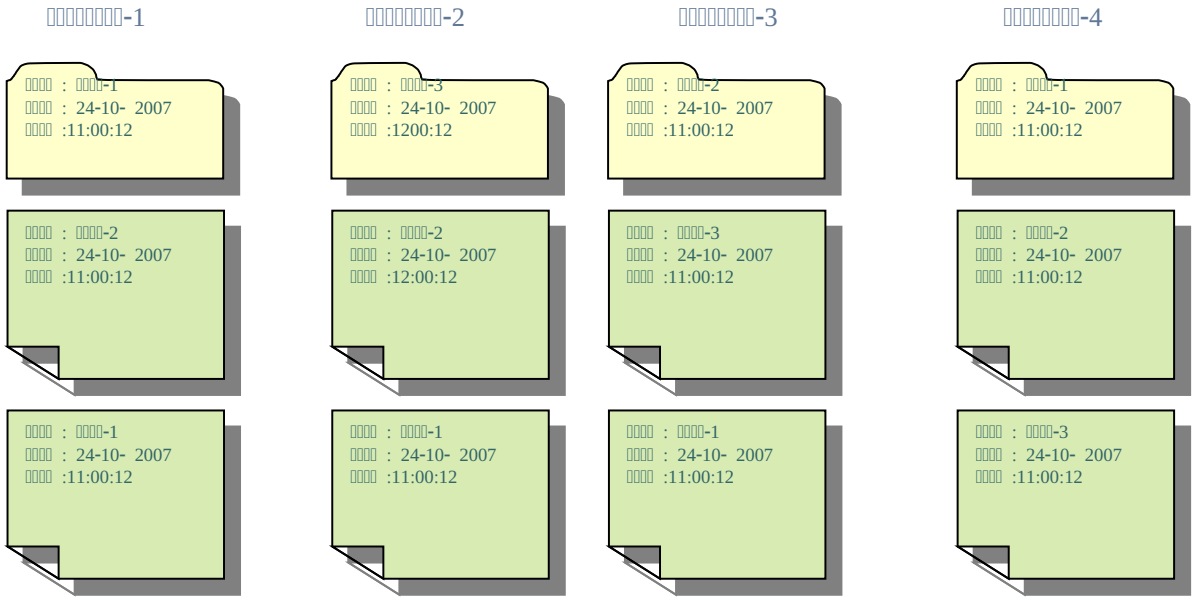
□□□ (2005)


Web Search Interest Share, top-5 VCS, 2016
Perforce
1.5%
Mercurial
13.5%
SVN
13.5%
Git
70.7% (70.7%)

“The problem with Git jokes？ Everybody has their own version”

[illegible]

Concetto □□□□: □□ repository

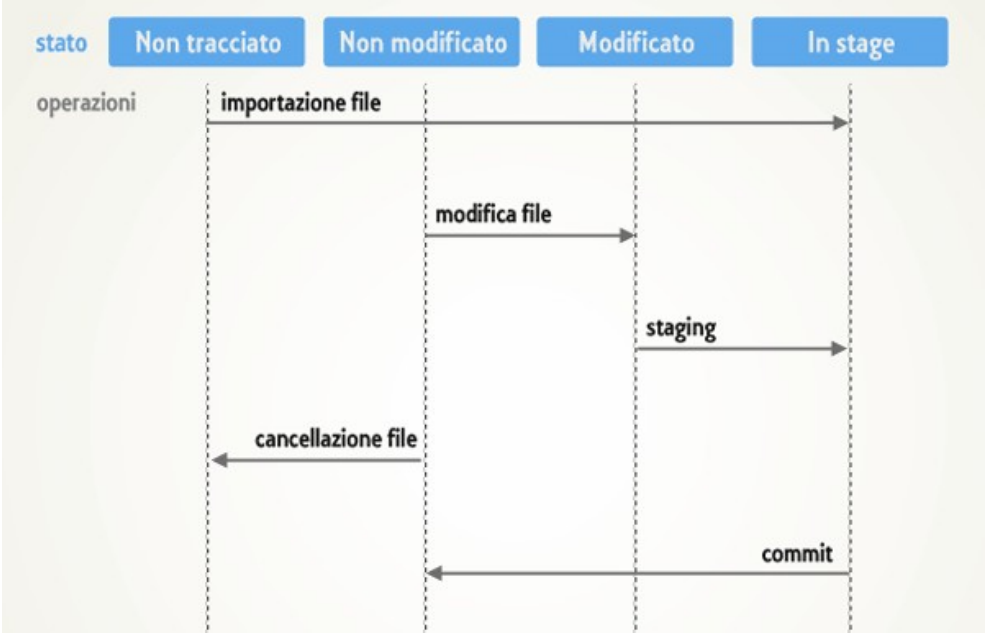


Versione-1	Versione-2	Versione-3	Versione-4
User : user-1 Date : 24-10- 2007 Time :11:00:12	User : user-3 Date : 24-10- 2007 Time :1200:12	User : user-2 Date : 24-10- 2007 Time :11:00:12	User : user-1 Date : 24-10- 2007 Time :11:00:12
User : user-2 Date : 24-10- 2007 Time :11:00:12	User : user-2 Date : 24-10- 2007 Time :12:00:12	User : user-3 Date : 24-10- 2007 Time :11:00:12	User : user-2 Date : 24-10- 2007 Time :11:00:12
User : user-1 Date : 24-10- 2007 Time :11:00:12	User : user-1 Date : 24-10- 2007 Time :11:00:12	User : user-1 Date : 24-10- 2007 Time :11:00:12	User : user-3 Date : 24-10- 2007 Time :11:00:12

[illegible] [illegible]

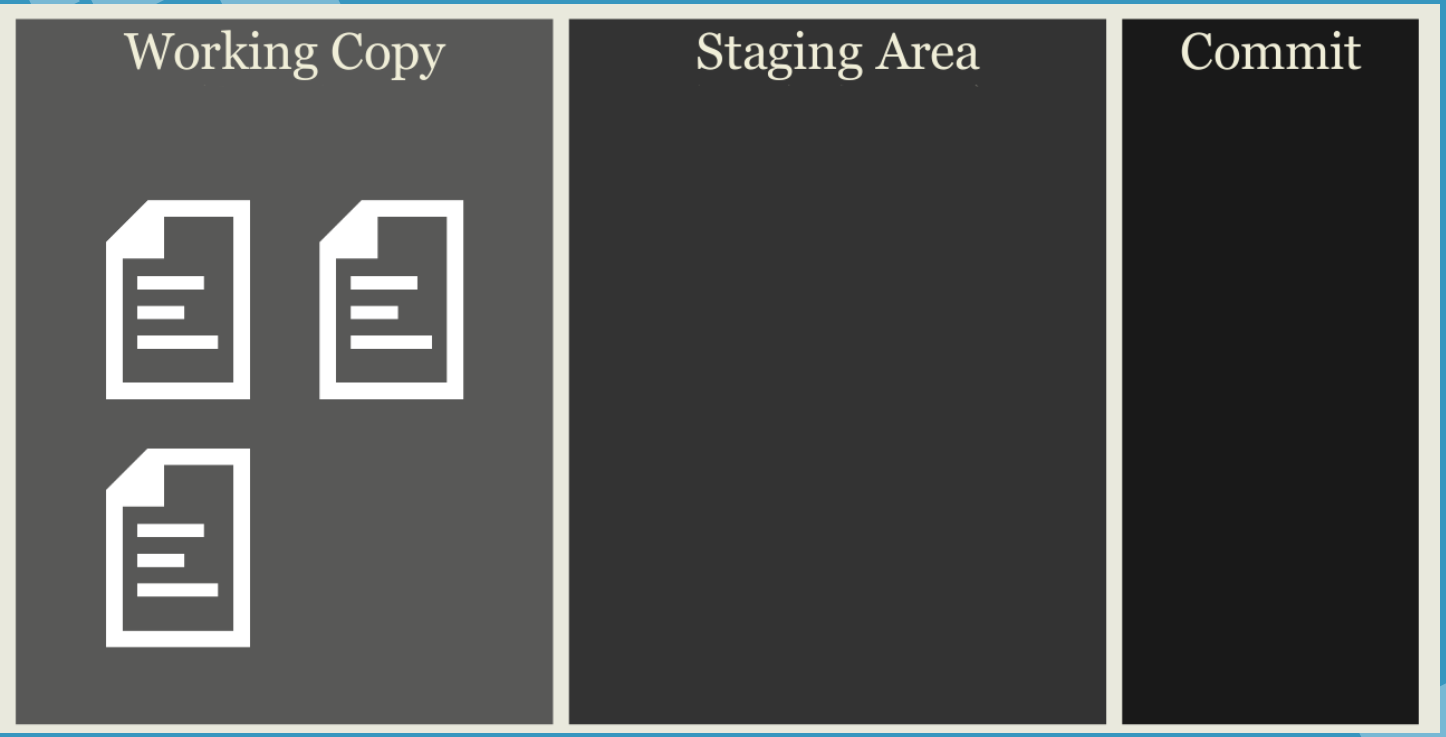
- Repository (deposito): un insieme di oggetti (file) e i riferimenti necessari - in pratica una “foto” del progetto. Può essere online o offline
- Clone: l'atto di duplicare localmente un repository
- Commit (lett. 'impegno'): Invio delle modifiche effettuate al repository
- Checkout: Ripristino di una particolare versione del progetto

