

Red Shift

Git me started!



Rodolfo Carobene
19/01/2023



Timetable




1. Introduzione a Git
2. Comandi principali
3. Advanced git
4. Introduzione a GitHub
5. "Comandi" e features di GitHub
6. Advanced GitHub



Obiettivi

1. Imparare le funzionalità base
2. Avere un'idea delle funzionalità più avanzate
3. Capire come usare Git e GitHub per ottimizzare il normale lavoro

Warning!

1. Esempio di progetto **Python** 
2. Sistema operativo: **Arch Linux** 
3. Shell: **zsh**
4. Editor di testo: **Neovim** 



Let's git it started!

Cos'è Git?



Cos'è Git?



Git è un software per il controllo versione distribuito creato da Linus Torvald nel 2005 per supportare lo sviluppo di Linux.

Git è uno strumento fondamentale per chiunque sviluppi software sia in azienda che nel mondo della ricerca. Non esistono competitor reali.

Git snapshots

Empty repository



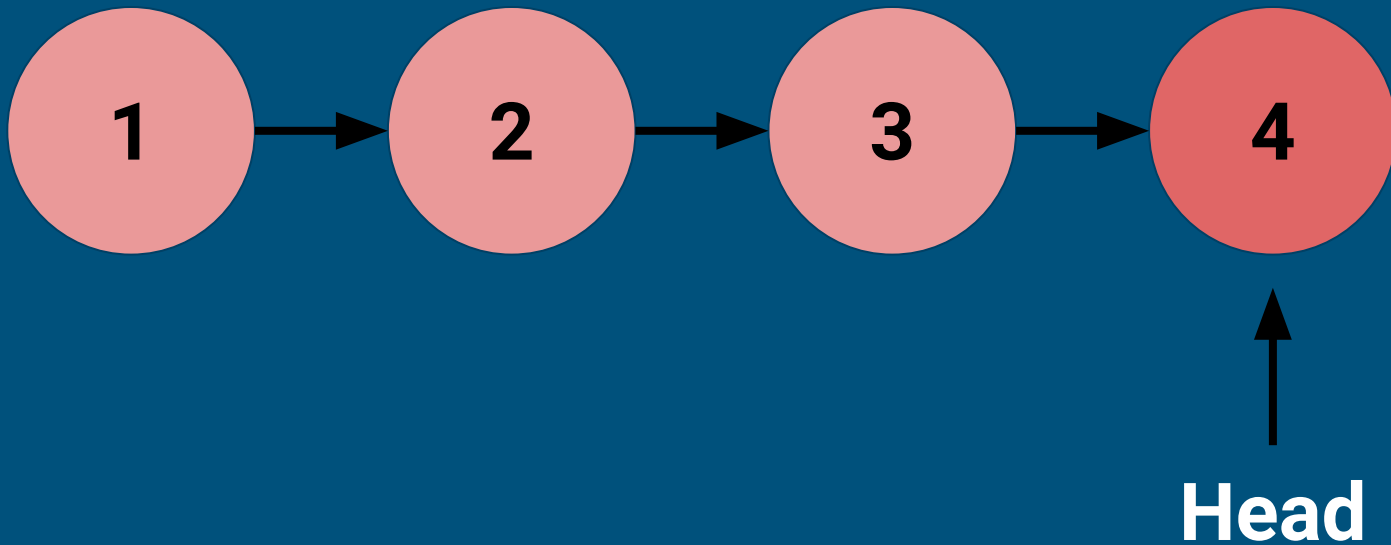
Head

```
example@system: ~Documents/test_git
```

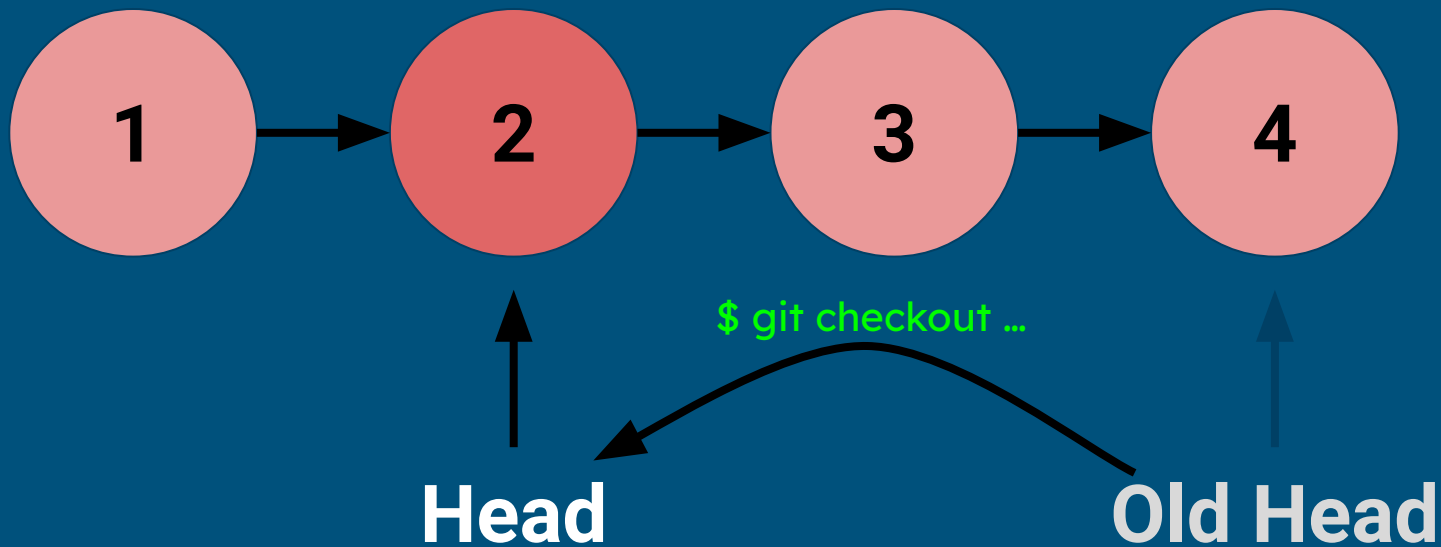
```
> git init
```

```
Initialized empty Git repository in  
/home/example/Documents/test_git
```

