# Git me started!

Rodolfo Carobene

19/01/2023

# Timetable

1. Git introduction
2. Main commands
3. Advanced git
4. GitHub introduction
5. Github “commands” and features
6. Advanced GitHub

# Objectives

1. Learning well the main features
2. Get an idea of the more advanced features
3. Understand how to use Git and GitHub to optimize standard work

# Warning!

1. Example project **Python**
2. Operating system: **Arch Linux**
3. Shell: **zsh**
4. Text editor: **Neovim**

# Let’s git it started!

# What is Git?

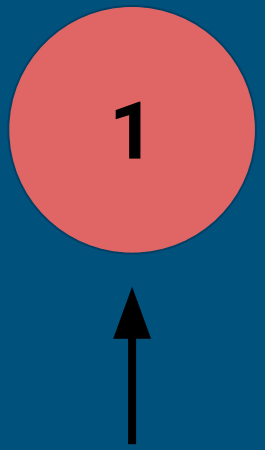
# What is Git?

Git is a distributed version control software created by Linus Torvald in 2005 to support Linux development.

Git is a fundamental tool for anyone developing software both in the company and in the world of research. There are no real competitors.

# Git snapshots

Empty repository



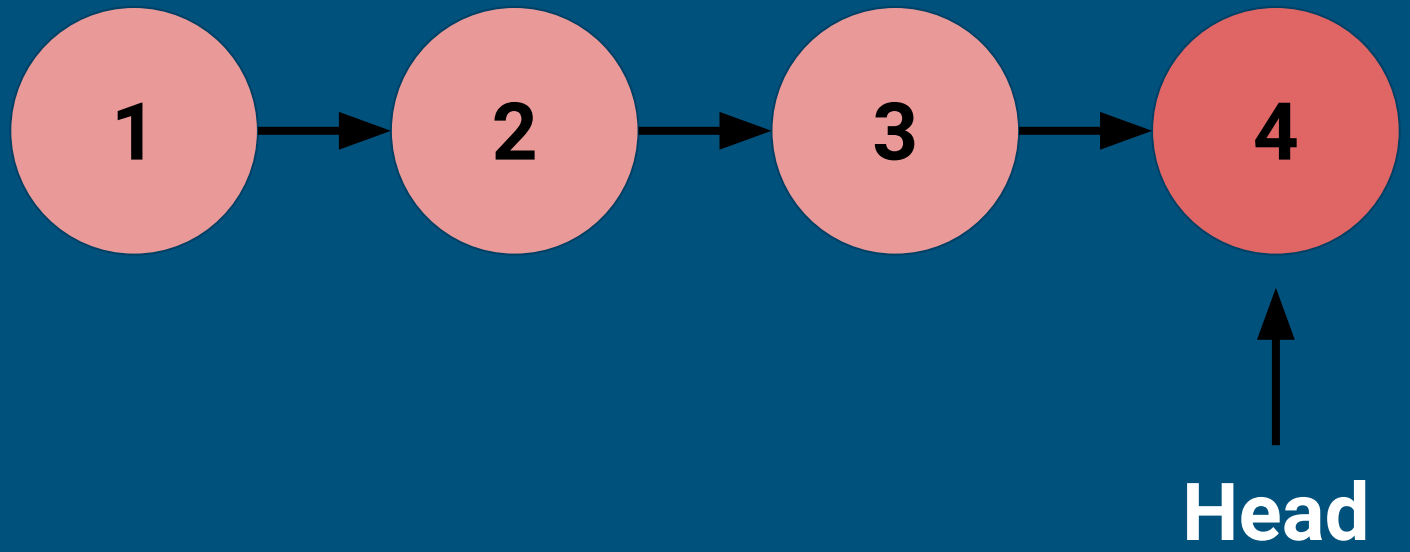

**Head**

```
example@system: ~Documents/test_git

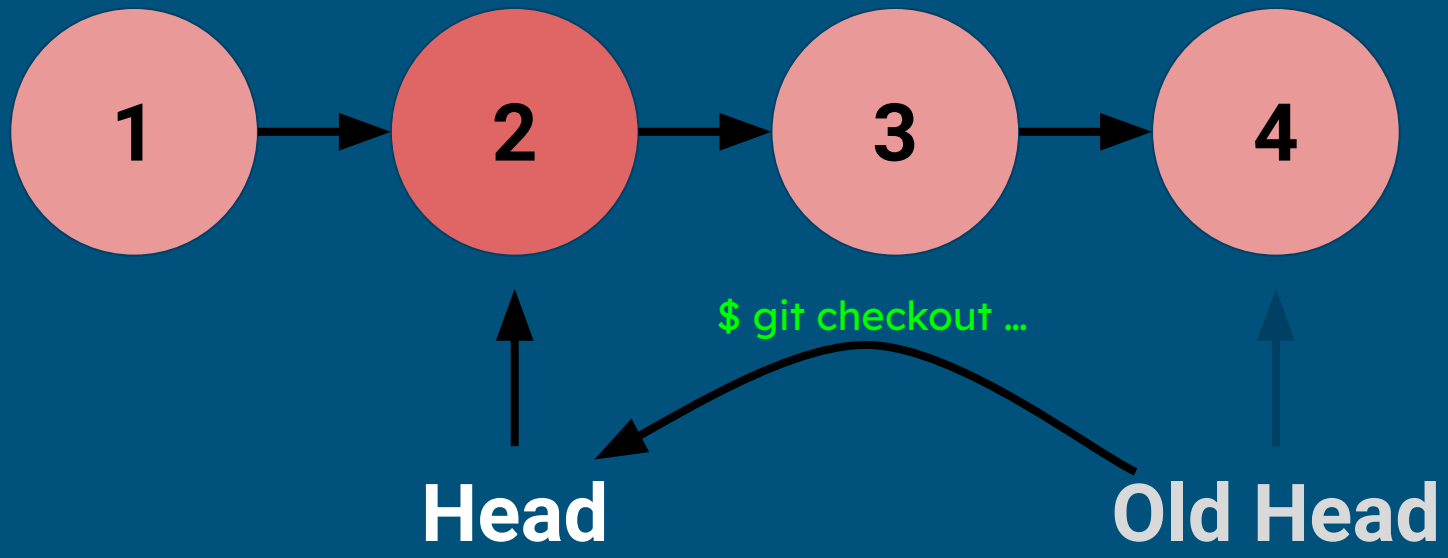
> git init

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

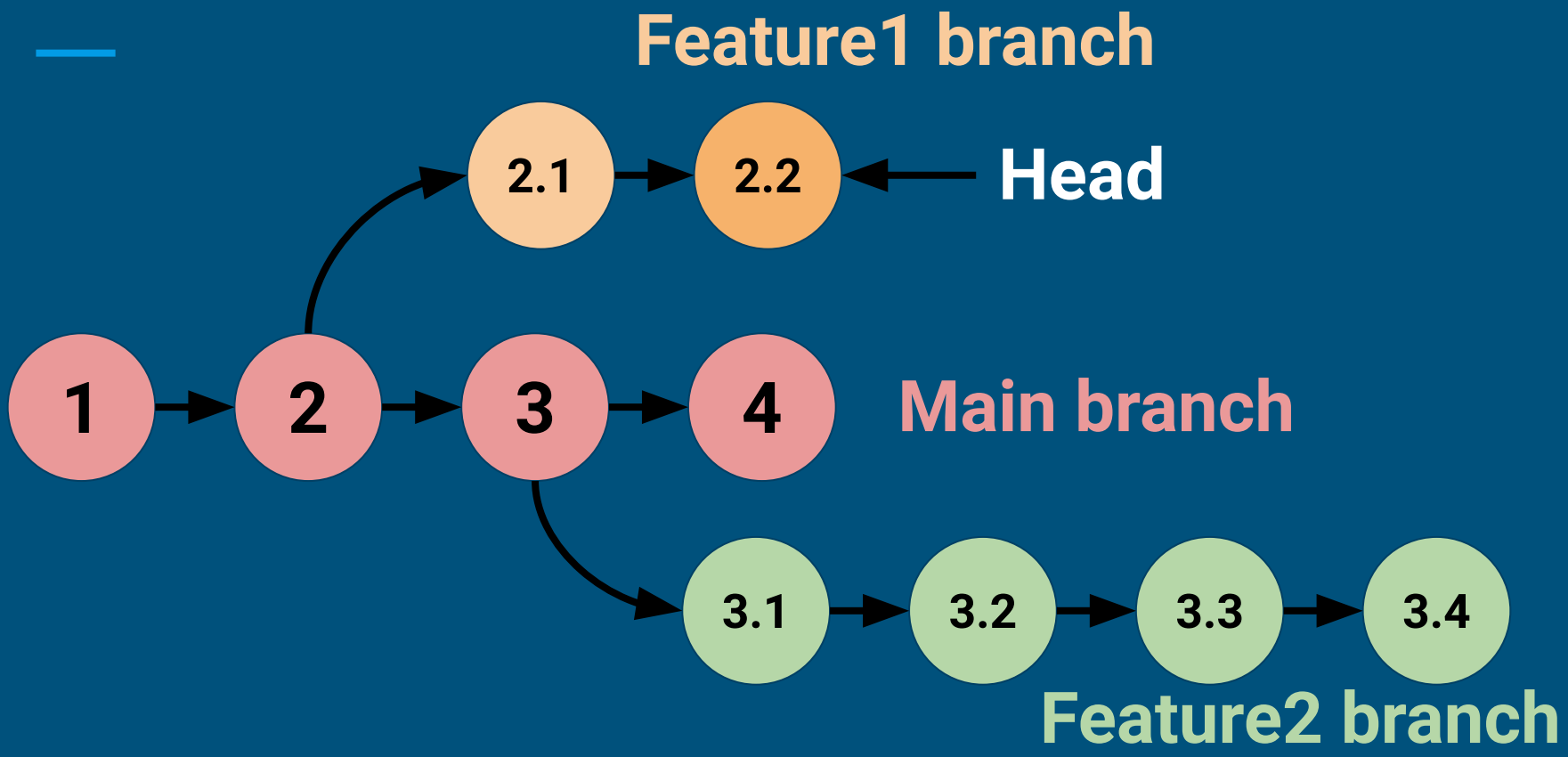
# Git snapshots: commits



# Git snapshots: commits



# Git branches



# Commit

1. Unic hash