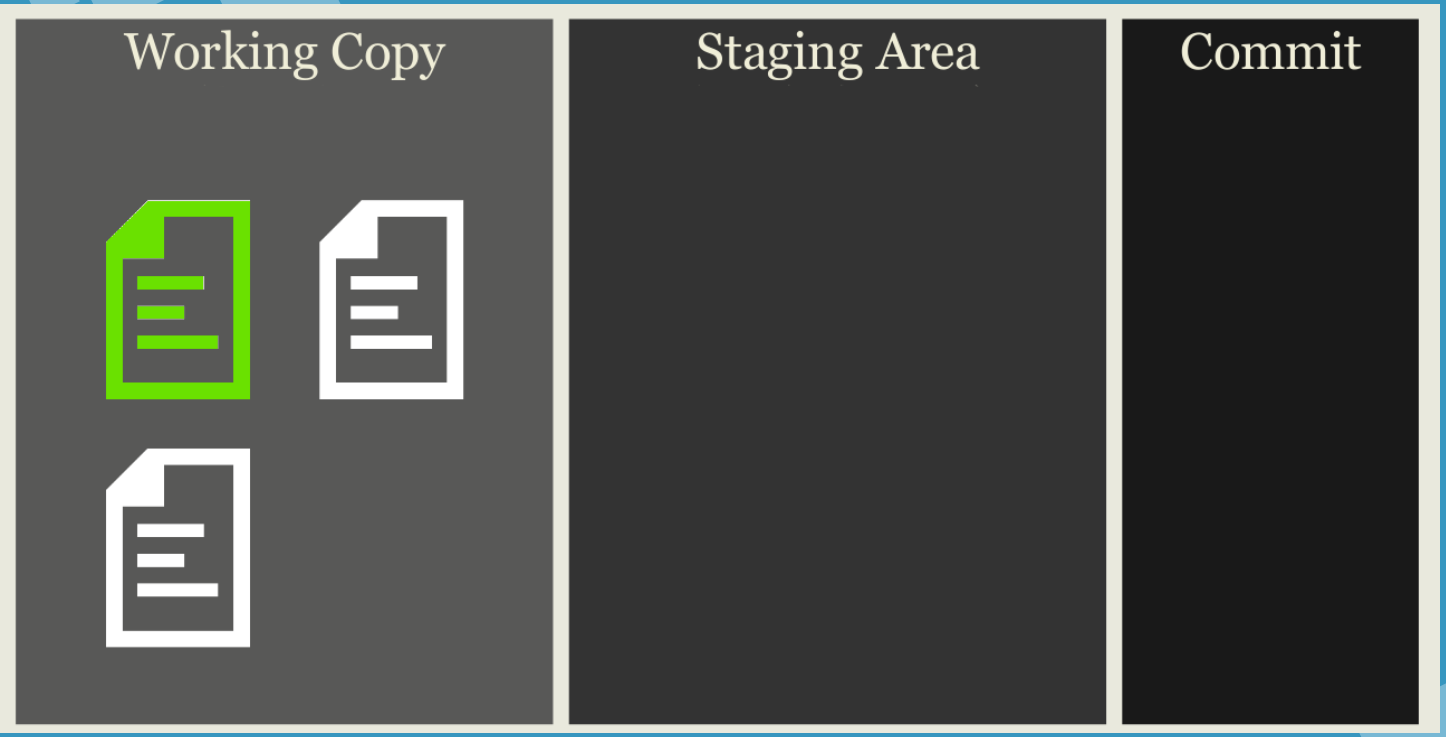
[illegible]

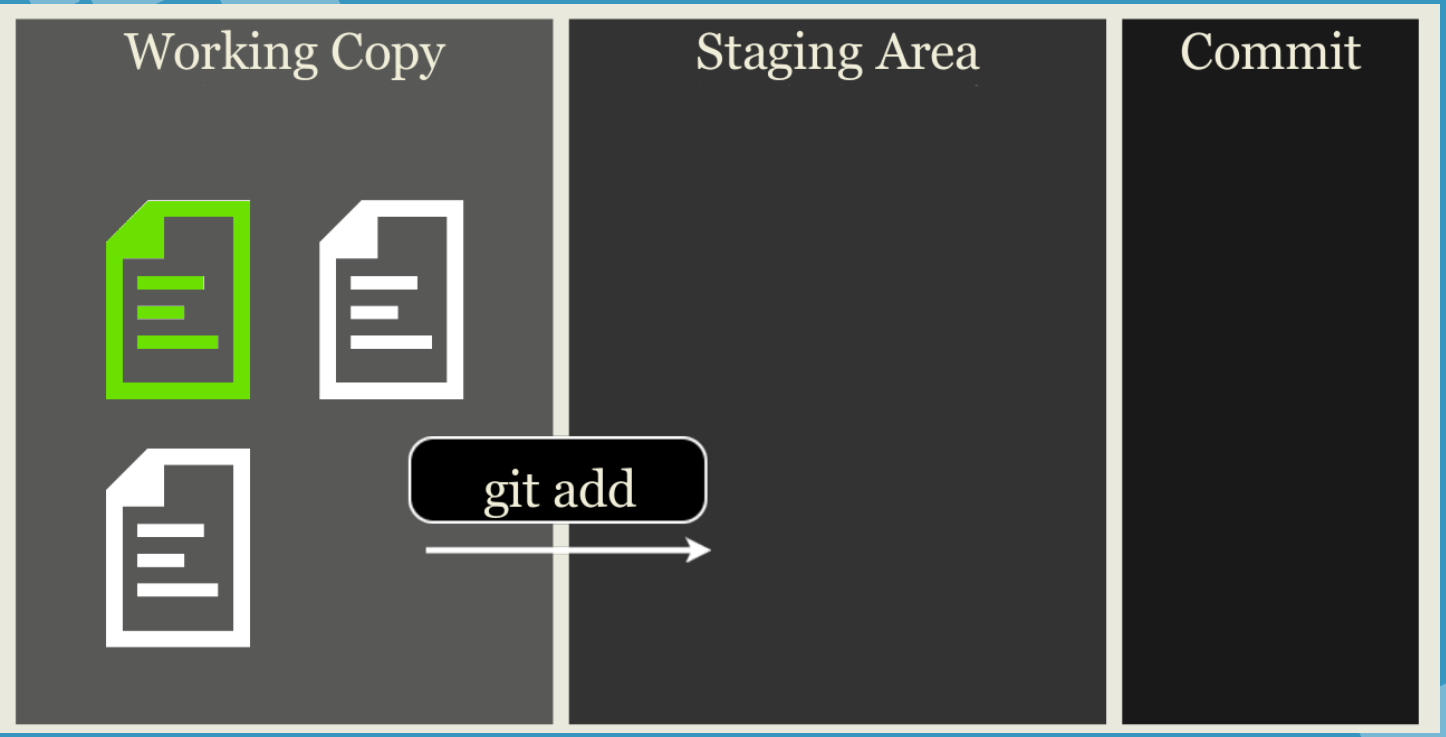

stato
Non tracciato
Non modificato
Modificato
In stage
operazioni
importazione file
modifica file
staging
cancellazione file
commit