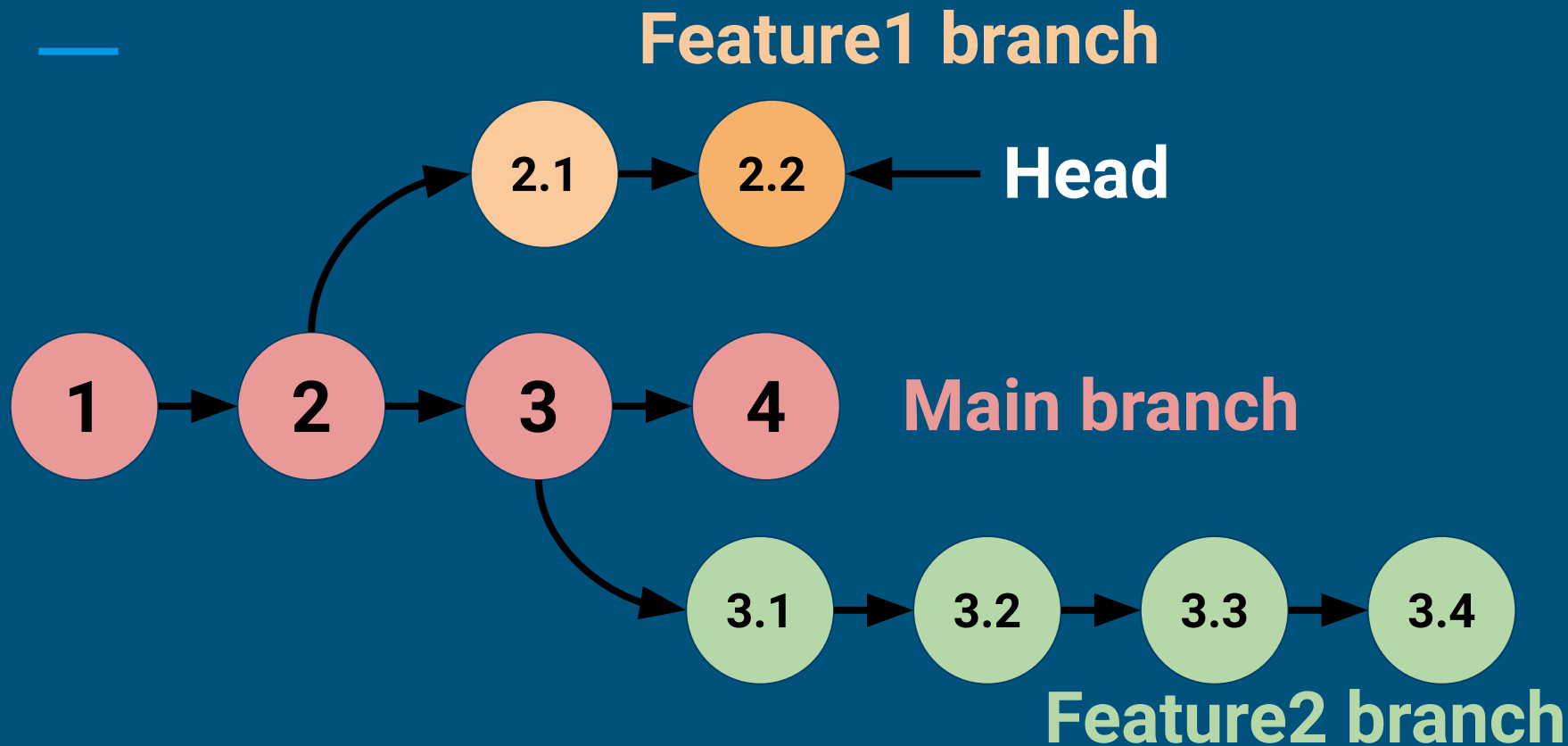
Git snapshots: commits



Git snapshots: commits



Git branches



Commit

1. Hash unico
2. Autore
3. Data e ora
4. Commit message

commit

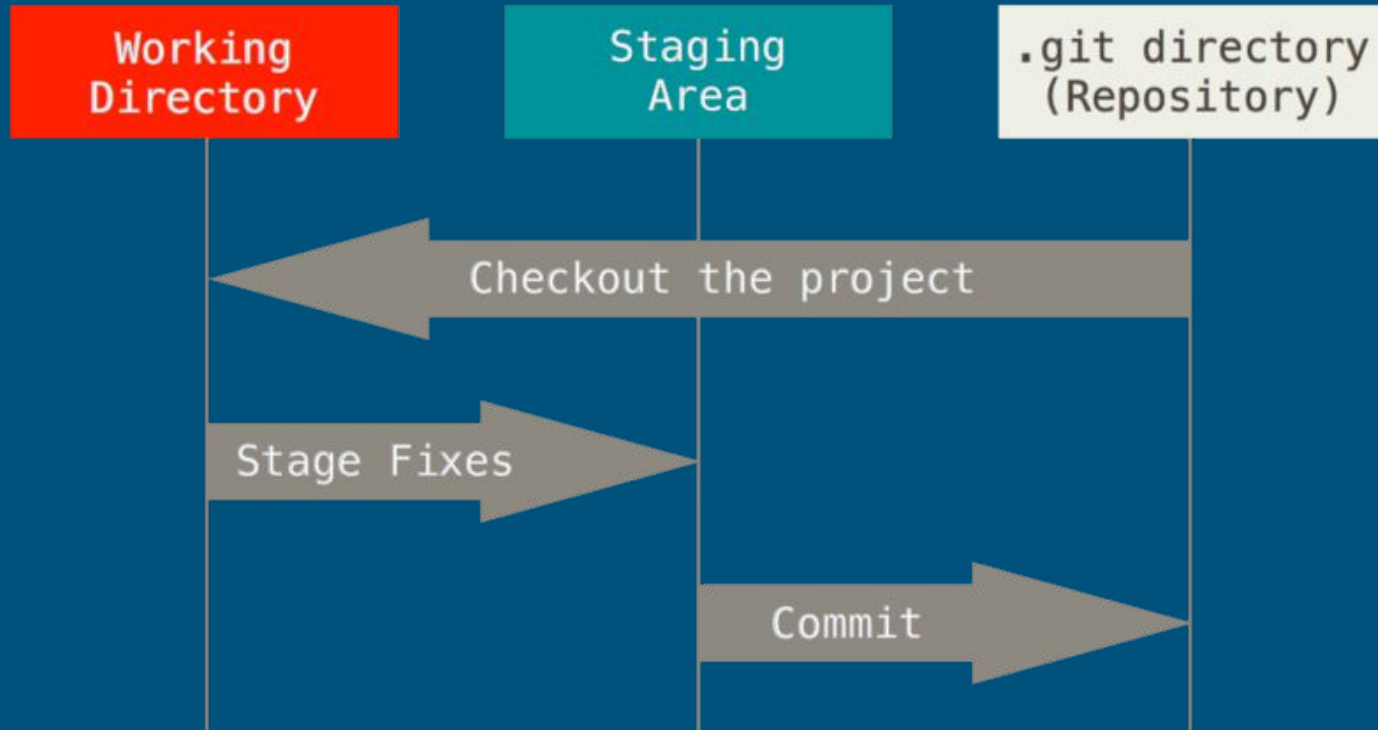
9261ccf03b0f4076fc6c533ed1bc030999e38269

