

Introduzione alla data science e al pensiero computazionale

Lezione 2: La produzione del software nel lavoro (condiviso); gitHub

Giancarlo Succi

Dipartimento di Informatica – Scienza e Ingegneria

Università di Bologna

`g.succi@unibo.it`



Piano della lezione

- La produzione del software
- git



Crediti per il materiale didattico

- Il materiale didattico relativo a **git** è basato sull'introduzione a **git** del **dott. Emanuele Olivetti**, che ha gentilmente concesso il suo uso
- La localizzazione dell'originale è https://github.com/emanuele/introduction_to_Git



La produzione del software

- Il software è un bene immateriale
- È un po' come la musica
- Produrlo insieme ad altre persone apparirebbe essere più semplice
 - in modo ingenuo si potrebbe pensare che basterebbe scambiarsi il codice
 - per questo in ogni caso il lavoro remoto e lo sviluppo distribuito su scala planetaria sono pratiche non solo comune, ma, alle volte, molto profittevoli
 - vi sono persino ditte virtuali di grande successo
- Allo stesso modo i musicisti potrebbero scambiarsi gli spartiti online



Il problema nella produzione del software (1/2)

- Avete mai provato a scambiarsi una traduzione di latino online o per telefono?
- Che cosa succede spesso?
 - Non ci si ricorda qual è la copia giusta
 - Se ci sono errori, spesso non si capisce chi lo ha commesso
 - Ci sono difficoltà a dare credito a chi ha fatto la mole principale di lavoro



Il problema nella produzione del software (2/2)

- Tutto questo, e ancora molto di più succede nel software
- E per noi per ora il testo dei lucidi è il software
- E vedete come è difficile recuperare la versione più recente e corretta, e qual è il rischio di sbagliare date, aula, indirizzi email, ecc



git

- Per questo si usano sistemi di sincronizzazione del lavoro condiviso, detti “sistemi di gestione delle versioni”
- Essi **NON** risolvono il problema di come sviluppare il software da soli o con altri
- Però danno un contributo **ESSENZIALE** al lavoro
- Per questo il loro uso sarà un aspetto centrale del corso



Outline

- Version Control: **git**.
- Scenario 1: **singolo** sviluppatore, repository **locale**.
- Scenario 2: **Team** di sviluppatori, repository **centralizzato e remoto**. Minimalistico.
- Branching.
- Scenario 3: Contribuire a un progetto software localizzato su **GitHub**.
- Extra: come organizzare un repository centralizzato e più.



Version Control: Nomi & Significato

Da Wikipedia in inglese

*“Il sistema di gestione delle revisioni, anche conosciuto come *version control*, *source control* o, in parte, *software configuration management* (SCM), è la modalità di gestione dei cambi a documenti, programmi e altre informazioni salvate come file su computer.”*

Acronimi in uso:

- VC
- SCM

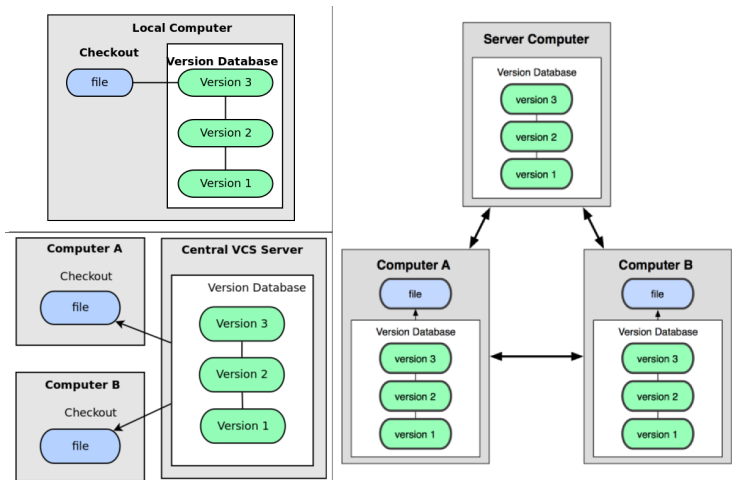
Errori di denominazione:

- ~~Versioning~~

D: Avete mai usato un sistema di VC?



VC: Locale, Centralizzato, Distribuito



Da *Pro Git*, S.Chacon 2009, CC 3.0 license.



Sondaggio: git


- D1: Avete mai sentito parlare di `git`?
- D2: Usate `git`?
- D3: Da dove viene il nome “`git`”? (dalla FAQ di `git`)
 - Una combinazione di tre lettere a caso che è pronunciabile.
 - Acronimo (global information tracker).
 - Ironia.



git? Perché “git”?