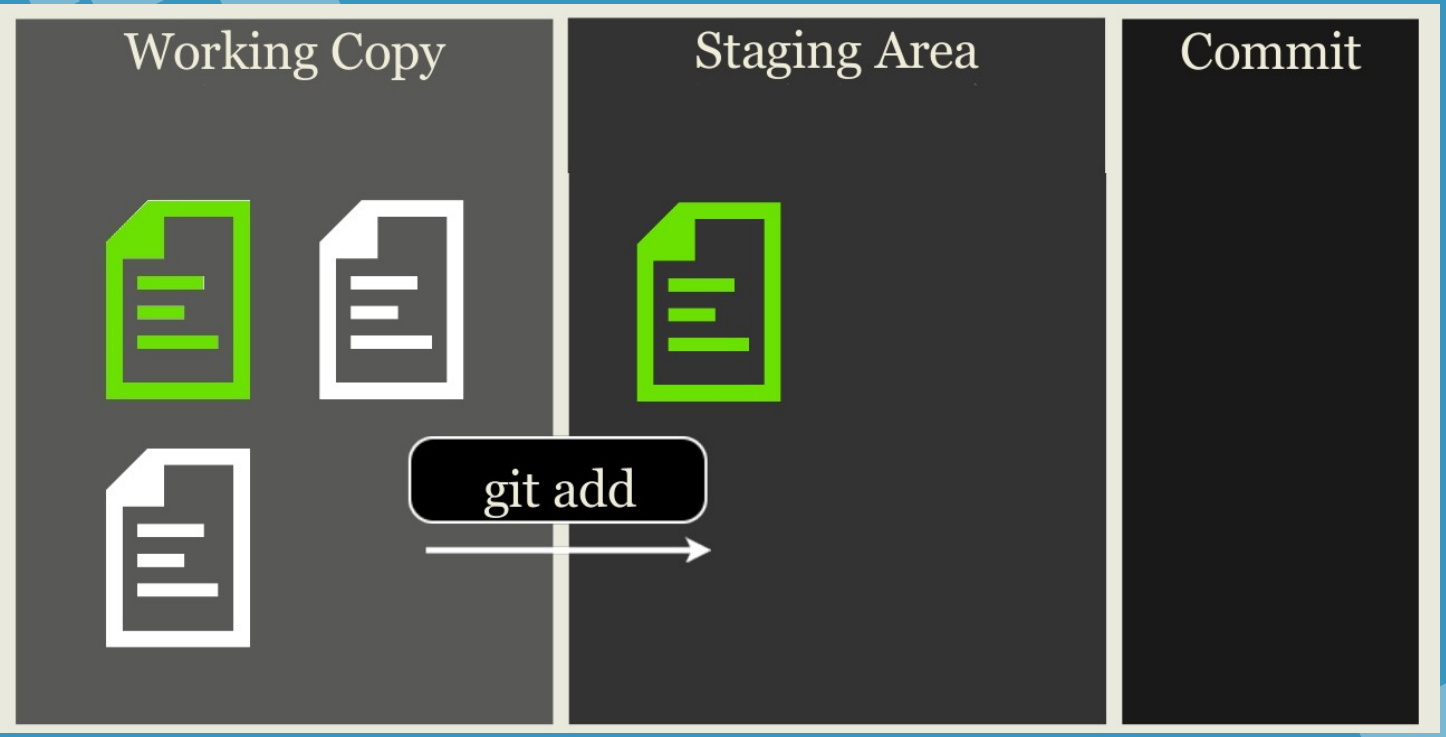
s□□u□nza di

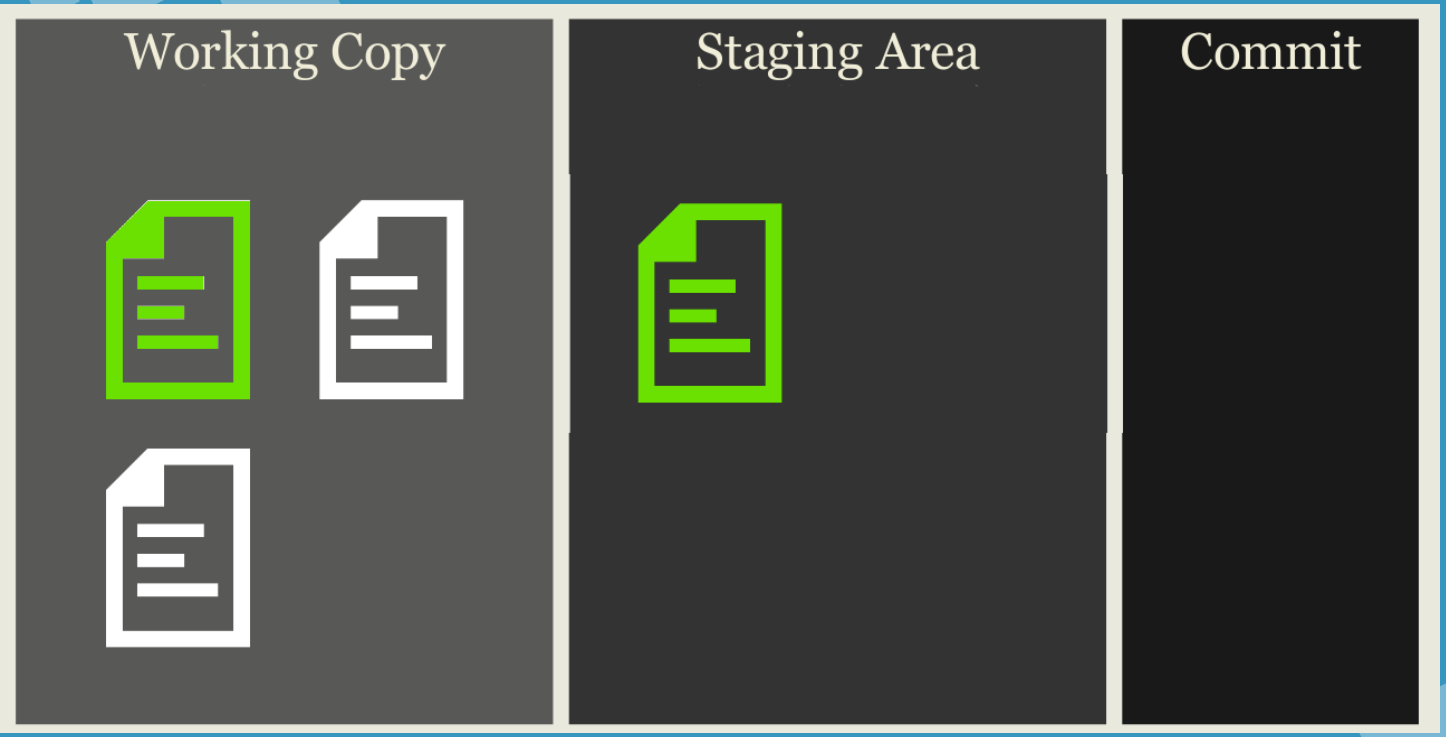
□□□□□□ tipic□

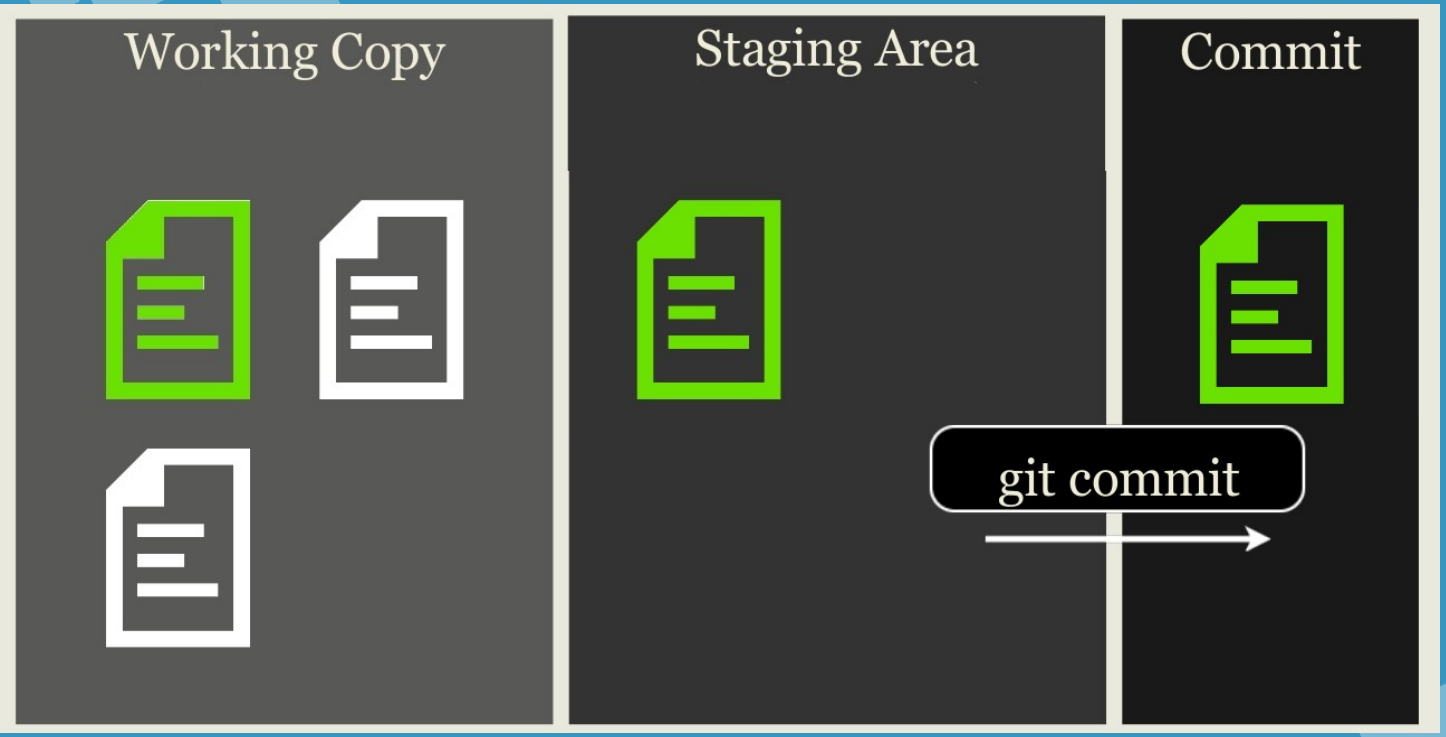

Working Copy
Staging Area
Commit

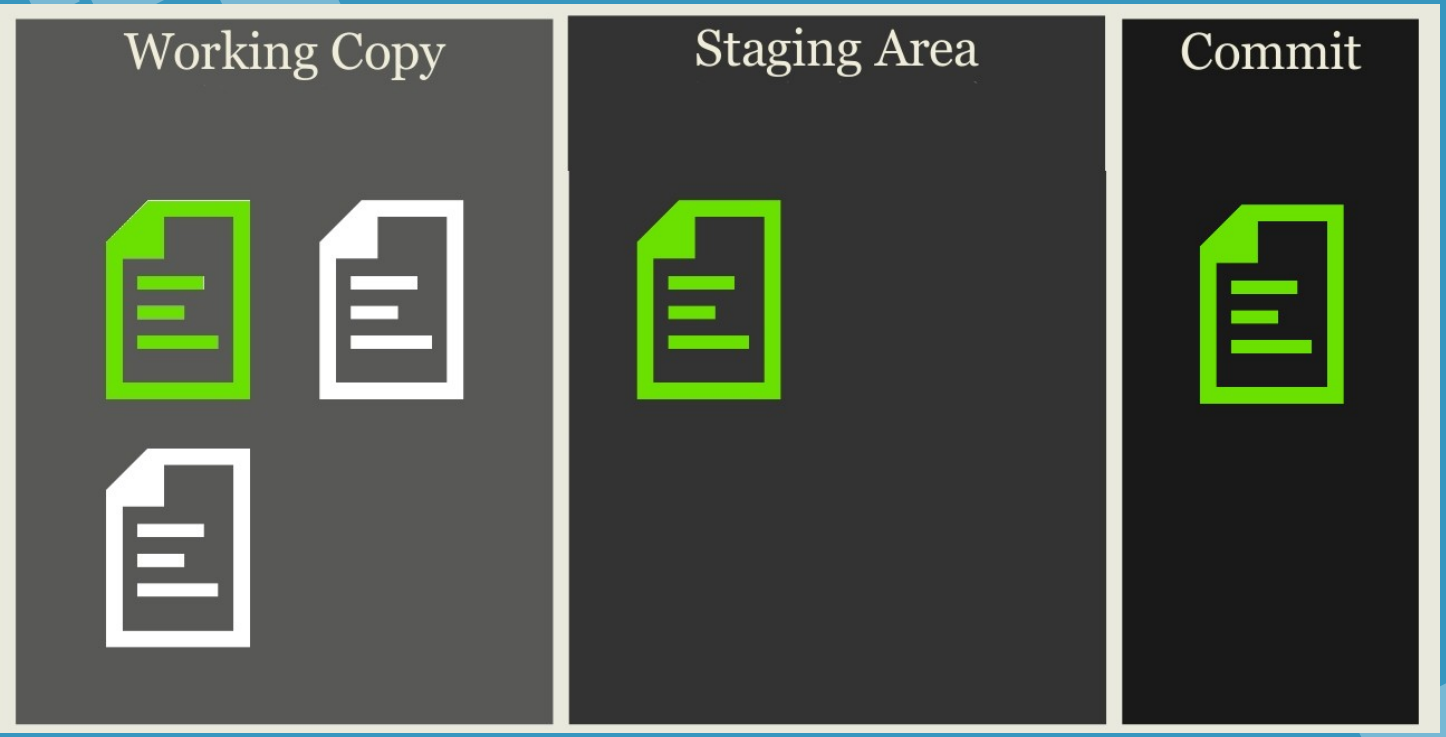

Working Copy
Staging Area
Commit

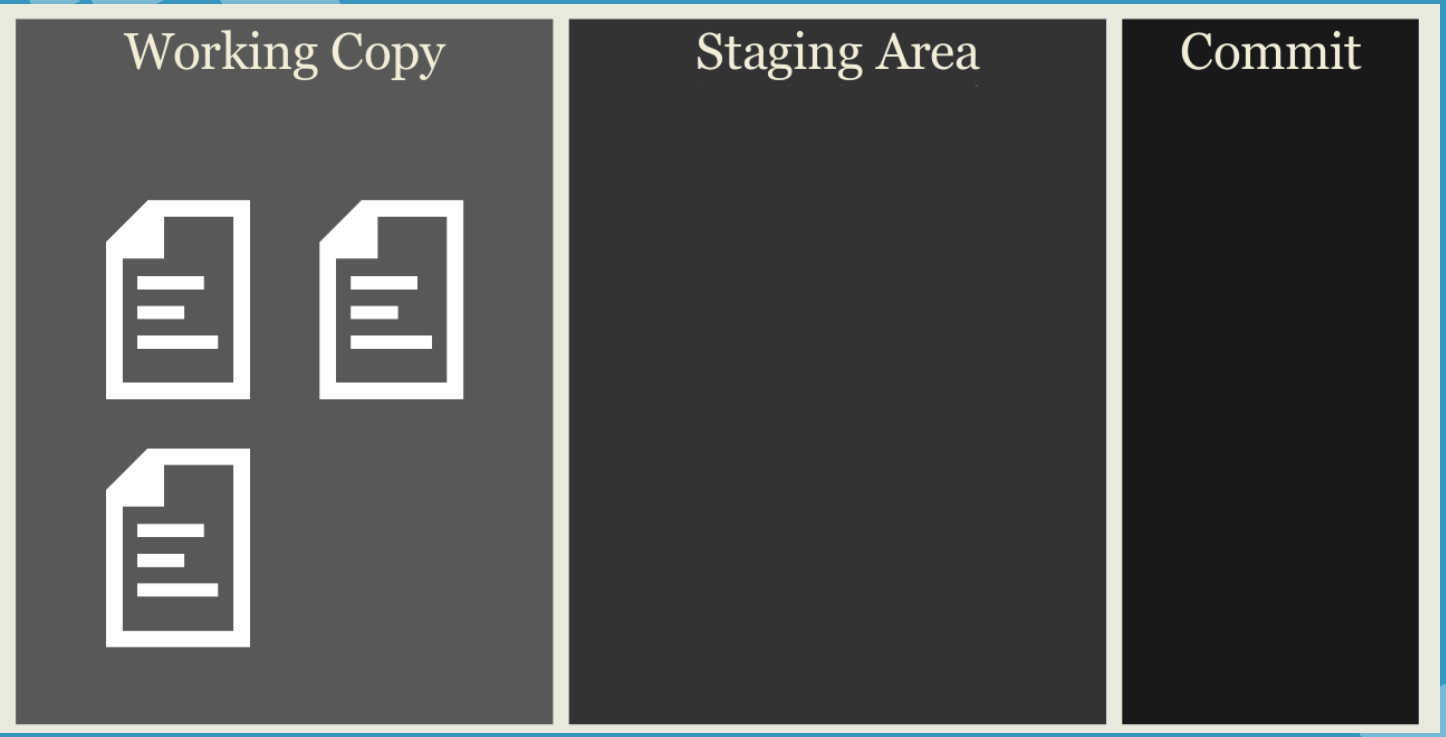

Working Copy
Staging Area
Commit
git add


Working Copy
Staging Area
Commit
git add

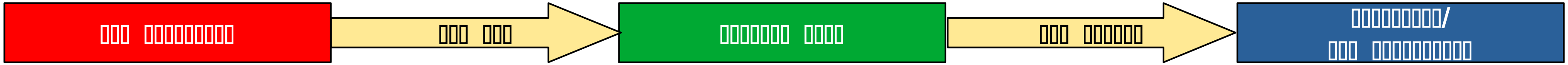

Working Copy
Staging Area
Commit


Working Copy
Staging Area
Commit
git commit


Working Copy
Staging Area
Commit


Working Copy
Staging Area
Commit

[illegible]



Quello che abbiamo visto è un esempio tipico del ciclo di lavoro base di git. Ora proveremo ad applicarlo davvero con il nostro computer.

3.

□□□□□□ □□□□□□

□□ □□□□□□□

L’arrivo di Internet 2.0, la necessità di condividere il codice con la comunità, il bisogno di accedere al codice sempre, comunque e da dovunque ha portato alla nascita dei primi siti di condivisione codice, come ad esempio Sourceforge (1999).