Author: Rodolfo Carobene
<rodolfo.carobene@gmail.com>

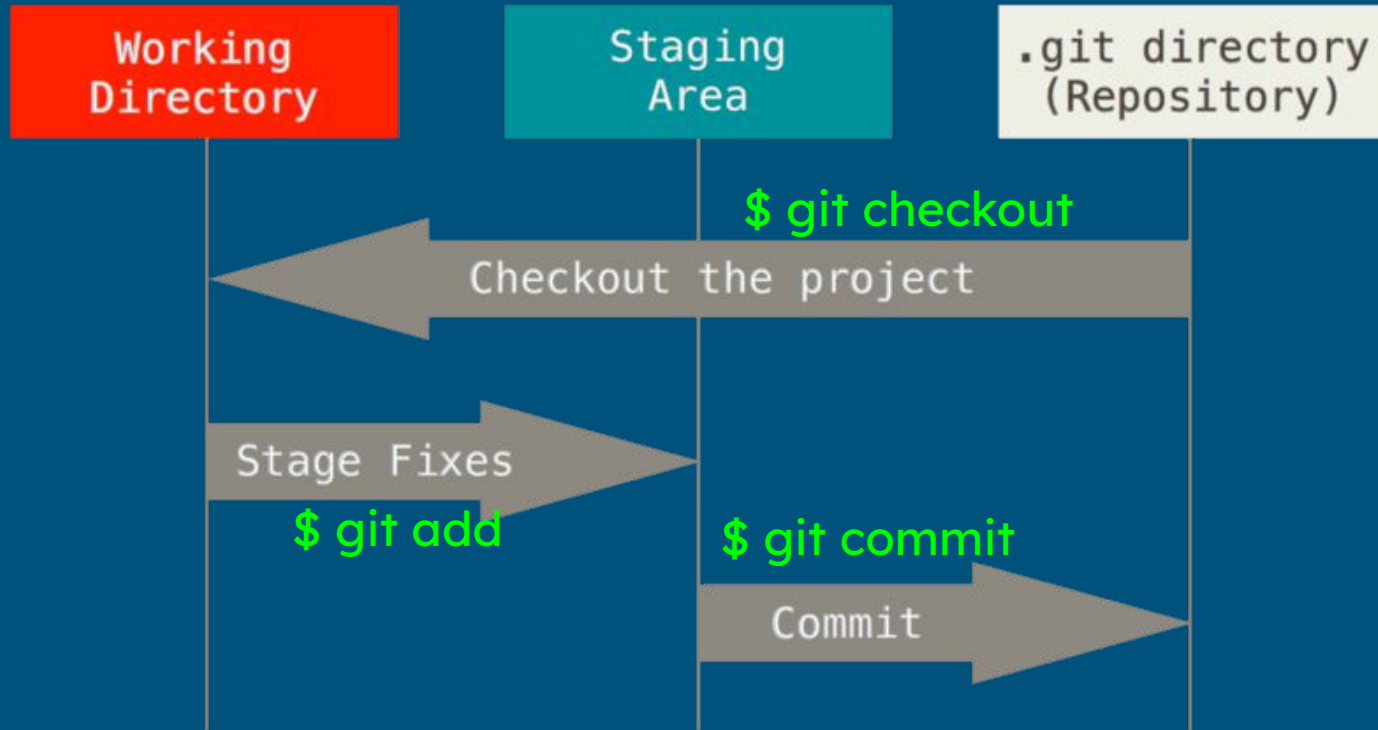
Date: Fri Dec 22 10:15:48 2023 +0100

Added feature ...

Workflow di commit




Workflow di commit



Workflow di commit

1. git add
2. git commit
3. git status
4. git log



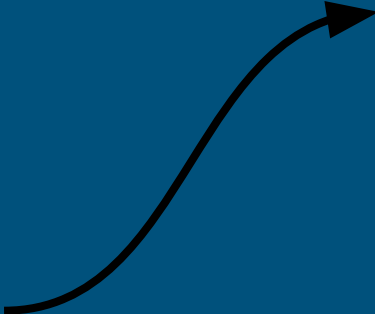
Il comando "**git add**" in Git viene utilizzato per aggiungere le modifiche apportate ai file alla **staging area**. Con "**git add**", è possibile selezionare le modifiche specifiche da includere nel prossimo commit.

```
> git add file1 file2 file3
```

```
> git add .
```

Workflow di commit

1. git add
2. git commit
3. git status
4. git log



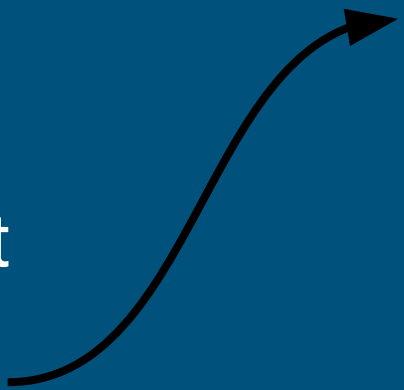
Il comando "**git commit**" viene utilizzato per registrare le modifiche nella staging area in un **nuovo commit** nella repository Git

> git commit -m "Descrizione"

> git commit

Workflow di commit

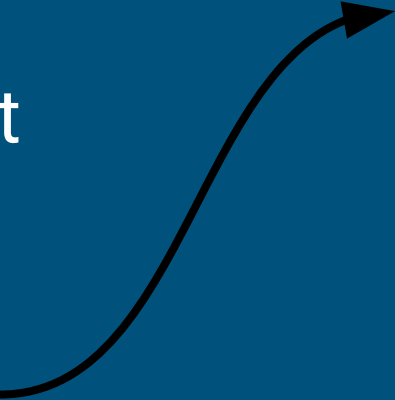
1. git add
2. git commit
3. git log
4. git status



Il comando "**git log**" fornisce la cronologia dei **commit**, inclusi hash, autori, data e messaggio di commit