2. Author
3. Datetime
4. Commit message

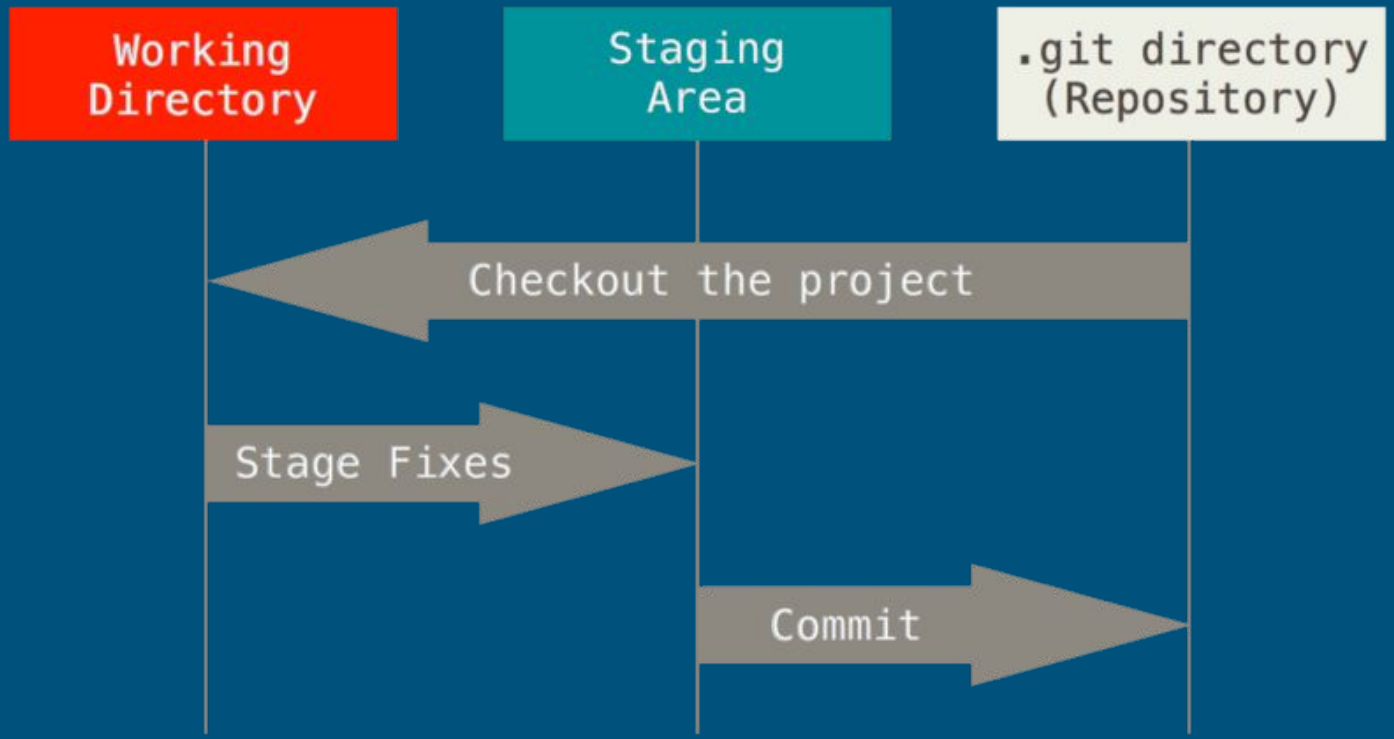
**commit**
9261ccf03b0f4076fc6c533ed1bc030999e38269

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

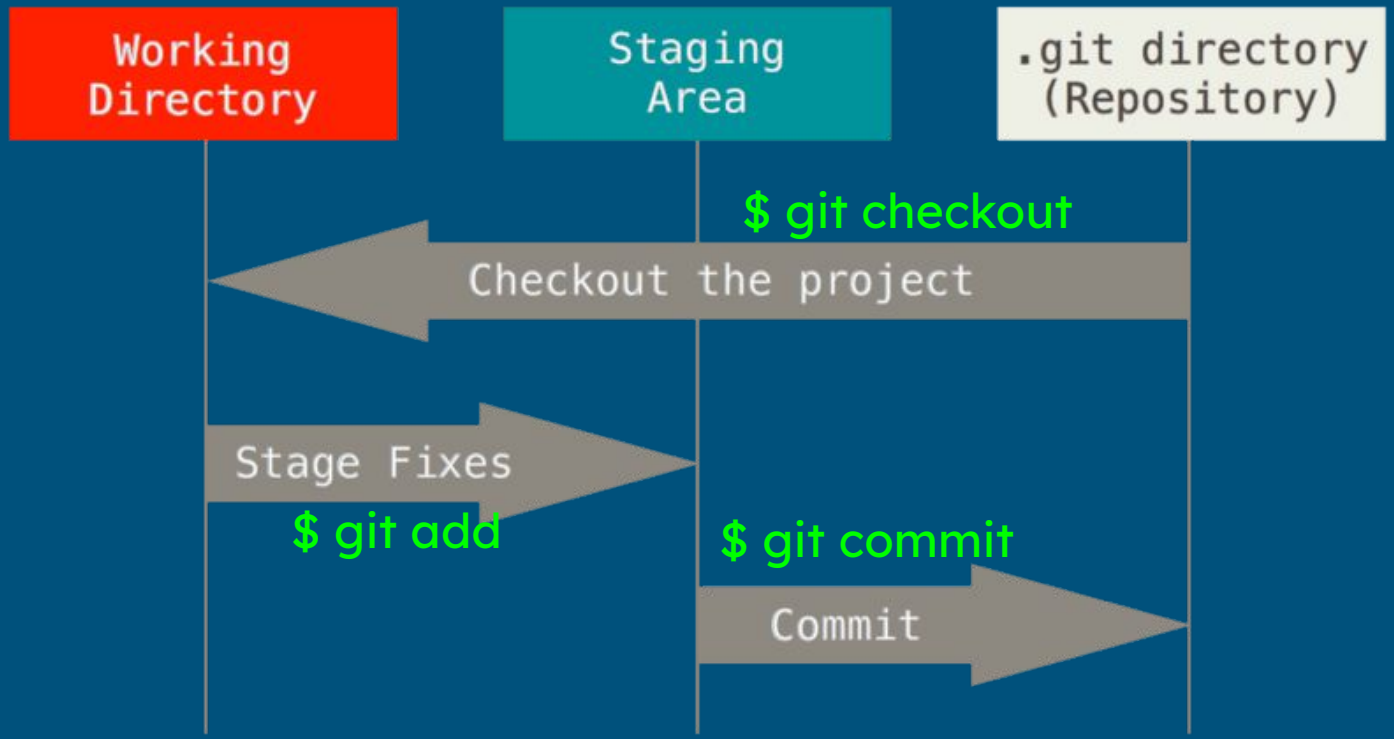
**Date**: Fri Dec 22 10:15:48 2023 +0100

Added feature ...

# Workflow di commit



# Workflow di commit



# Commit workflow

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

The "**git add**" command in Git is used to add changes made to files to the staging area. With "git add", you can select specific changes to include in the next commit.