Linus Torvalds: *“Io assegno nomi a tutti i miei progetti da me, prima Linux e ora git.”*

<http://www.merriam-webster.com/dictionary/git>

git  *noun* \ˈɡɪt\

Definition of GIT

British : a foolish or worthless person

Examples of GIT

- That *git* of a brother of yours has ruined everything!
- <oh, don't be such a silly *git*, of course your mates want you around>

Origin of GIT

variant of *get*, term of abuse, from ²*get*

First Known Use: 1929

Related to GIT

Synonyms: *berk* [*British*], *booby*, *charlie* (*also* *charley*) [*British*], *cuckoo*, *ding-a-ling*, *dingbat*, *ding-dong*, *dipstick*, *doofus* [*slang*], *featherhead*, *fool* [*British*], *goose*, *half-wit*, *jackass*, *lunatic*, *mooncalf*, *nincompoop*, *ninny*, *ninnyhammer*, *nit* [*chiefly British*], *nitwit*, *nut*, *nutcase*, *simp*, *simpleton*, *turkey*, *yo-yo*

Related Words: *daredevil*; *madman*, *madwoman*; *airhead*,



git(1/2)

Dalla pagina del manuale (in inglese):

git

usage: git [OPTIONS] COMMAND [ARGS]

The most commonly used git commands are:

add	Add file contents to the index
commit	Record changes to the repository
diff	Show changes between commits, commit and working
...	

git help <command>

git status



git(2/2)

Presentatevi a git:

```
git config --global user.name "Emanuele Olivetti"
```

```
git config --global user.email "olivetti@fbk.eu"
```



git– Singolo sviluppatore + repository locale

Scenario 1: singolo sviluppatore + repository locale



git – Singolo+Locale – Motivazioni

- Per caso usate un VC per il vostro repository locale?
- Perché usare un VC per un singolo sviluppatore con un repository locale?
 - Primo passo per un progetto condiviso.
 - Backup.
 - Per tracciare il proprio lavoro.



git – Singolo+Locale – Init

`git init`

- Crea un repository `git` vuoto.
- Crea una directory `git`: `.git/`

**working
directory**

**staging
area**

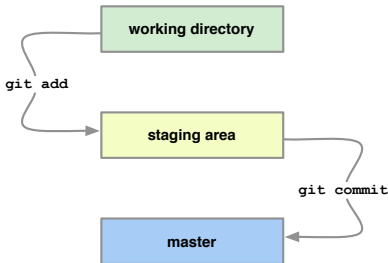
master

Nota: è una azione **sicura**. Non cambia i file esistenti.



git – Singolo+Locale – Il flusso dei dati

```
git add <filename>
```



```
git commit -m "Let us begin."
```

Wikipedia (tradotto dall'inglese)

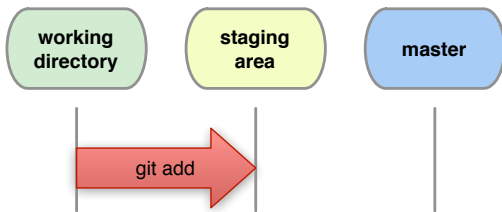
“La *staging area* è il luogo dove organismi, persone, veicoli, equipaggiamenti o materiali sono messi insieme prima dell'uso”.



git – Singolo+Locale – Add

```
git add file1 [file2 ...]
```

- Aggiunge nuovi file per il successivo **commit**.
- Aggiunge contenuto dalla directory di lavoro alla staging area per il successivo commit, in pratica indicizza
- Non aggiunge informazioni sui permessi dei file
- Di per sé non crea directory

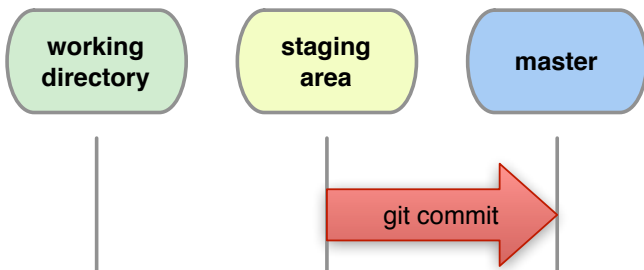




git – Singolo+Locale – Commit (1/2)

```
git commit [-m "Commit message."]
```

Trascrive tutti i cambiamenti dalla staging area al master.

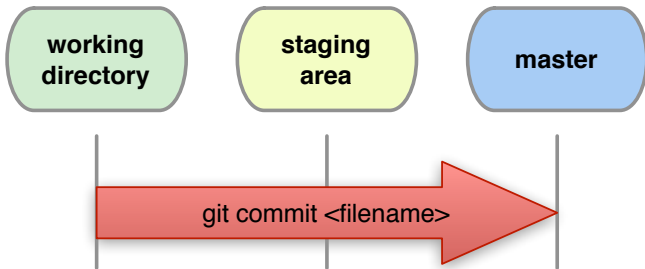




git – Singolo+Locale – Commit (2/2)

```
git commit file1 file2
```

Trascrive i cambiamenti dalla directory di lavoro e dalla staging area al master per file1, file2



```
git commit -a
```

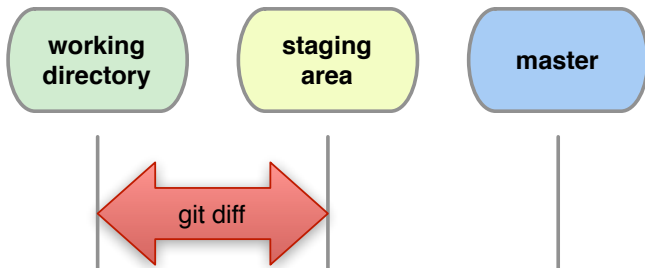
Attenzione! Trascrive tutti i cambiamenti nella directory di lavoro e



git – Singolo+Locale – Diff

`git diff`

Presenta quali cambiamenti siano present tra la *directory di lavoro* e la *staging area*; lavora sugli indici