> git log

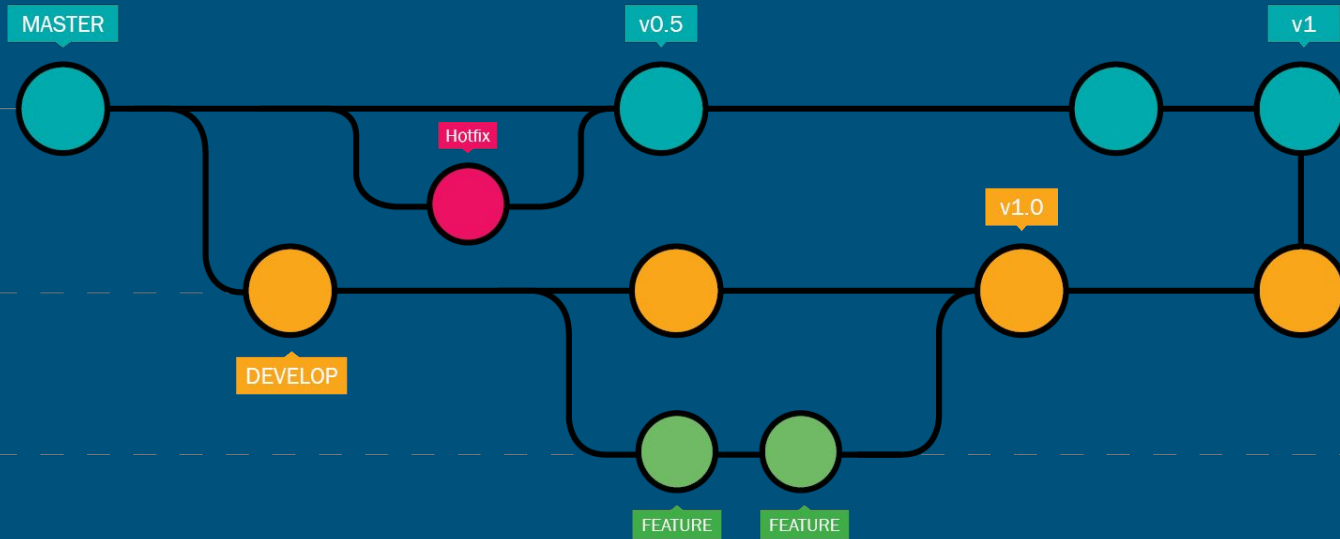
Workflow di commit

1. git add
 2. git commit
 3. git log
 4. git status
- 

Il comando "**git status**" fornisce informazioni sullo stato corrente del repository, mostrando i **file modificati**, quelli nella **staging area** e quelli **non tracciati**.


```
> git status
```

Workflow di branch



Workflow di branch

1. git branch
2. git checkout
3. git merge
4. git diff



Il comando "**git branch**" in Git viene utilizzato per visualizzare la lista delle branch presenti nel repository. Con le giuste opzioni si può usare per collegare una branch GitHub o cancellare una branch locale


> git branch

> git branch -D nome_branch

> git branch --set-upstream ...

Workflow di branch

1. git branch
2. git checkout
3. git merge
4. git diff



Il comando "**git checkout**" serve per spostarsi fra le **branch** (o i commit). Si usa anche per creare una nuova branch.

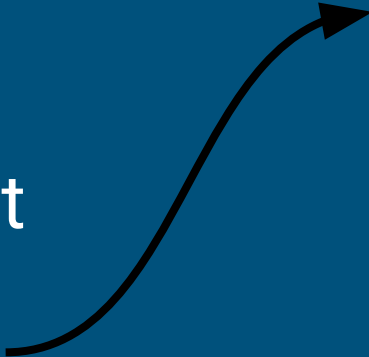
```
> git checkout nome_branch
```

```
> git checkout hash -- file_name
```

```
> git checkout -b nuova_branch
```

Workflow di branch

1. git branch
2. git checkout
3. git diff
4. git merge



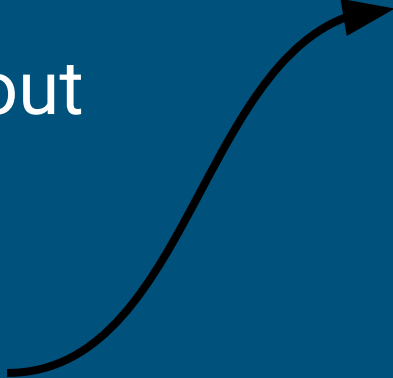
Il comando "**git diff**" mostra le differenze fra due punti dello storico dei commit. Può essere usato fra commit, fra Head e last commit o fra branch.

```
> git diff branch1 branch2
```

```
> git diff
```

```
> git diff hash1 hash2
```

Workflow di branch

1. git branch
 2. git checkout
 3. git diff
 4. git merge
- 

Il comando "**git merge**" viene utilizzato per combinare le modifiche da una branch all'altra.

```
> git merge main  
(merge main into Head)
```

Merge conflicts

example@system:

~Documents/test_git

> git merge main

Auto-merging file.md

CONFLICT (content): Merge conflict in file.md

Automatic merge failed; fix conflicts and then commit the result



Merge conflicts

```
example@system:
```

```
~Documents/test_git
```

```
> git merge main
```

```
Auto-merging file.md
```

```
CONFLICT (content): Merge conflict in file.md
```

```
Automatic merge failed; fix conflicts and then commit  
the result
```

```
<<<<<< HEAD
```

```
this is some content to mess with  
content to append
```

```
=====
```

```
totally different content to merge  
later
```

```
>>>>>> main
```

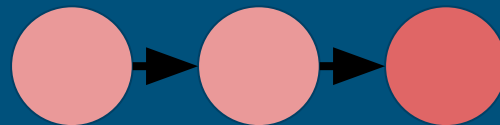
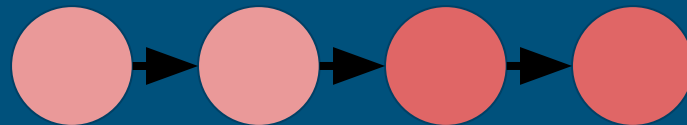
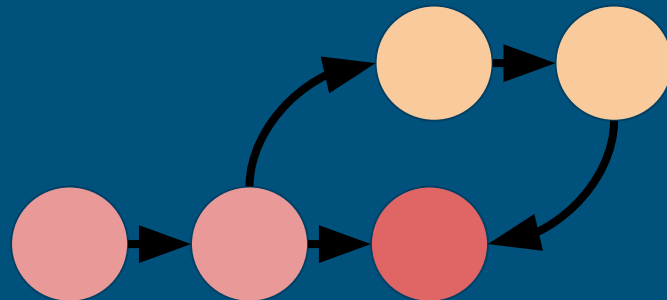
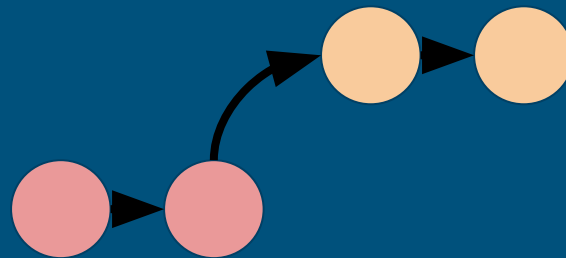