```
> git add file1 file2 file3

> git add .
```

# Commit workflow

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

The "**git commit**" command is used to register the changes in the staging area to a **new commit** in the Git repository.

```
> git commit -m “Description”

> git commit
```

# Commit workflow

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

The "**git log**" command gives the history of **commits**, hash, authors, datetime included for each commit.

```
> git log
```

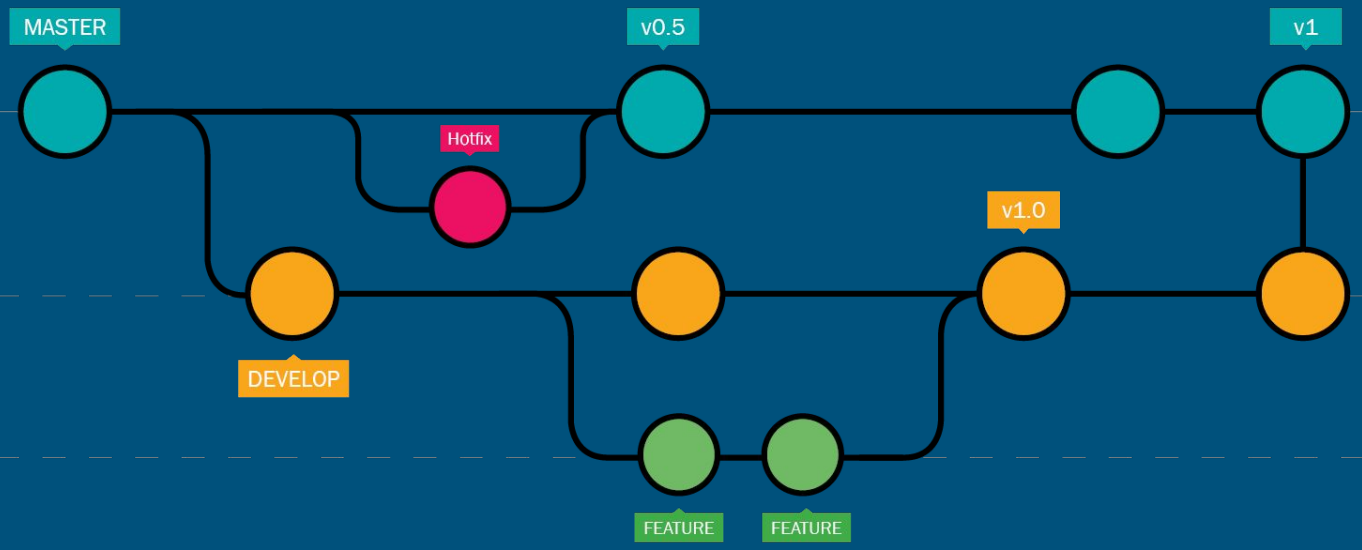
# Commit workflow

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

The "**git status**" command gives information on the current status of the repo. It shows modified files, what is in the staging area ad what is still to track.