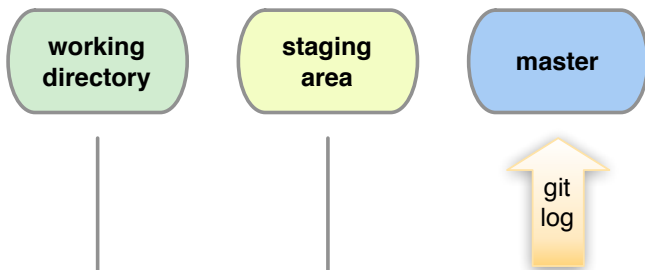




git – Singolo+Locale – Logs (1/2)

`git log`

Presenta i dettagli dei commit; ecco perché servono i messaggi

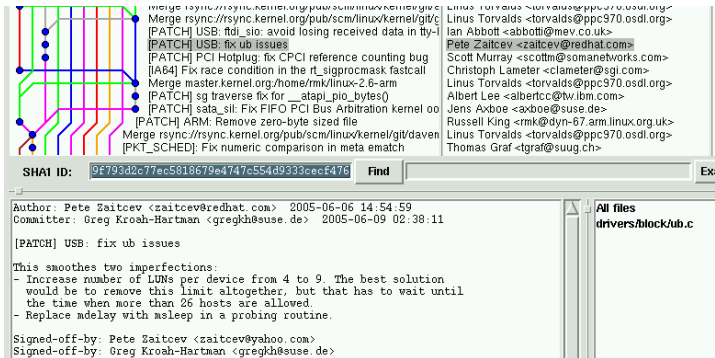




git – Singolo+Locale – Logs (2/2)

gitk

Interfaccia grafica per navigare in un repository git:



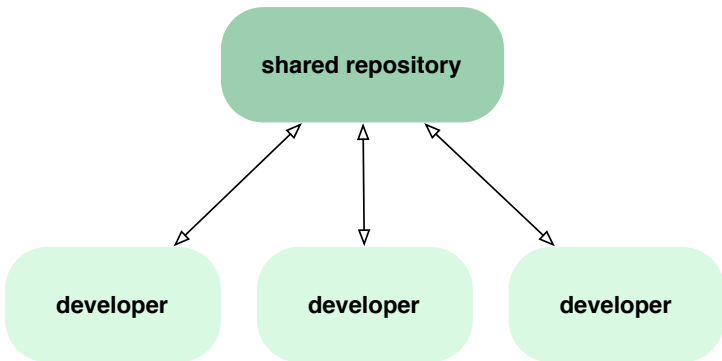


git – Team+Remoto/Condiviso (1/3)

Scenario 2: **Team** di sviluppatori, repository **centralizzato e remoto**. Minimalistico.

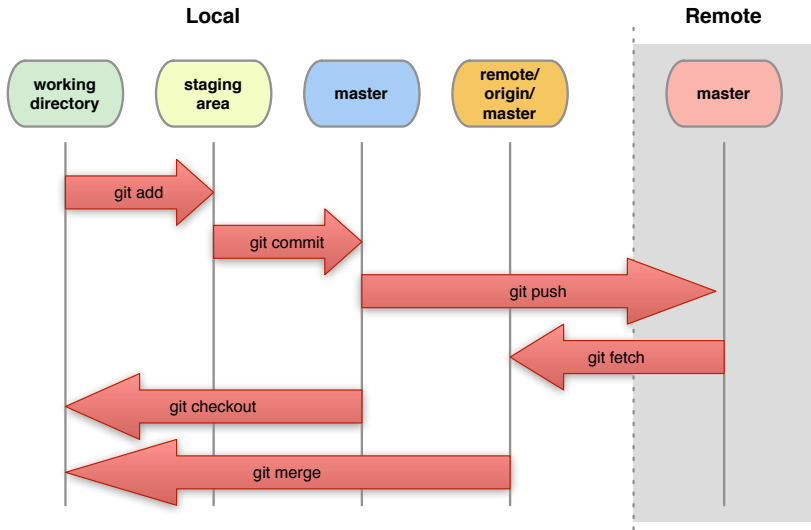


git – Team+Remoto/Condiviso (2/3)





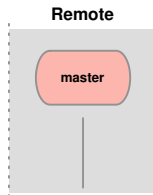
git – Team+Remoto/Condiviso (3/3)





git – Team+Remoto/Condiviso – Passi

```
git clone <URL>
```

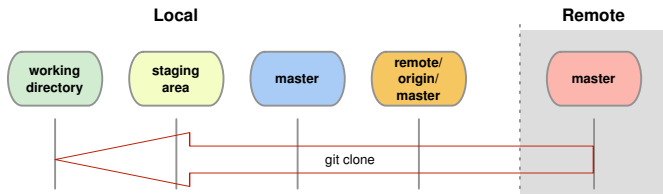




git – Team+Remoto/Condiviso – Passi

```
git clone <URL>
```

Crea *due* copie locali di *tutto* il repository remoto.