Alcuni di questi siti, grazie anche alla potenzialità di networking di Git, si sono evoluti in vere e proprie comunità di social networking

□□□□□□

GitHub

Fondato nel 2008, si tratta del sito sicuramente più famoso e con maggior numero di utenti. Fu comprato nel 2018 da Microsoft causando sconcerto e preoccupazione soprattutto nella comunità FLOSS.

Ciononostante, il sito è florido e ben congegnato. È tuttora la scelta principale per i progetti open source, grazie alla sua lunga storia e alle funzionalità di ricerca interna.

□□□□□□

Lanciato nel 2014 dalla due sviluppatori ucraini, il sistema è cresciuto considerevolmente negli anni e si distingue dai prodotti simili per essere open source ed essere ottimizzato per il modello di sviluppo DevOps.

È favorito dalle aziende con problemi di privacy, ed ha il maggior tasso di crescita negli ultimi anni.

□□□□□□□□□

Bitbucket

Il servizio è di proprietà di Atlassian, un grosso calibro nella gestione automatica dei progetti, noto in particolare per la suite Jira. Per questo motivo, è particolarmente diffuso nelle aziende medio-grandi.

Suo punto di forza è l’integrazione con altri servizi come JIRA Software, HipChat, Confluence e Bamboo.

Su che basi scegliere?

Github ha il miglior supporto della community, essendo il più anziano e pullula di progetti interessanti. Pende l'ombra di Microsoft.

Gitlab permette di lavorare installando un server privato e si concentra sulla catena di produzione

BitBucket è sicuramente la scelta più “corporate”.

□ □□□□□□?

Per i nostri interessi , tutte queste aziende offrono la possibilità di utilizzare gratuitamente i loro servizi , e regalano non meno di 500Gb per ogni repository. Le differenze s'iniziano a vedere a partire dai piani a pagamento. Qui faremo riferimento soprattutto a □□□□□□, per via del suo impegno a favore del software open source.

È molto probabile che li userete tutti e tre .

Git: a foolish or worthless [illegible]

"I'm an egotistical bastard, and I name all my products after myself. First Linux, then Git."

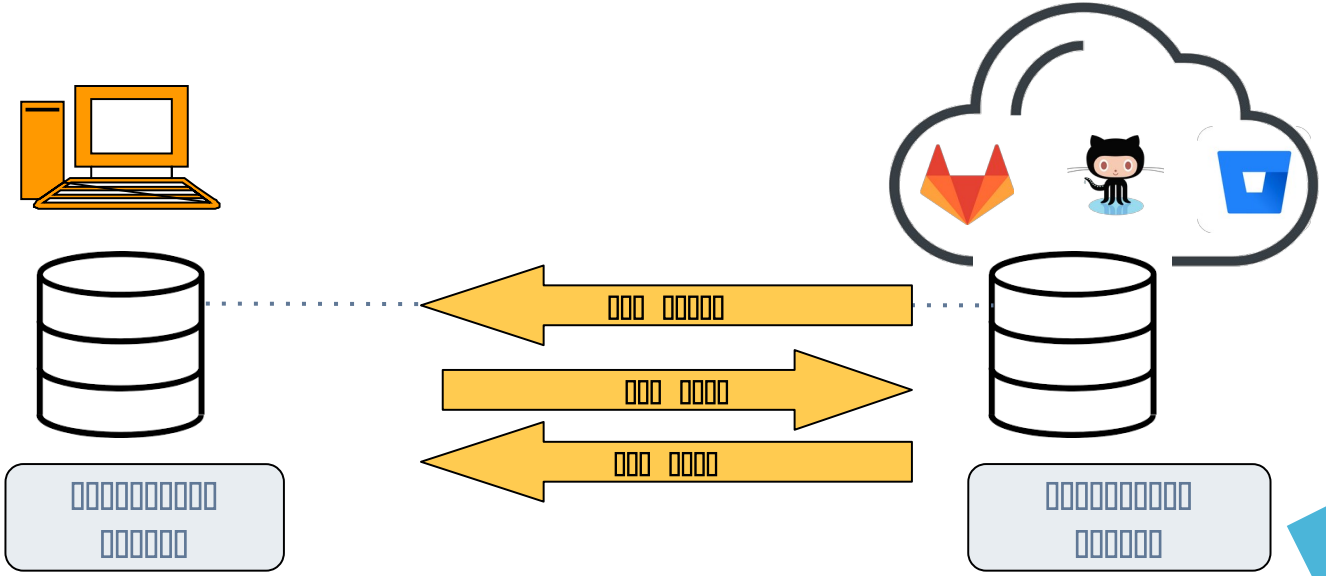
[illegible]

4.