```
> git status
```

# Branch workflow



# Branch workflow

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

The "**git branch**" command in Git is used to display the list of branches present in the repository. With the right options, it can be used to link a GitHub branch or delete a local branch.

```
> git branch

> git branch -D branch_name

> git branch --set-upstream ...
```

# Branch workflow

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

The "**git checkout**" command is used to switch between branches (or commits) in Git. It is also used to create a new branch.

```
> git checkout branch_name

> git checkout hash -- file_name

> git checkout -b new_branch
```

# Branch workflow

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

The "**git diff**" command shows the differences between two points in the commit history. It can be used between commits, between Head and the last commit, or between branches.

```
> git diff branch1 branch2

> git diff

> git diff hash1 hash2
```

# Branch workflow

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

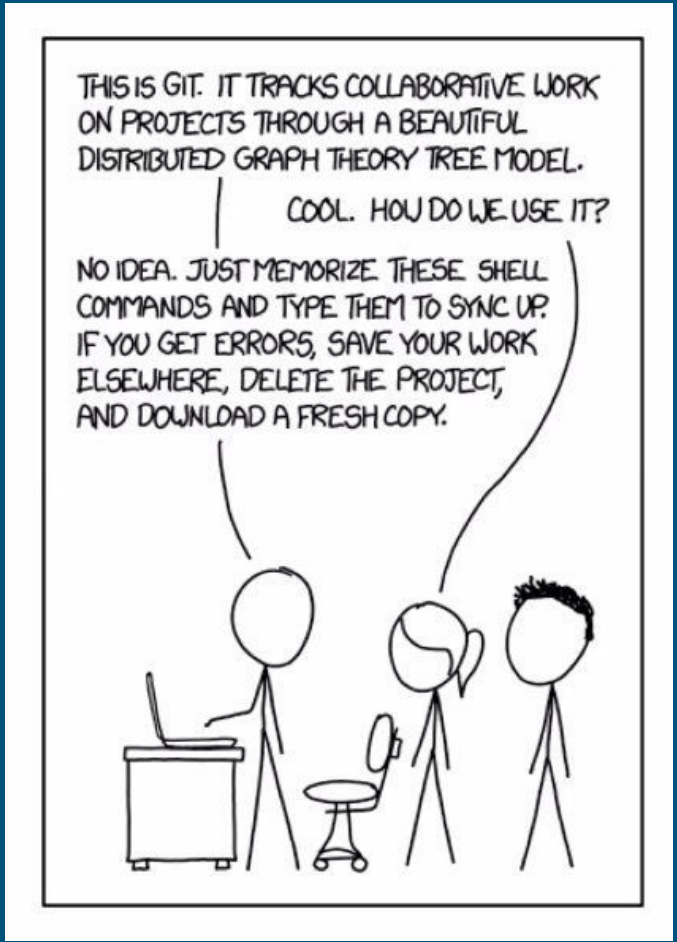
The "**git merge**" command is used to combine changes from one branch to another.

> 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
```

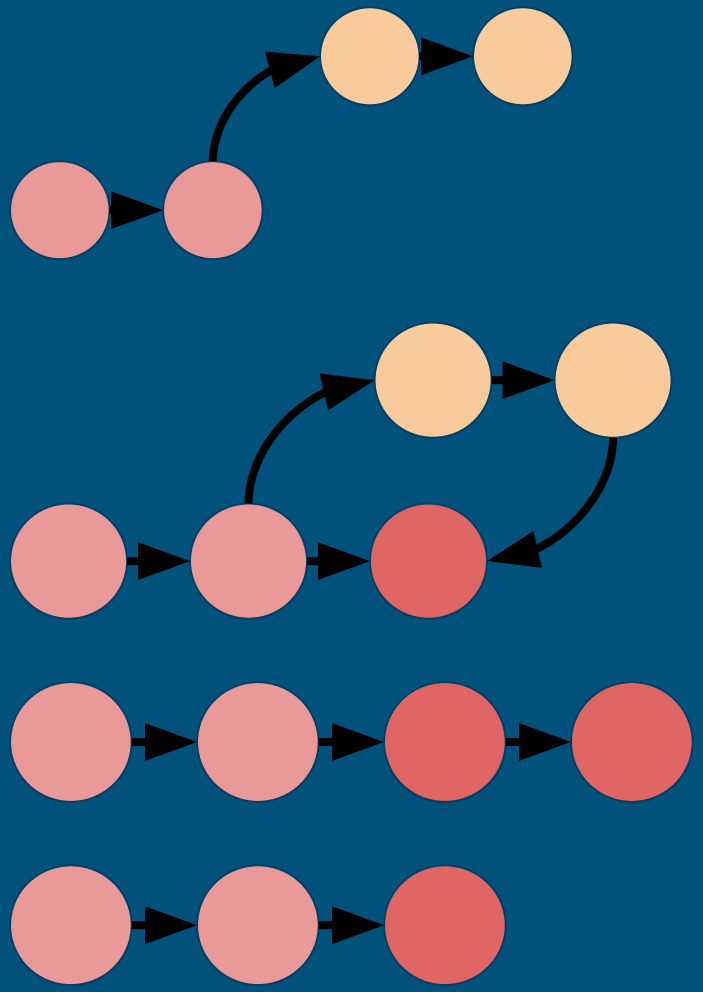
# Exercises: batch 1!