Protocolli di trasmissione dati disponibili:

- ssh://, git://, http://, https://, file://

Es.: `git clone https://github.com/ASPP/pelita.git`

```
git remote -v
```

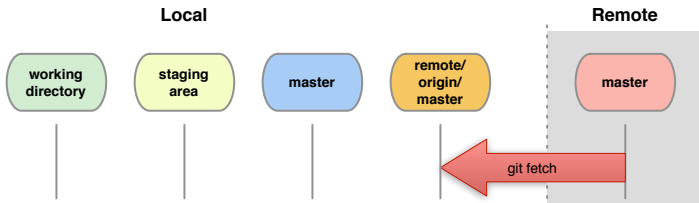
mostra **nome** e URL del repository remoto.



git – Team+Remoto/Condiviso – Fetch

`git fetch`

- Scarica gli aggiornamenti dal master remoto al master locale.
- Il master locale, la staging area e la directory di lavoro non cambiano.

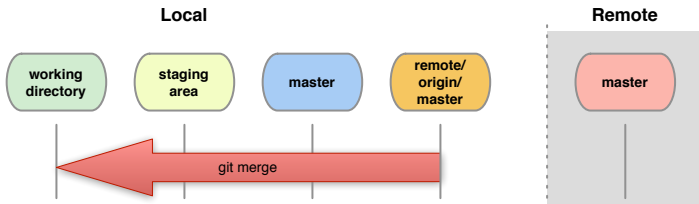




git – Team+Remoto/Condiviso – Merge

git merge

- Permette di riunire flussi di sviluppo diversi.
- **Attenzione:** può generare *conflitti*!
- **Nota:** La riunione (merge) avviene solo quando i cambiamenti sono stati sottoposti a commit.



`git fetch + git merge = git pull`



git – Team+Remoto/Condiviso – Conflitti (1/2)

Conflitti!

```
...
<<<<<< yours:sample.txt
Conflict resolution is hard;
let's go shopping.
=====
Git makes conflict resolution easy.
>>>>>> theirs:sample.txt
...
```




Risoluzione dei conflitti

- Bisogna innanzitutto vedere dove sono i conflitti:

```
git diff
```

- Poi occorre modificare le linee che provocano conflitto.
- Quindi si aggiungono le modifiche alla staging area:

```
git add file1 [...]
```

- E quindi si ripete un commit:

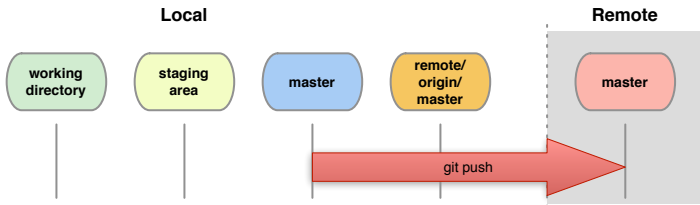
```
git commit -m "Conflicts solved."
```



git – Team+Remoto/Condiviso – Push

git push

- Modifica i *remote masters* (sia locale che remoto).
- Occorre prima di tutto fare un **fetch+merge**
- Solo dopo si può fare un push.



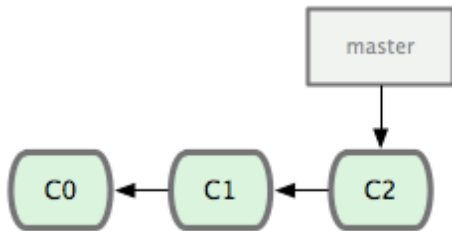


Branching

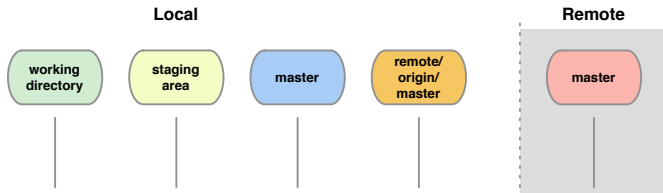
I fondamenti del branching, ovvero la suddivisione del prodotto per gestire configurazioni diverse



Dinamica del branching – partenza

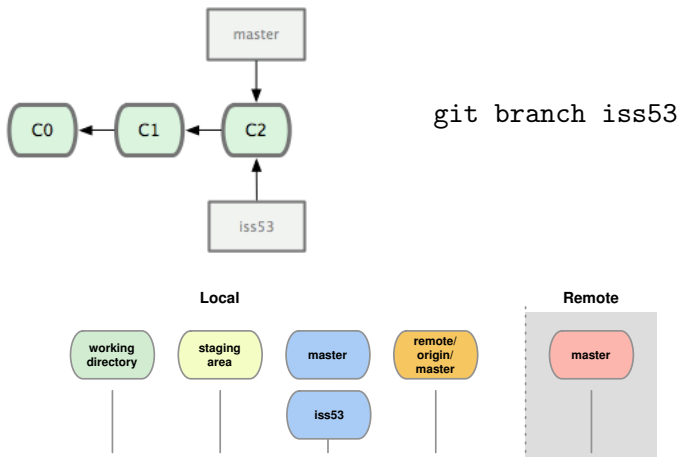


```
git commit (C0)
git commit (C1)
git commit (C2)
```



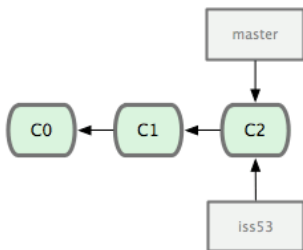


Dinamica del branching – evoluzione (1/3)