□□□ □□□□□□□

□□□□□□ □□□□□ □□□□□□

[illegible] [illegible] [illegible]



git clone

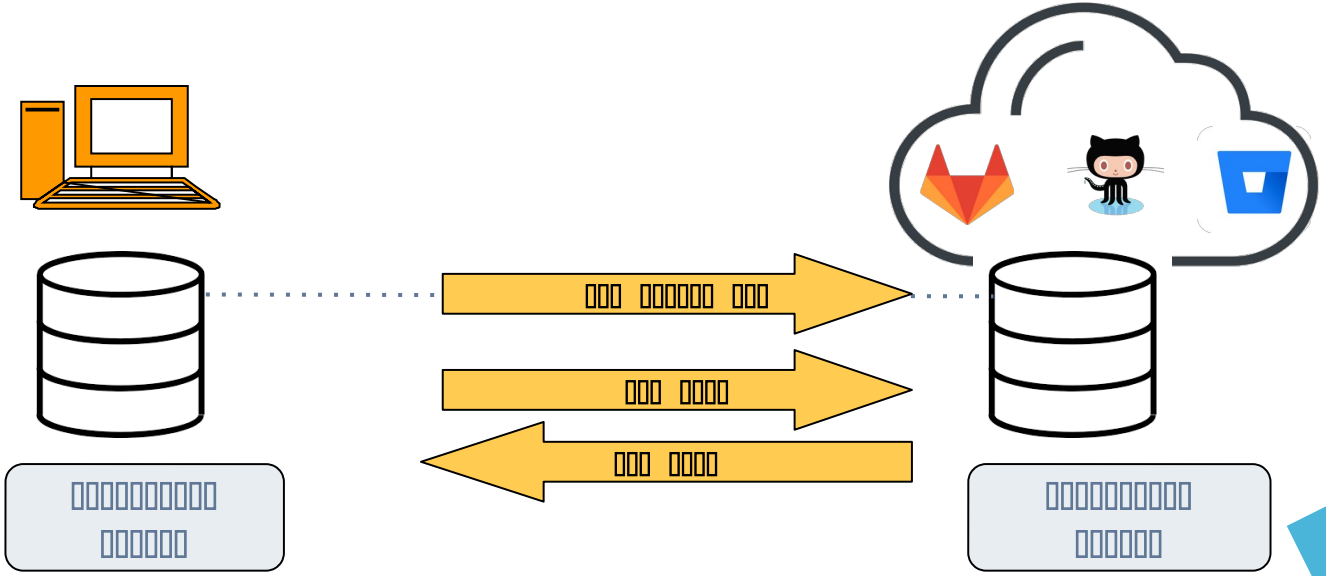
git push

git pull

Repository
locale

Repository
remoto

[illegible]



git remote add

git push

git pull

Repository
locale

Repository
remoto

5.

□□□□□□

[illegible]

Finora tutte le modifiche confluiscono in un unico prodotto finale, e ogni volta può essere necessario risolvere dei conflitti - magari inutilmente se il lavoro non è definitivo.

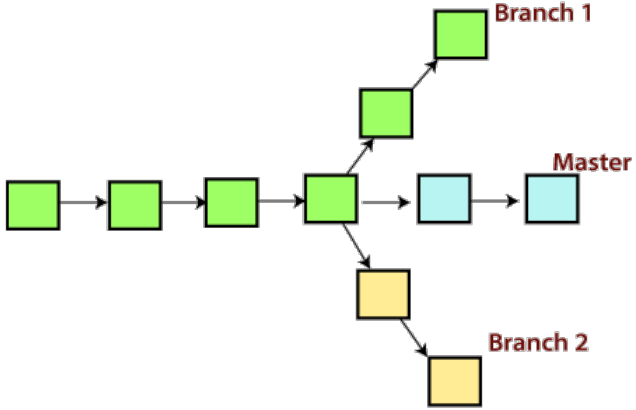
Si può però lavorare su versioni separate dei file per evitare di "sporcare" il lavoro degli altri (o proprio). In pratica si ha la libertà si lavorare, provare, "sporcare" il codice senza compromettere lavori di altri e/o defininivi.

□ □□□□□□

Per questo si usano i branch， una diramazione del codice； i commit relativi a questa diramazione resteranno del tutto isolati rispetto alla versione principale ， quella chiamata “main” o “master”， dandoci totale libertà di lavoro． Quando si è pronti， si può la diramazione al tronco principale， e solo in quel momento ci dovremo occupare degli eventuali conflitti ．

[illegible]

[illegible] [illegible] [illegible] [illegible] [illegible], [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible]. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] (master [illegible] [illegible]).



> “Git gets easier once you get the basic idea that branches are homeomorphic endofunctors mapping submanifolds of a Hilbert space.”
>
> *[illegible]*

5.

[illegible]

[illegible]

‘□ □□□□□□□□□□?

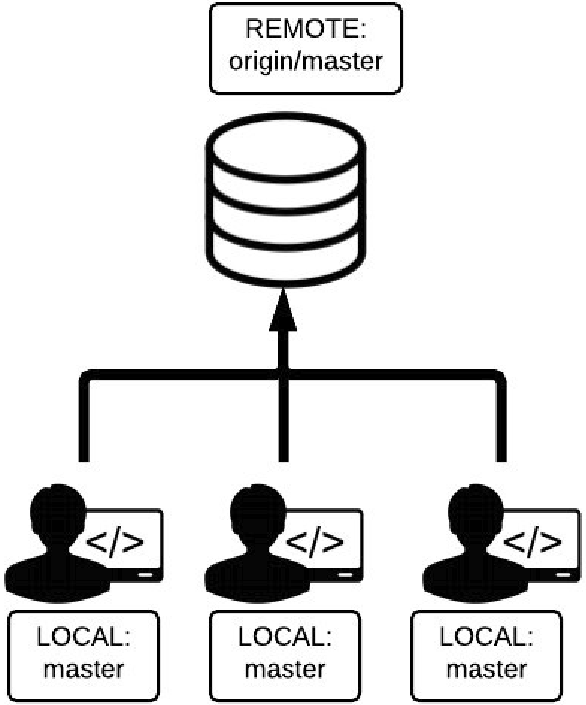
□□□□□□□□□ □□□□□ □□□□□□ □ □□□□□□□□□□□□ □□□□ □□ □□□□□□□□. □□□ □ □□□□□□□□□□□□□□□□ □□□□□□□□□□□ □ □□ □□□□□□□ □□□□ □□□□□ □□ □□□□□□□ □□ □□□□□□□□ (□ □□□ □□□□□□□□□□). □□□□□□□□□□ □□□□□□□.


N'CHE
SENSO ??
memegen.it

Centralized Workflow

Il server remoto (su Github) è il server di riferimento ("custode della verità").
Ogni sviluppatore clona il repository e sviluppa localmente, committando quando necessario.
Quando uno sviluppatore termina una sezione significativa, pulla e integra localmente le eventuali modifiche. Quindi reinvia al repo remoto la versione aggiornata.

É il sistema visto finora.