*https://github.com/rodolfocarobene/git_lecture*

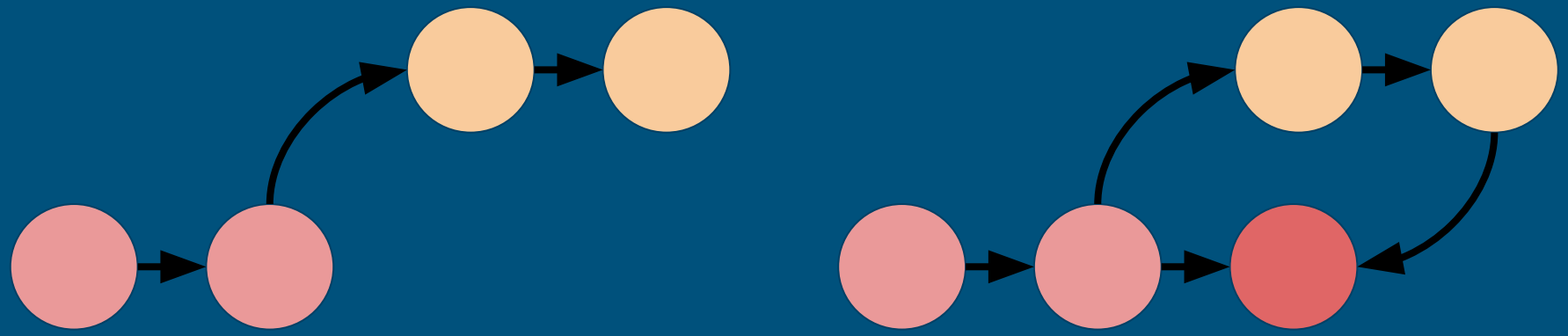
Configure Git, initialize a repository, add the main files of your project. Use various branches to add features. Merge them with merge and resolve conflicts!

# Merge strategies:

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

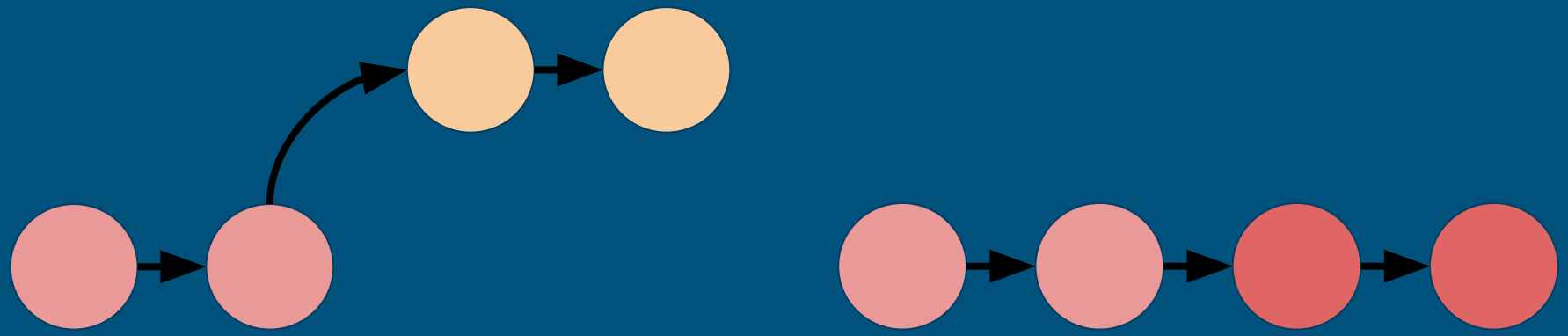


# Merge strategies: git merge



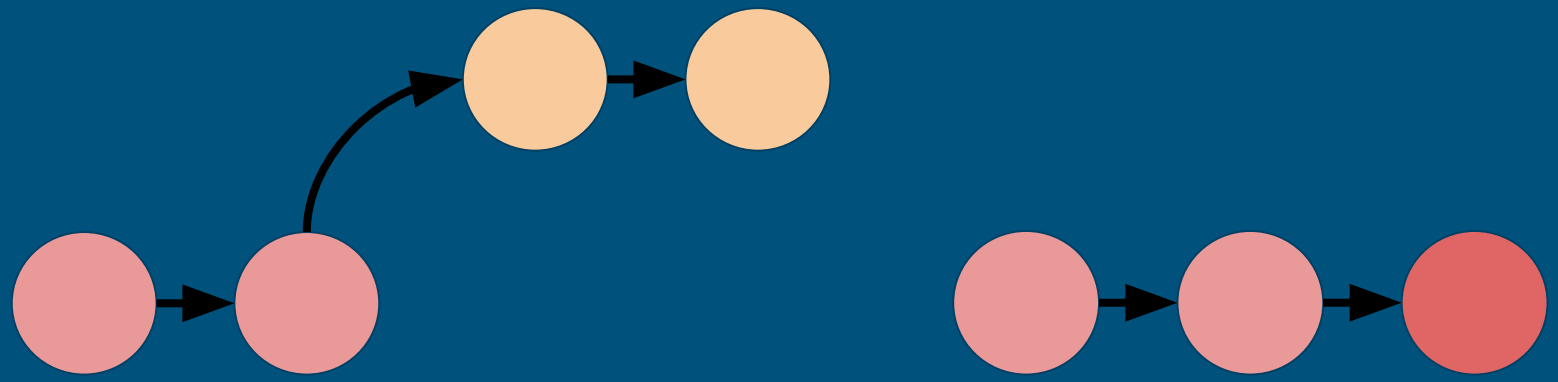
New merge commit that references the two "parent" branches.