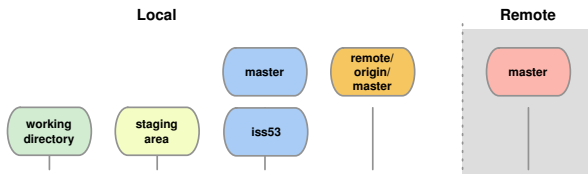




Dinamica del branching – evoluzione (2/3)

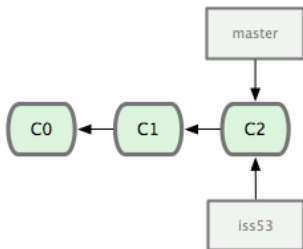


```
git branch iss53  
git checkout iss53
```

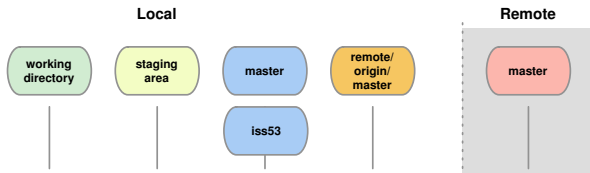




Dinamica del branching – evoluzione (3/3)

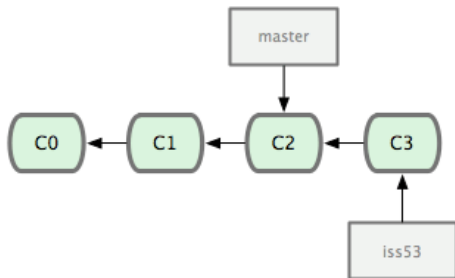


```
git branch iss53  
git checkout iss53  
git checkout master
```

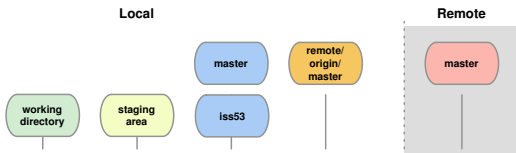




Dinamica del branching – nuovo commit

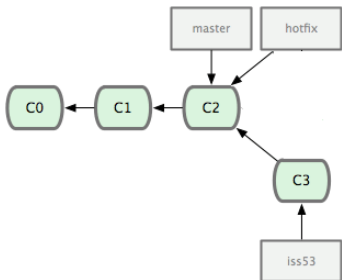


```
git checkout iss53  
git commit (C3)
```

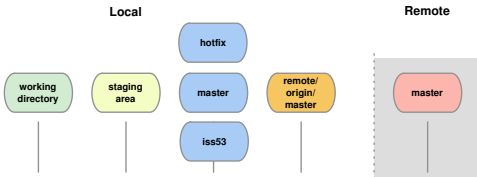




Dinamica del branching – fix (1/2)

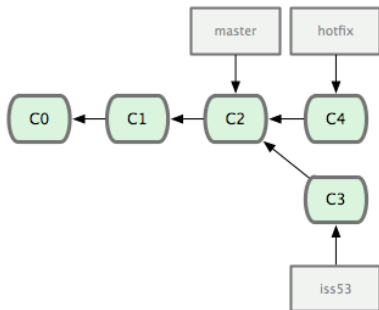


git branch hotfix

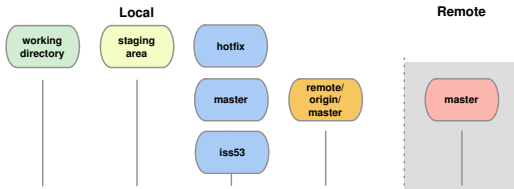




Dinamica del branching – fix (2/2)

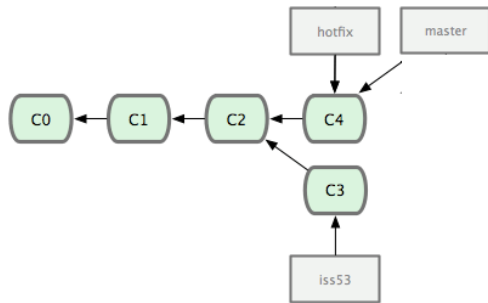


git checkout hotfix
git commit (C4)



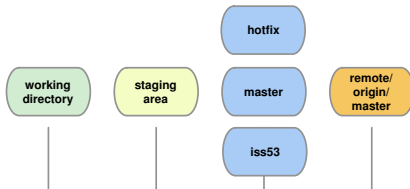


Dinamica del branching – ulteriori branch (1/3)

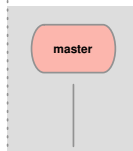


`git checkout master`
`git merge hotfix`

Local

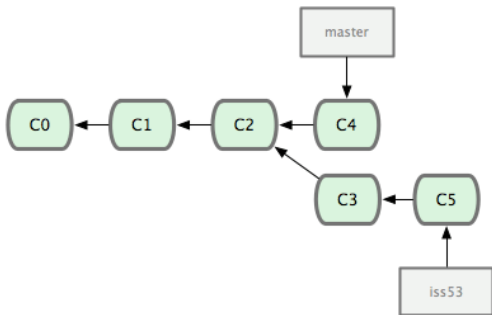


Remote

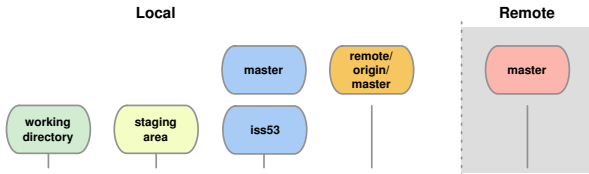




Dinamica del branching – ulteriori branch (2/3)