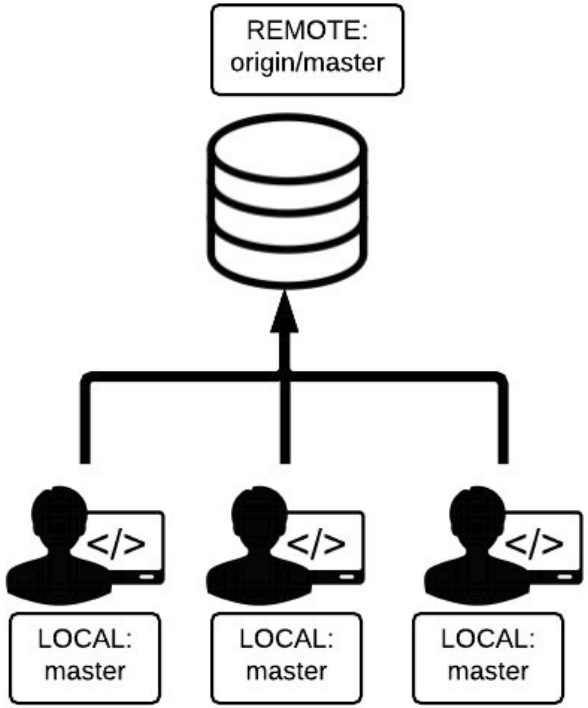
[illegible]

[illegible]

- Facile da capire
- Ottimo per team piccoli

[illegible]

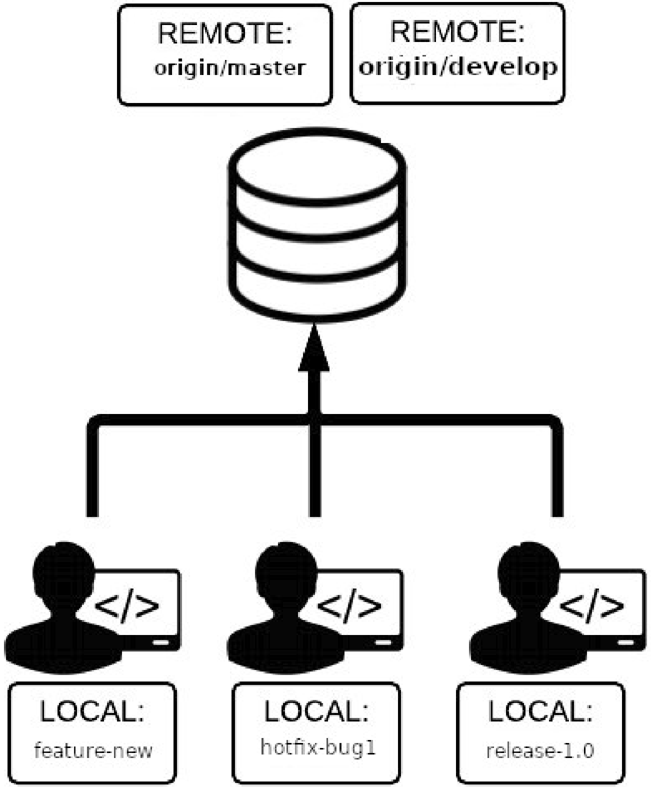
- Alla lunga gestire i conflitti diventa problematico
- Non sfrutta la natura distribuita di Git



□□□□□□□□

È un flusso molto famoso proposto da Vincent Driessen nel 2010. Prevede due branch di durata infinita:

- master/main — contiene il codice pronto (in gergo, "di produzione").
- develop — branch di sviluppo

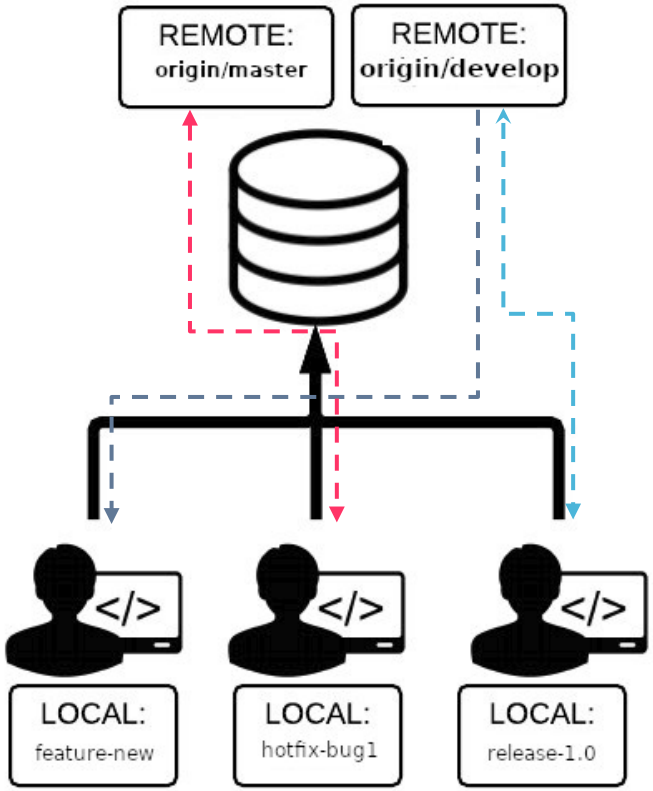


□□□□□□□ (2)

Ogni sviluppatore crea branch temporanei secondo una terminologia ben precisa:

- feature-* — per sviluppare nuove funzionalità. Figlia di develop.
- hotfix-* — per riparare ad errori del master. Da mergiare in master o develop
- release-* — in preparazione di un rilascio. Figlia di develop, da riunire a master e develop.

. Ulteriori fettagli qui.



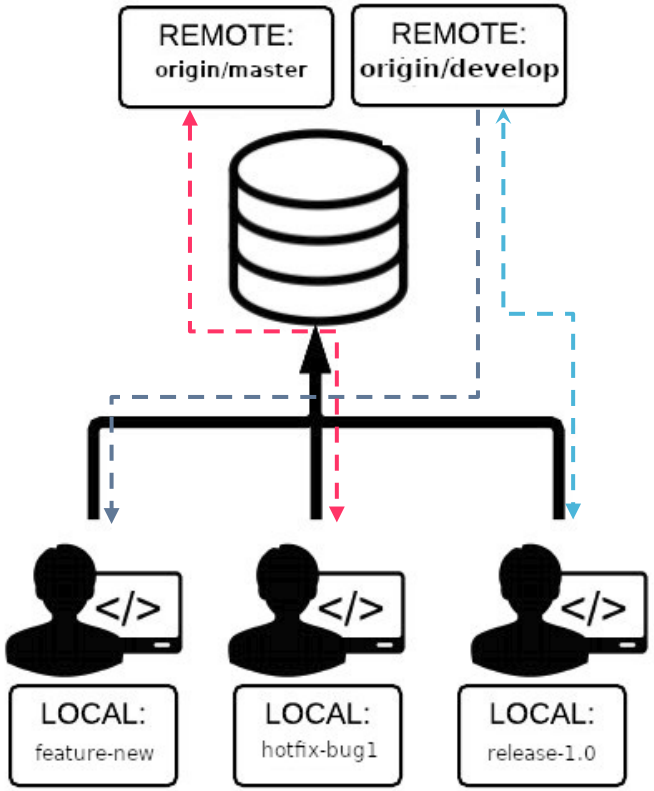
[illegible]

[illegible]

- Ideale se occorrono rilasci regolari.
- Presenza di tool
- Master ha solo codice stabile..