Exercises: batch 1!

https://github.com/rodolfocarobene/git_lecture

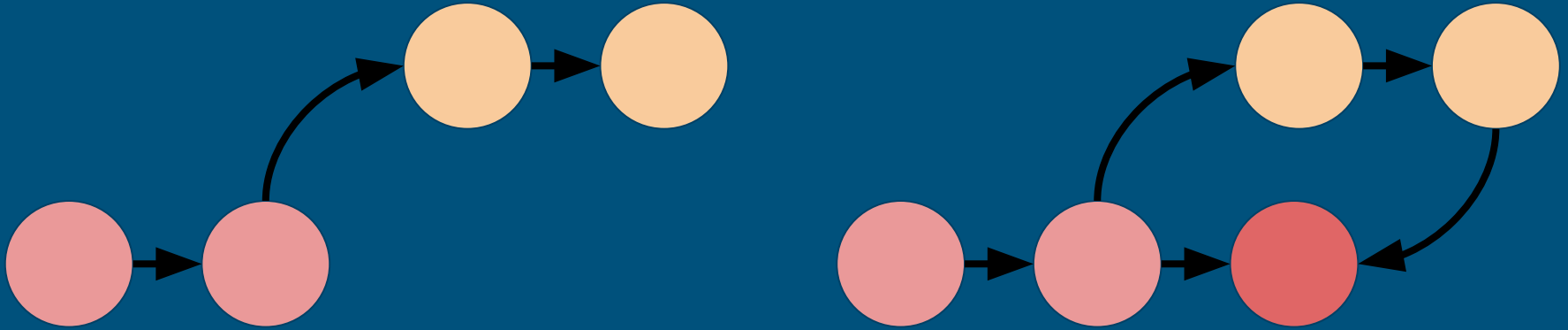
Configura git, inizializza una repository, aggiungi i file principali del tuo progetto. Usa varie branch per aggiungere feature. Uniscile con merge e risolvi dei conflitti!

Merge strategies:

1. git merge
2. git rebase
3. git squash

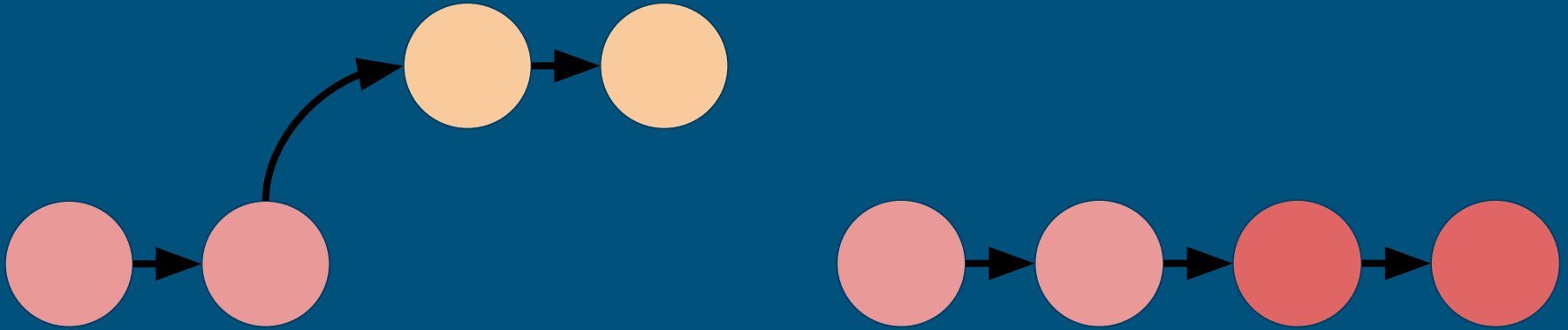


Merge strategies: git merge



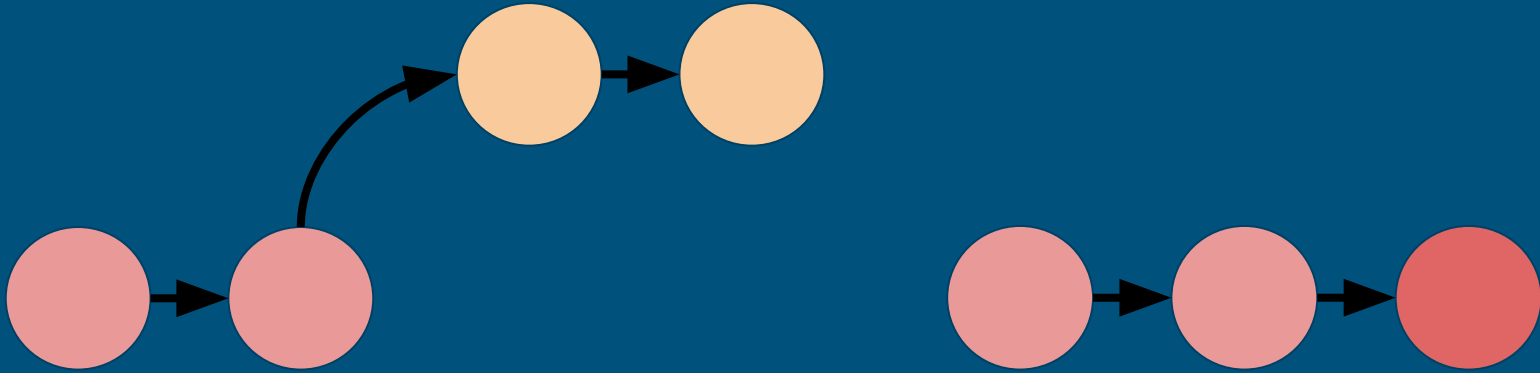
Nuovo commit di merge che rimanda alle due “parent” branch