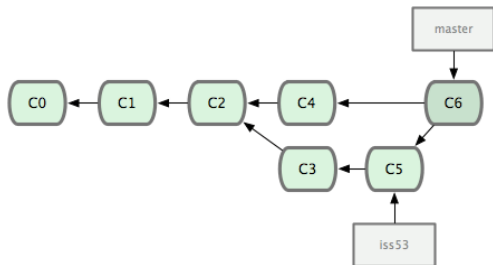


```
git checkout iss53  
git commit (C5)
```

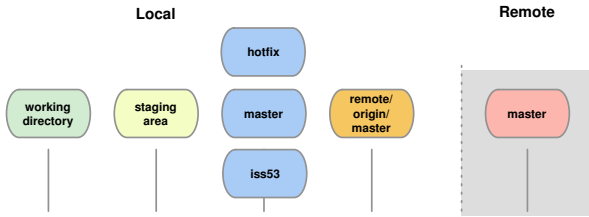




Dinamica del branching – ulteriori branch (3/3)



`git checkout master`
`git merge iss53`





Contribuire a un progetto via GitHub

Scenario 3: Contribuire a un progetto software localizzato su **GitHub**.



GitHub (1/2)

D: Avete mai sentito parlare di GitHub?

The screenshot shows the GitHub web interface for the `numpy/numpy` repository. At the top, the browser address bar shows the URL `https://github.com/numpy/numpy`. The GitHub header includes the logo, a search bar, and navigation links like `Explore`, `Features`, `Enterprise`, and `Blog`. Below the header, the repository name `numpy / numpy` is displayed, along with statistics: 917 stars and 398 forks. The main content area shows repository details: `10,000+` commits, `12` branches, `75` releases, and `142` contributors. A progress bar is visible below these stats. On the right, a sidebar contains links to `Code`, `Issues` (782), `Pull Requests` (34), and `Wiki`. At the bottom of the main area, there is a message about merging pull request `#3636` from `charris/fix-nanfunction-deprecations`.



GitHub (2/2)

Che cos'è GitHub?

- Wikipedia (tradotto): *“GitHub è un servizio di hosting basato sul web che usa **git** per progetti di sviluppo software”*.
- 200 milioni di repository (Giugno 2022).
- Commerciale...
- ... ma non ostile ai progetti Free, Libre, and Open Source.
- Per software si intendono non solo i programmi, ma qualunque entità rappresentabile in forma digitale ...
- ... però Per quello memorizzabili in “formato testo” il sistema *funziona meglio*
- per scrivere un libro con LaTeX funziona benissimo



Contribuire a progetti tramite GitHub

Assunti

- Usi o conosci un software e ti senti pronto a contribuirvi
- Il progetto software viene ospitato all'url <http://github.com>

Idea intuitiva

- Non si fanno **push** dei cambiamenti nel repository principale
- Invece si crea una copia pubblica del repository principale you create a public copy (**fork**) ...
- ... e poi si fa push dei cambiamenti in quella.
- Infine si chiede ai proprietari del progetto (del repository principale) se a loro piacciono tali cambiamenti e se vogliono fare un merge di essi (**pull request**).



Non è per tutti!



torvalds commented

I don't do github pull requests.

github throws away all the relevant information, like having even a valid email address for the person asking me to pull. The diffstat is also deficient and useless.

Git comes with a nice pull-request generation module, but github instead decided to replace it with their own totally inferior version. As a result, I consider github useless for these kinds of things. It's fine for *hosting*, but the pull requests and the online commit editing, are just pure garbage.

I've told github people about my concerns, they didn't think they mattered, so I gave up. Feel free to make a bugreport to github.

Linus

...

<https://github.com/torvalds/linux/pull/17>



Ricetta (1/4)

- Registrarsi su <http://github.com>
- Visitare la pagina GitHub del progetto software e fare un Fork di esso:

The screenshot shows the GitHub interface for the 'numpy/numpy' repository. The browser address bar displays 'https://github.com/numpy/numpy'. The repository is public, and the 'Fork' button is highlighted with a red circle. The repository statistics show 10,000+ commits, 12 branches, 75 releases, and 142 contributors. The 'Code' tab is selected, and the 'Issues' tab shows 782 issues. The 'Pull Requests' tab shows 34 pull requests. The 'Wiki' tab is also visible.



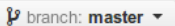
Ricetta (2/4)

- Fare un **clone** della propria copia del progetto sul proprio computer

```
git clone git@github.com:<login>/<project>.git
```

- Creare un **branch** per ospitare i propri miglioramenti.

- `git branch <new-feature>`
- `git checkout <new-feature>`






Ricetta (3/4)

- Aggiungere i propri miglioramenti.
 - `git add <new-file>`
 - `git commit -m ...`
- Fare un **push** dei propri miglioramenti.
`git push origin <new-feature>`



Ricetta (4/4)

● Mandare una richiesta di **pull**.


 Compare & pull request

Write


Preview

Comments are parsed with [GitHub Flavored Markdown](#)

Leave a comment



Attach images by dragging & dropping, [selecting them](#), or pasting from the clipboard.