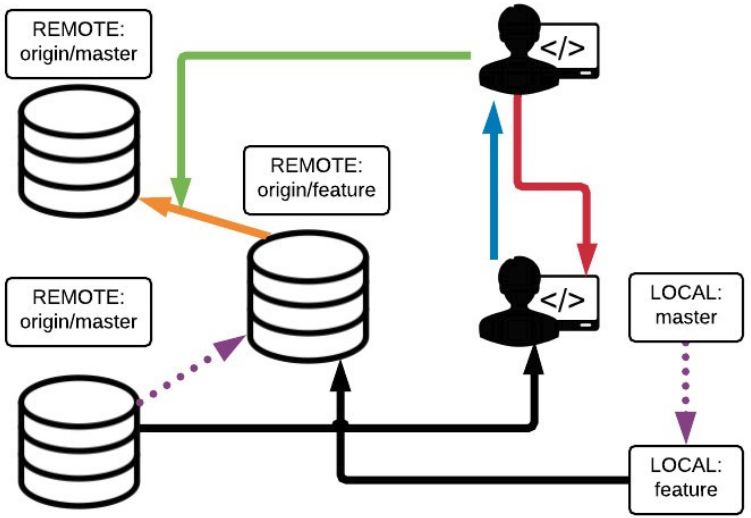
[illegible]

- La storia passata diventa intricata e illeggibile
- Non funziona bene con versioni multiple
- Benvenuti a Merge Hell. Popolazione: 1 (tu).



Удаленные ветки

Не всегда ветки в Git остаются на компьютере. Иногда приходится работать на нескольких, поэтому ветки приходится отправлять; но в какой момент отправлять не ветки нужно решать. Для этого существуют удаленные ветки (специальные ветки для работы с удаленными репозиториями). Удаленные ветки не являются полноценными ветками на компьютере и нам нужно (например чтобы с ними работать), они являются лишь ссылками.



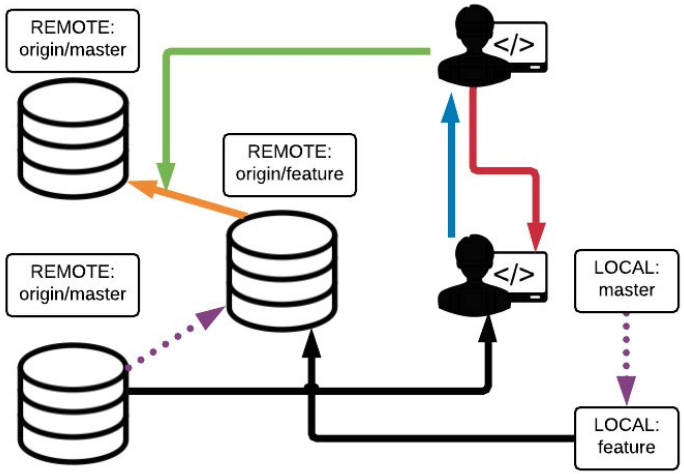
[illegible]

[illegible]

- [illegible]
- [illegible]

[illegible]

- [illegible]


REMOTE:
origin/master
REMOTE:
origin/feature
REMOTE:
origin/master
LOCAL:
master
LOCAL:
feature

□□□□□□□ &
□□□□/□□□□□
□□□□□□□

□□□ □□□□□□□□?

Il forking è una caratteristica presente in tutti i siti di social coding che permette di duplicare un repository esistente, inserendolo tra i propri progetti (quindi con libertà totale di accesso) pur mantenendo un legame con il repository originario (nome in codice "upstream")

□ □□□□□□□?

Questa relazione è alla base del cosiddetto sistema di □□□□ □□□□□□□□ （Github， BitBucket）/m□r□e re□u□st （Gitlab）.

Dopo aver forkato un repository e aver realizzato delle modifiche， si invia una richiesta di inserire la modifica invece di farlo direttamente． Questi potrà accettare la modifica oppure rifiutarla， inviando magari un commento．

Рабочий процесс

Рабочий процесс (workflow) это последовательность

этапов выполнения задач. Рабочий

процесс описывает их порядок

Рабочий процесс. Рабочий процесс,

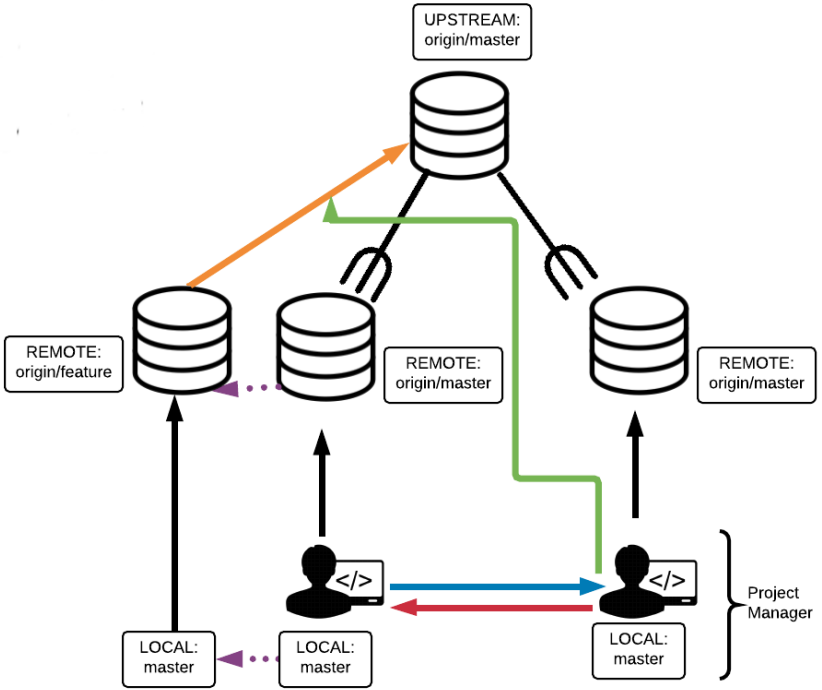
задает этап выполнения и их названия

(названия веток)

Он задает их роли, их названия

названия их назначения и их работу их

названия.



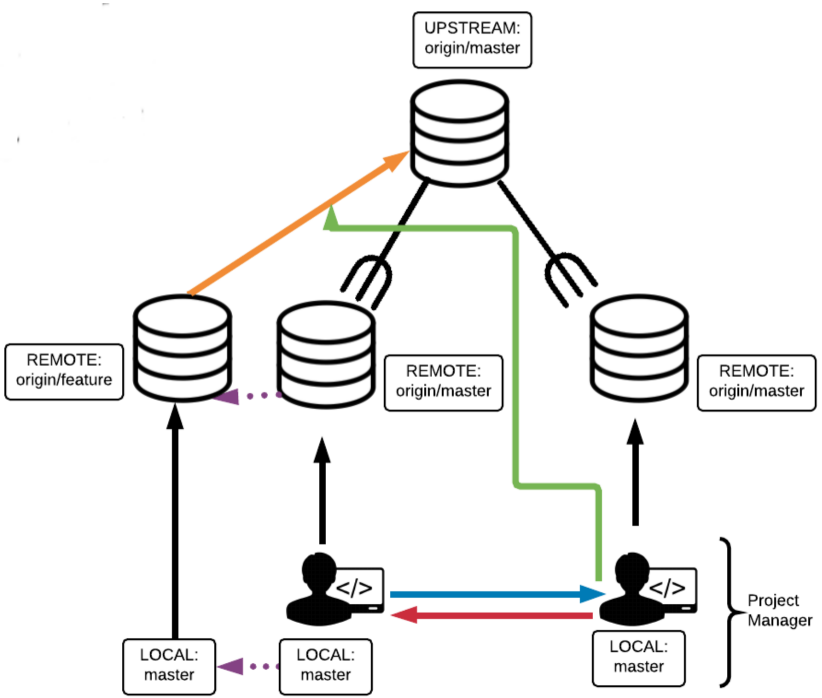
[illegible]

[illegible]

- Il repo upstream resta impeccabile, tramite un ferreo controllo degli accessi in scrittura .

[illegible]

- Più complicata
- Implica una struttura gerarchica



□□□□□□!

Ora □□ va s□l pra□ico

[illegible]

Ringraziamenti a :

- Presentation template by SlidesCarnival
- Photographs by Unsplash
- Anil Gupta (www.guptaanil.com)
- Pete Nicholls (github.com/Aupajo)
- Armando Fox

[illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible]ea[illegible]iveComm[illegible] [illegible]-[illegible] 3.0