Merge strategies: git rebase



Tutti i commit della feature branch vengono spostati su quella principale

Merge strategies: `git merge --squash`



Tutti i commit della feature branch vengono condensati in un singolo commit e aggiunti a quella principale

More commands



1. git stash
2. git restore
3. git blame
4. git cherry-pick
5. git rm
6. git bisect
7. git tag

Il comando "**git stash**" permette di compiere salvataggi temporanei delle modifiche nella working dir

> git stash

> git stash save "description"

> git stash list

> git stash pop

> git stash apply <name>

> git stash drop <name>

More commands

1. git stash
2. git restore
3. git blame
4. git cherrypick
5. git rm
6. git bisect
7. git tag



Il comando "**git restore**" permette di scartare le modifiche nella working directory

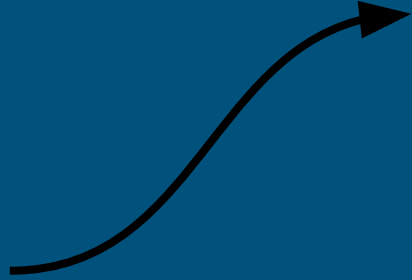
> git restore .

> git restore --staged .

> git restore --staged --worktree .

More commands

1. git stash
2. git restore
3. git blame
4. git cherry-pick
5. git rm
6. git bisect
7. git tag



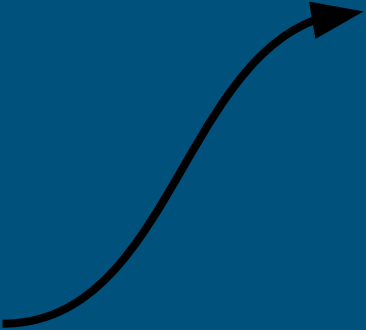
Il comando "**git blame**" mostra autori e commit dei file, linea per linea. Si può tipicamente integrare nel proprio IDE.

```
> git blame <nome_file>
```

```
> git blame <file> -L 12,18
```


More commands

1. git stash
2. git restore
3. git blame
4. git cherrypick
5. git rm
6. git bisect
7. git tag

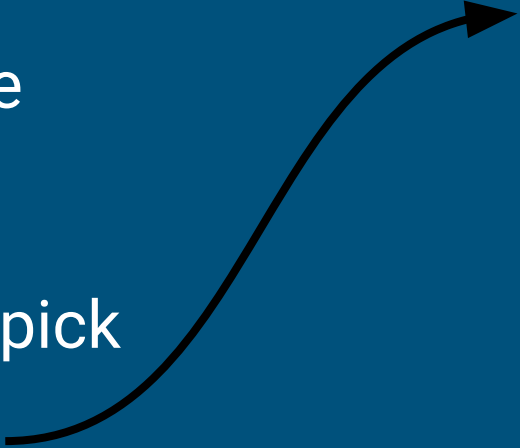


Il comando "**git cherrypick**"
permette di fare merge di un singolo
commit (di un'altra branch) su quella
di lavoro

```
> git cherrypick <hash_commit>
```

More commands

1. git stash
2. git restore
3. git blame
4. git cherrypick
5. git rm
6. git bisect
7. git tag



Il comando "**git rm**" è usato per smettere di tenere sotto version-control un determinato file. Attenzione! Non cancella il file dalla history

```
> git rm <file_name>
```

```
> git rm --cached <file_name>
```

More commands

1. git stash
 2. git restore
 3. git blame
 4. git cherrypick
 5. git rm
 6. git bisect
 7. git tag
- 

Il comando "**git bisect**" può essere usato per identificare il commit origine di un certo problema.

```
> git bisect start  
> git bisect bad  
> git bisect good <hash_commit>  
  
> git bisect run <command_line>  
  
> git bisect reset
```

More commands

1. git stash
 2. git restore
 3. git blame
 4. git cherry-pick
 5. git rm
 6. git bisect
 7. git tag
- 

Il comando "**git tag**" può essere usato per dare un nome (es: v0.1.0) a uno specifico punto del git tree

```
> git tag <nuovo_tag>
```

```
> git tag -a <nuovo_tag> -m  
"message"
```

```
> git tag
```

```
> git checkout <nome_tag>
```

Special git files

1. .gitignore
2. .gitkeep
3. .gitmodules
4. .gitattributes
5. .gitconfig



Il file "**.gitignore**" viene usato per indicare di quali file e cartelle non è necessario tenere traccia

Special git files

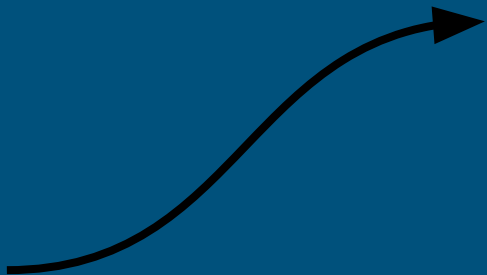
1. .gitignore
2. .gitkeep
3. .gitmodules
4. .gitattributes
5. .gitconfig



Il file **".gitkeep"** viene usato per aggiungere alla repository cartelle vuote. Il suo nome è una convenzione, non è un vero file git!

Special git files

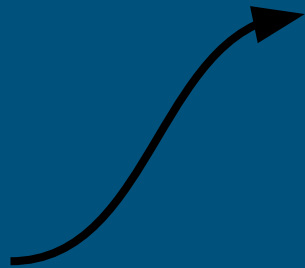
1. .gitignore
2. .gitkeep
3. .gitmodules
4. .gitattributes
5. .gitconfig



Il file "**.gitmodules**" viene usato per includere e gestire altri moduli all'interno della repository. I **submodules** sono a tutti gli effetti altre repository git indipendenti, ma incluse nel progetto principale.

Special git files

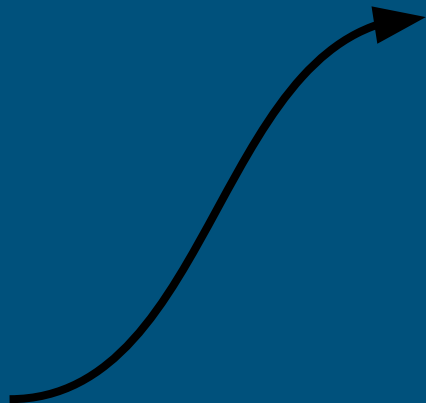
1. .gitignore
2. .gitkeep
3. .gitmodules
4. .gitattributes
5. .gitconfig



Il file **".gitattributes"** viene usato per assegnare "attributi" a specifici tipi (estensioni) di file. Può controllare le merge-strategies per ciascun tipo di file o indicare cosa salvare e cosa no

Special git files

1. .gitignore
2. .gitkeep
3. .gitmodules
4. .gitattributes
5. .gitconfig



Il file "**.gitconfig**" contiene le preferenze globali / locali con cui git lavora.
Ad esempio contiene nome / email dell'autore principale; editor preferenziale; nome della branch iniziale ed alias

Exercises: batch 2!