✓ Able to merge
These branches can be automatically merged

Send pull request



Spiegazione dettagliata

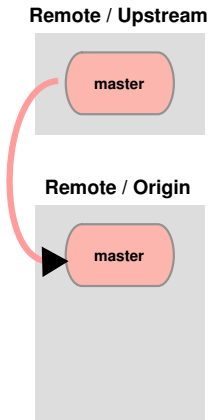
Remote / Upstream



C'è un progetto ospitato su un repository GitHub remoto (**upstream**).
Si vuole contribuire a migliorarlo.



Spiegazione dettagliata

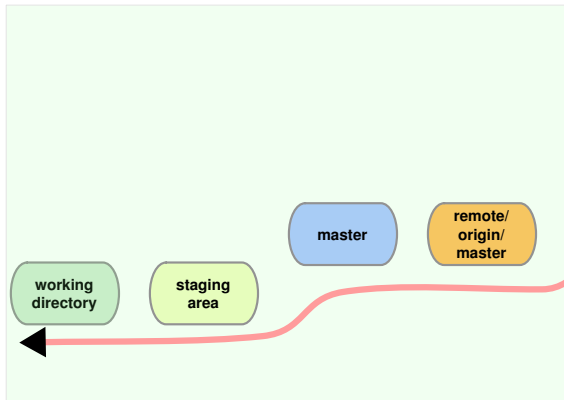


Quindi si fa un **fork** per crearne una copia (remota):
`git clone --bare <UPSTREAM_URL>`



Spiegazione dettagliata

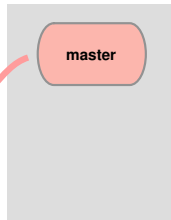
Local



Remote / Upstream



Remote / Origin

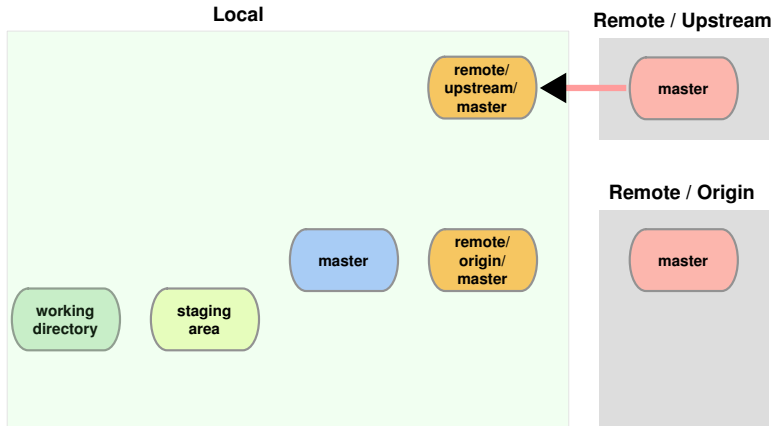


Ora si clona la copia sul computer locale:

```
git clone <ORIGIN_URL>
```



Spiegazione dettagliata

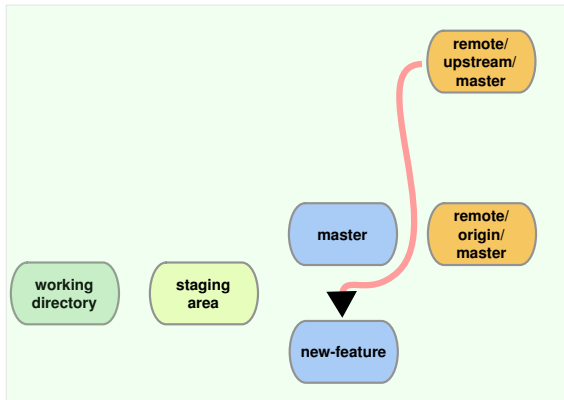


```
git remote add upstream <UPSTREAM_URL>  
git fetch upstream
```



Spiegazione dettagliata

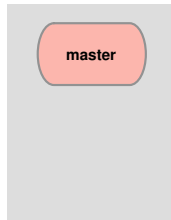
Local



Remote / Upstream



Remote / Origin

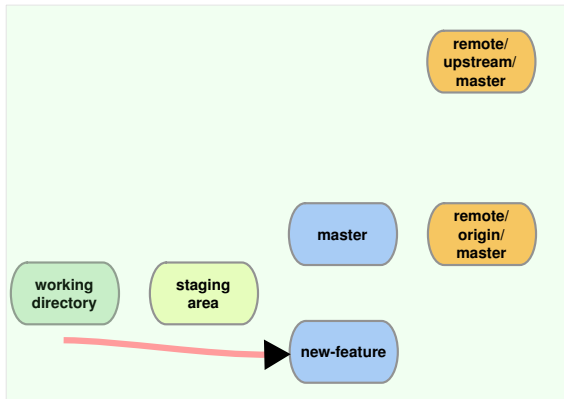


```
git branch new-feature upstream/master  
git checkout new-feature
```