# Merge strategies: git rebase



All the commits from the feature branch are moved to the main branch.

# Merge strategies: git merge --squash



All the commits from the feature branch are condensed into a single commit and added to the main branch.

# More commands

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

The "**git stash**" command allows you to make temporary backups of changes in the working directory.

> 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

The "**git restore**" command allows you to discard changes in the working directory.

```
> git restore .

> git restore --staged .

> git restore --staged --worktree .
```

# More commands

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

The "**git blame**" command displays authors and commits for each line of a file. It is often integrated into your IDE for a more seamless experience.

```
> 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

The "**git cherrypick**" command allows you to merge a single commit (from another branch) into your current working branch.

```
> 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

The "**git rm**" command is used to stop tracking a specific file in version control. Caution! This does not delete the file from the version 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

The "**git bisect**" command can be used to identify the commit that introduced a specific issue.

```
> 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 cherrypick
5. git rm
6. git bisect
7. git tag

The "**git tag**" command can be used to give a name (e.g., v0.1.0) to a specific point in the 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

The ".**gitignore**" file is used to specify which files and folders should not be tracked by Git.

# Special git files

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

The ".**gitkeep**" file is used to add empty folders to a repository. Its name is a convention; it is not a real Git file.

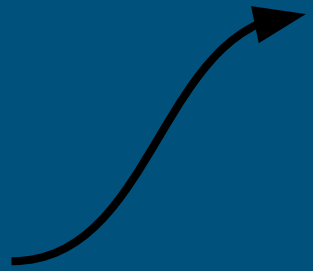
# Special git files

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

The ".**gitmodules**" file is used to include and manage other modules within the repository. Submodules are, in essence, independent Git repositories but are included in the main project.

# Special git files

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



The ".**gitattributes**" file is used to assign "attributes" to specific types (extensions) of files. It can control merge strategies for each file type or indicate what to save and what not to save.

# Special git files

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

The ".**gitconfig**" file contains the global/local preferences that Git uses. For example, it includes the main author's name/email, preferred editor, initial branch name, and aliases.

# Exercises: batch 2!

Add the main files (.gitignore) to your project. Experiment with the just-defined commands.

# 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

- Automatic scripts triggered by Git actions (examples: pre-commit, post-commit, pre-push...)
- Some examples are automatically generated in the ./git/hooks/ folder.
- By default, they use Bash Shell commands, but you can use any scripting language (e.g., Python).
- Completely local!

# 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!

Install pre-commit. Add important pre-commit hooks for your project. Define useful aliases. Make a commit and undo it (in multiple ways, if possible!).

# Break !

# What is GitHub?

# What is GitHub?

GitHub is a hosting platform based on Git.

GitHub provides a centralized platform where developers can host their Git repositories, collaborate on source code, manage issues, plan projects, and much more. It is widely used both in the corporate sector and in the open-source community.

# Main git commands

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

The "**git remote**" command is used to manage a remote repository. For example, to add or remove a remote repository.

```
> git remote -v

> git remote add origin
  <my_repo_URL>
```

# Main git commands

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

The "**git push**" command is used to send local commits to a remote repository.

```
> git push

> git push --force

> git push --all --tags
```

# Main git commands

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

The "**git fetch**" command is used to download remote commits into the local repository.

```
> git fetch

> git fetch --all
```

# Main git commands

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

The "**git pull**" command is used to download and merge remote commits into the local repository.

```
> git pull

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

# Main git commands

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

The "**git clone**" command is used to locally copy a remote repository.