Aggiungi i file principali (.gitignore) al tuo progetto. Sperimenta con i comandi appena definiti

Git Aliases: custom commands

- st = status
- ls = branch
- lg = log --graph --decorate --date=short --format='%C(bold blue)%h %C(bold green)%ad %C(auto)%d %C(white)%s%C(reset)'
- c = commit
- pl = pull --all
- ps = push

example@system:

~Documents/test_git

```
> git config --global alias.st status
```

```
> git config --get-regexp alias
```

...

...

Git Hooks

- Script automatici in corrispondenza di azioni di git (esempi: pre-commit, post-commit, pre-push...)
- Alcuni esempi generati in automatico nella cartella `./git/hooks/`
- Di default usano comandi Bash Shell, ma si può usare ogni linguaggio di scripting (esempio: Python)
- Completamente locali!

Pre-commit: A framework for managing and maintaining multi-language pre-commit hooks.

- id: trailing-whitespace
- id: end-of-file-fixer
- id: debug-statements
- id: black
- id: isort
- id: pycln
- id: pydocstyle

example@system:

`~Documents/test_git`

`> pip install pre-commit`

`> pre-commit install`

`> pre-commit run --all-files`

Rimozione di commit



example@system:

~Documents/test_git

> git reset --hard HEAD~n

> git revert HEAD~n..HEAD

Exercises: batch 3!

Installa pre-commit. Aggiungi dei pre-commit importanti per il tuo progetto. Definisci degli alias utili. Fai un commit e cancellalo (in più modi magari!)

Break !



Cos'è GitHub?



Cos'è GitHub?



GitHub è una piattaforma di hosting basata su Git.

GitHub offre una piattaforma centralizzata in cui gli sviluppatori possono ospitare i loro repository Git, collaborare su codice sorgente, gestire problemi, pianificare progetti e molto altro ancora. È ampiamente utilizzato sia nel settore aziendale che nella comunità open source.

Comandi di git principali

1. git remote
2. git push
3. git fetch
4. git pull
5. git clone



Il comando "**git remote**" si usa per gestire una repository remota. Ad esempio per aggiungerla o rimuoverla

```
> git remote -v
```

```
> git remote add origin  
  <my_repo_URL>
```

Comandi di git principali

1. git remote
2. git push
3. git fetch
4. git pull
5. git clone



Il comando "**git push**" serve per inviare i commit locali alla repository remota

> git push

> git push --force

> git push --all --tags

Comandi di git principali

1. git remote
2. git push
3. git fetch
4. git pull
5. git clone



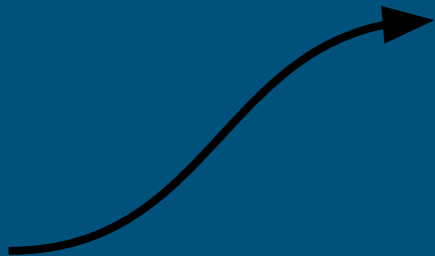
Il comando "**git fetch**" serve per scaricare i commit remoti nella repository locale.

> **git fetch**

> **git fetch --all**

Comandi di git principali

1. git remote
2. git push
3. git fetch
4. git pull
5. git clone



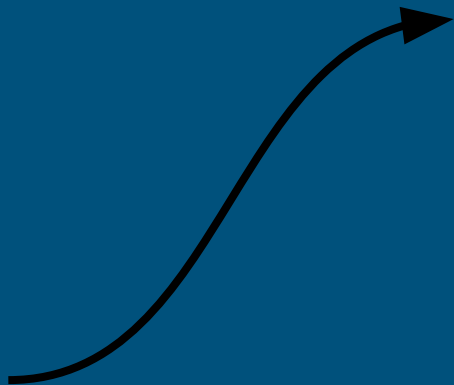
Il comando "**git pull**" serve per scaricare **e unificare** (merge) i commit remoti nella repository locale.

```
> git pull
```

```
> git pull [ --squash | -r ]
```

Comandi di git principali

1. git remote
2. git push
3. git fetch
4. git pull
5. git clone



Il comando "**git clone**" serve per copiare in locale una repository remota.

```
> git clone <url_repo>
```

Git Pull --force ?



```
example@system:
```

```
~Documents/test_git
```

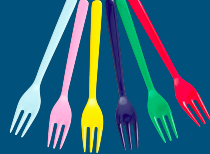
```
> git fetch
```

```
> git branch backup_current
```

```
> git reset --hard origin/main
```


Exercises: batch 4!

Riprendiamo il progetto di prima e mettiamolo su GitHub! Creiamo la repository




GitHub forks

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk ().*

Owner *

 rodolfocarobene ▾

Repository name *

/ numpy

✔ numpy is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

The fundamental package for scientific computing with Python.
























☒ Copy the `main` branch only

Contribute back to numpy/numpy by adding your own branch. [Learn more.](#)

 You are creating a fork in your personal account.

Create fork

GitHub pull-requests


 API: Introduce stringdtype [NEP 55] ✖	 54
#25347 opened 3 weeks ago by  ngoldbaum · Draft  updated 1 hour ago	
30 - API	
 BUG: Fix incorrect 'inner' method type annotation in <code>__array_ufunc__</code> ✖	
#25510 opened 12 hours ago by  zachbrugh  updated 12 hours ago	
00 - Bug	
 API: adjust <code>ndfft</code> <code>s</code> param to array API ✔	 5
#25495 opened 4 days ago by  lucascolley  updated 18 hours ago	
30 - API	
 DOC: mention string, bytes, and void dtypes in dtype intro ✔	 27
#25507 opened 2 days ago by  ngoldbaum  updated 18 hours ago	
04 - Documentation	
 DOC: add warning to <code>allclose</code>, revise "Notes" in <code>isclose</code> ✖	 1
#24407 opened on Aug 13 by  OverLordGoldDragon  updated yesterday	
04 - Documentation	
 DOC: Improve <code>np.mean</code> documentation of the out argument ✖	 8
#25431 opened 2 weeks ago by  pieleric  updated 2 days ago	