Spiegazione dettagliata

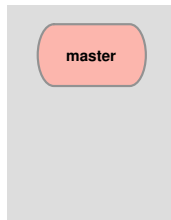
Local



Remote / Upstream



Remote / Origin

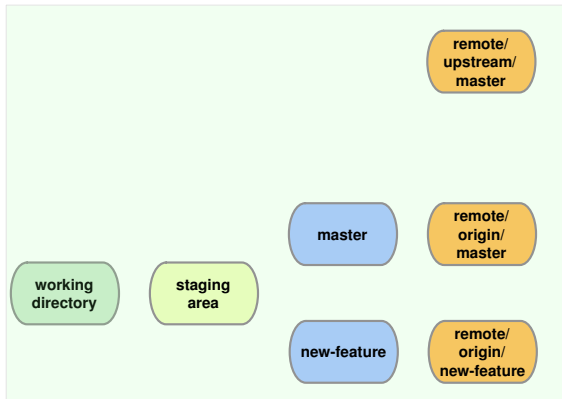


```
git add ...  
git commit ...
```



Spiegazione dettagliata

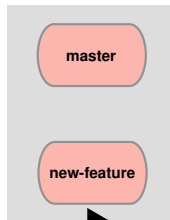
Local



Remote / Upstream



Remote / Origin

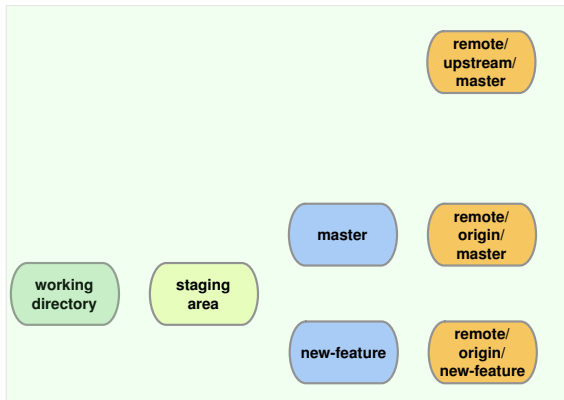


Pubblica le nuove caratteristiche (feature):
`git push origin new-feature`



Spiegazione dettagliata

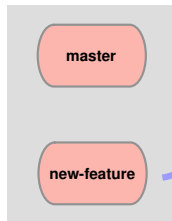
Local



Remote / Upstream



Remote / Origin



Notifica i proprietari del repository principale sulle nuove caratteristiche (**new-feature**)

loro potranno fare: `git fetch` + (eventualmente) `git merge`



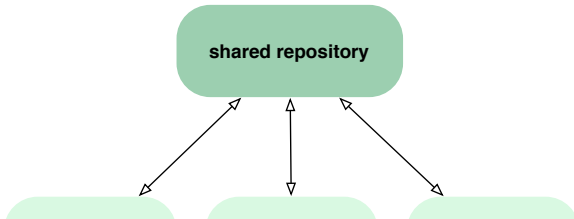
Creare un repository remoto/condiviso (1/3)

SCOPO: voglio condividere il mio repository locale in modo che altri possano fare **push**.

“Perché non posso semplicemente cambiare i permessi nel mio repository **locale**?”

- Certamente si può...
- ...ma i tuoi colleghi non potranno fare push (**read-only**).

Per poterlo avere con permessi **read-write**: occorre fare un repository **remoto** *condiviso*.





Creare un repository remoto/condiviso (2/3)

Hai un repository locale e lo vuoi condividere da (**ssh**) da un server remoto su cui i tuoi colleghi hanno già accesso

Sul server *remoto* crea un repository vuoto+condiviso:

- `mkdir newproject`
- genera i permessi necessari per il *gruppo*: `chmod g+rws newproject`
- `cd newproject`
- `git --bare init --shared=group`



Creare un repository remoto/condiviso (3/3)

Sulla macchina *locale* fai push del tuo repository verso quello remoto:

```
git remote add origin  
                ssh://remote.com/path/newproject  
git push -u origin master
```

E poi alla fine ognuno può fare un clone del repository condiviso:

```
git clone ssh://remote.com/path/newproject
```



Crediti

- **Rike-Benjamin Schuppner**
- Zbigniew Jedrzejewski-Szmek
- Tiziano Zito
- Bastian Venthur
- <http://progit.com>
- apcmag.com
- lwn.net
- <http://www.markus-gattol.name/ws/scm.html>
- [http://matthew-brett.github.io/pydagogue/gitwash/
git_development.html](http://matthew-brett.github.io/pydagogue/gitwash/git_development.html)



Se si vuole sapere di più su git

Sul funzionamento di git:

- gitfoundations, di Matthew Brett: <http://matthew-brett.github.com/pydagogue/foundation.html>
- The gitparable, by Tom Preston-Werner: <http://tom.preston-werner.com/2009/05/19/the-git-parable.html>

Ottime guide:

- “Pro Git” book: <http://git-scm.com/book> (gratis)
- gitmagic: <http://www-cs-students.stanford.edu/~blynn/gitmagic/>

Per contribuire a un progetto ospitato su GitHub:

- “Gitwash”, di Matthew Brett: http://matthew-brett.github.io/pydagogue/gitwash/git_development.html



Materiale interessante

Gource:

<http://code.google.com/p/gource/>



Licenza per la parte su Git

Copyright Emanuele Olivetti, 2014

Questa distribuzione viene distribuita con la licenza

Creative Commons *Attribution* 3.0

<https://creativecommons.org/licenses/by/3.0/>

I diagrammi per il branching sono presi da *Pro Git*, (copyright S.Chacon, 2009) e sono distribuiti con la licenza Creative Commons 3.0 Attribution-Non Commercial-Share Alike.



Domande?

Fine della seconda lezione.