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

# Git Pull --force ?


```
example@system:
~Documents/test_git
> git fetch

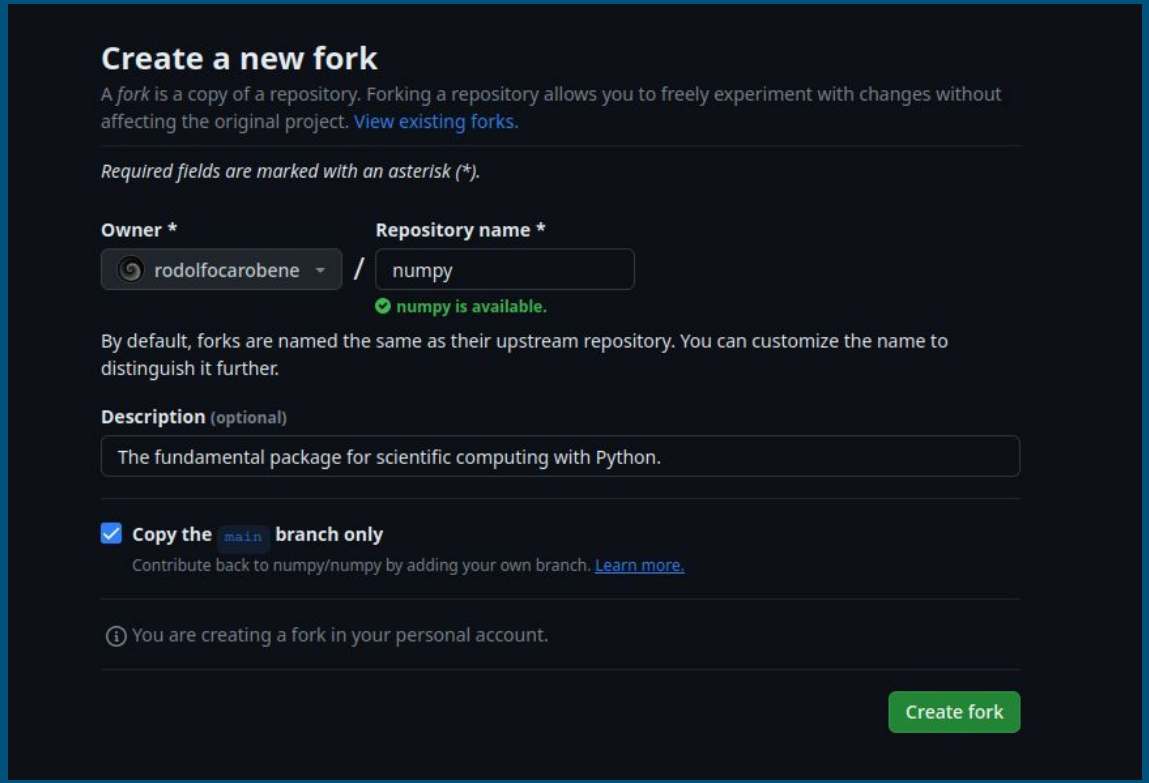
> git branch backup_current

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

# Exercises: batch 4!

Let's resume the previous project and put it on GitHub! Let's create the repository.

# GitHub forks



# 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 __array_ufunc_ ✕
#25510 opened 12 hours ago by zachbrugh updated 12 hours ago
00 - Bug

API: adjust nD fft `s` 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 `allclose`, revise "Notes" in `isclose` ✕ — 1
#24407 opened on Aug 13 by OverLordGoldDragon updated yesterday
04 - Documentation

DOC: Improve `np.mean` 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

Releases | Tags

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

# GitHub actions: .github/workflows/action.yml

```yaml
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

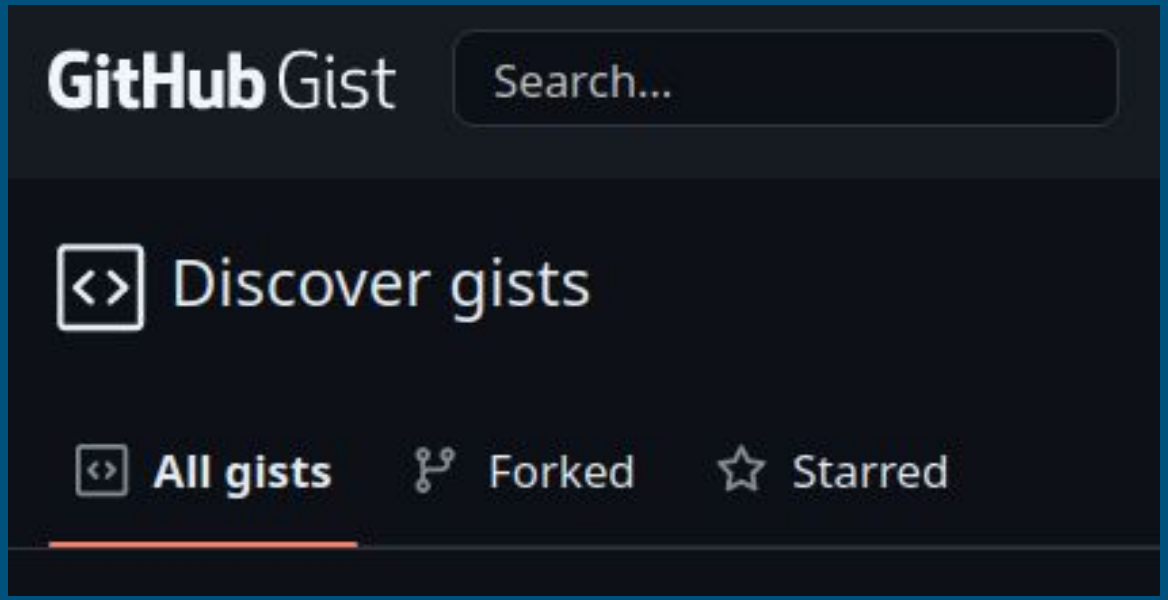
# Exercises: batch 5!

Example of a Python project with actions and reviews. Typical collaboration workflow. Examples of open-source projects on GitHub.

# GitHub.io


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