GitHub issues

✓ 1,967 Open 10,088 Closed


Open all Author ▾ Label ▾ Projects ▾ Milestones ▾ Assignee ▾ Sort ▾

• NumPy 2.0 development status & announcements 30

#24300 opened on Jul 31 by  rgommers updated 2 hours ago 2.0.0 release


62 - Python API 63 - C API Tracking / planning

• DISCUSS: remove remaining usages of character codes in array reprs 13

#25508 opened 2 days ago by  ngoldbaum updated 2 hours ago


15 - Discussion

• BUG: Wrong type annotation in ndarray.__array_ufunc__ 3

#25499 opened 4 days ago by  aerobio updated 13 hours ago


00 - Bug Static typing

• BUG: Rounding floats which are already equal to an integer changes the value 11

#20514 opened on Dec 4, 2021 by  FudgeMunkey updated 2 days ago

00 - Bug

• BUG: `assert_allclose` cannot handle object arrays 1

#25496 opened 4 days ago by  h-vetinari updated 2 days ago

component: numpy.testing

Markdown GitHub flavored

My GitHub-Flavored Markdown Example

Introduction

Welcome to my example Markdown document. This is a simple guide to showcase some common Markdown features.

Text Formatting

This text is italicized. ; **This text is bold.** ; ***This text is bold and italicized.***

Lists

- Item 1
- Item 2
- Item 3

1. First item
2. Second item
3. Third item

Code

Inline code: ``printf("Hello, Markdown!");``

Code block:

```
```python
def greet(name):
 print(f"Hello, {name}!")
```
```

Links

[Github](https://github.com)

Images

![GitHub Logo](https://github.githubassets.com/images/modules/logos_page/GitHub-Mark.png)

Tables

| Name | Occupation |
|------|------------|
| John | Developer |
| Jane | Designer |
| Alex | Scientist |

GitHub tags and releases

ReleasesTags

Find a release

Nov 13

charris

<> v1.26.2 ✓

03b6260 ✓

± Commits

Compare ▾

1.26.2 release Latest

NumPy 1.26.2 Release Notes

NumPy 1.26.2 is a maintenance release that fixes bugs and regressions discovered after the 1.26.1 release. The 1.26.release series is the last planned minor release series before NumPy 2.0. The Python versions supported by this release are 3.9-3.12.

Contributors

A total of 13 people contributed to this release. People with a "+" by their names contributed a patch for the first time.

GitHub actions



**Some checks were not successful**
3 cancelled, 2 successful, and 1 failing checks

[Hide all checks](#)

| | | |
|---|---|-------------------------|
|   | Lint-Test-Mypy / build (3.8) (push) Cancelled after 55s | Details |
|   | Docs / deploy (3.9) (push) Successful in 53s | Details |
|   | Lint-Test-Mypy / build (3.9) (push) Cancelled after 58s | Details |
|   | Lint-Test-Mypy / build (3.10) (push) Cancelled after 57s | Details |
|   | Lint-Test-Mypy / build (3.11) (push) Failing after 51s | Details |
|   | pre-commit.ci - pr — checks completed successfully | Details |

GitHub actions: .github/workflows/action.yml

```
name: Your Workflow Name

on:
  push:
    branches:
      - main

jobs:
  your_job_name:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Repository
        uses: actions/checkout@v2

      - name: Set Up Environment
        run: |
          # Your commands to set up the environment

      - name: Run Your Commands
        run: |
          # Your commands to run in the job
```

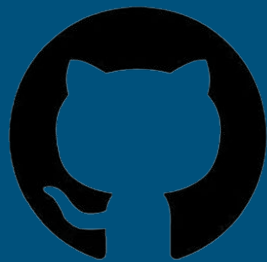
Useful for:

1. code analysis
2. code testing
3. code formatting
4. deployment

Exercises: batch 5!

Esempio di un progetto Python con actions e review. Workflow tipico di collaborazione. Esempi di progetti opensource su GitHub.

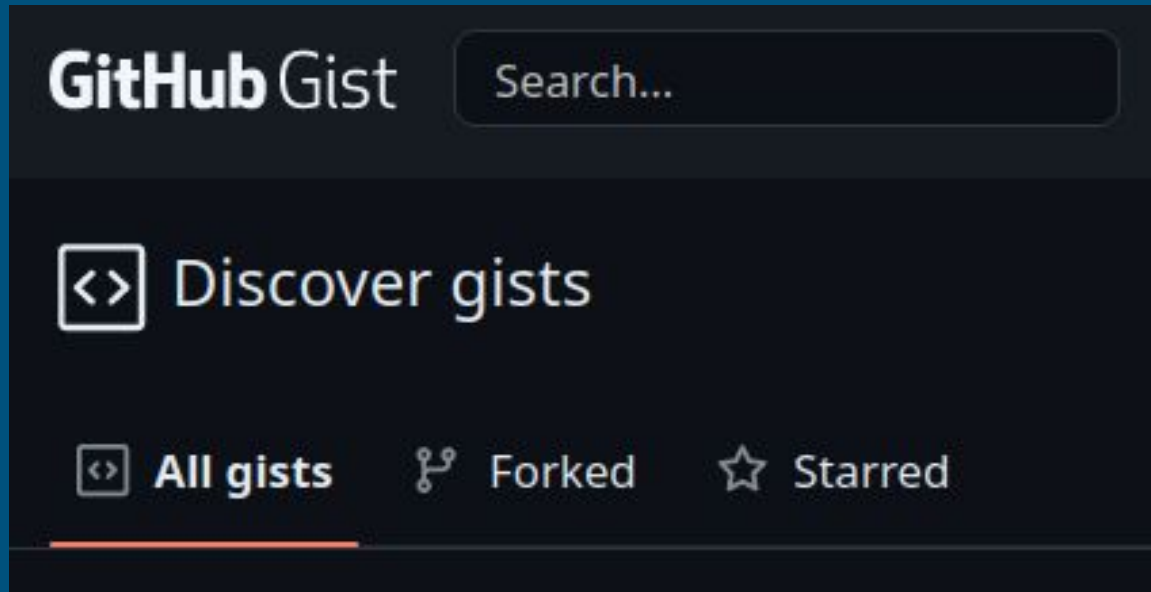
GitHub.io



GitHub Pages

1. Hosted directly on GitHub
2. Usable with raw HTML or frameworks
3. Automatic deployment or with actions

Gist: *gist.github.com/discover*



GitHub achievements badges



The End

