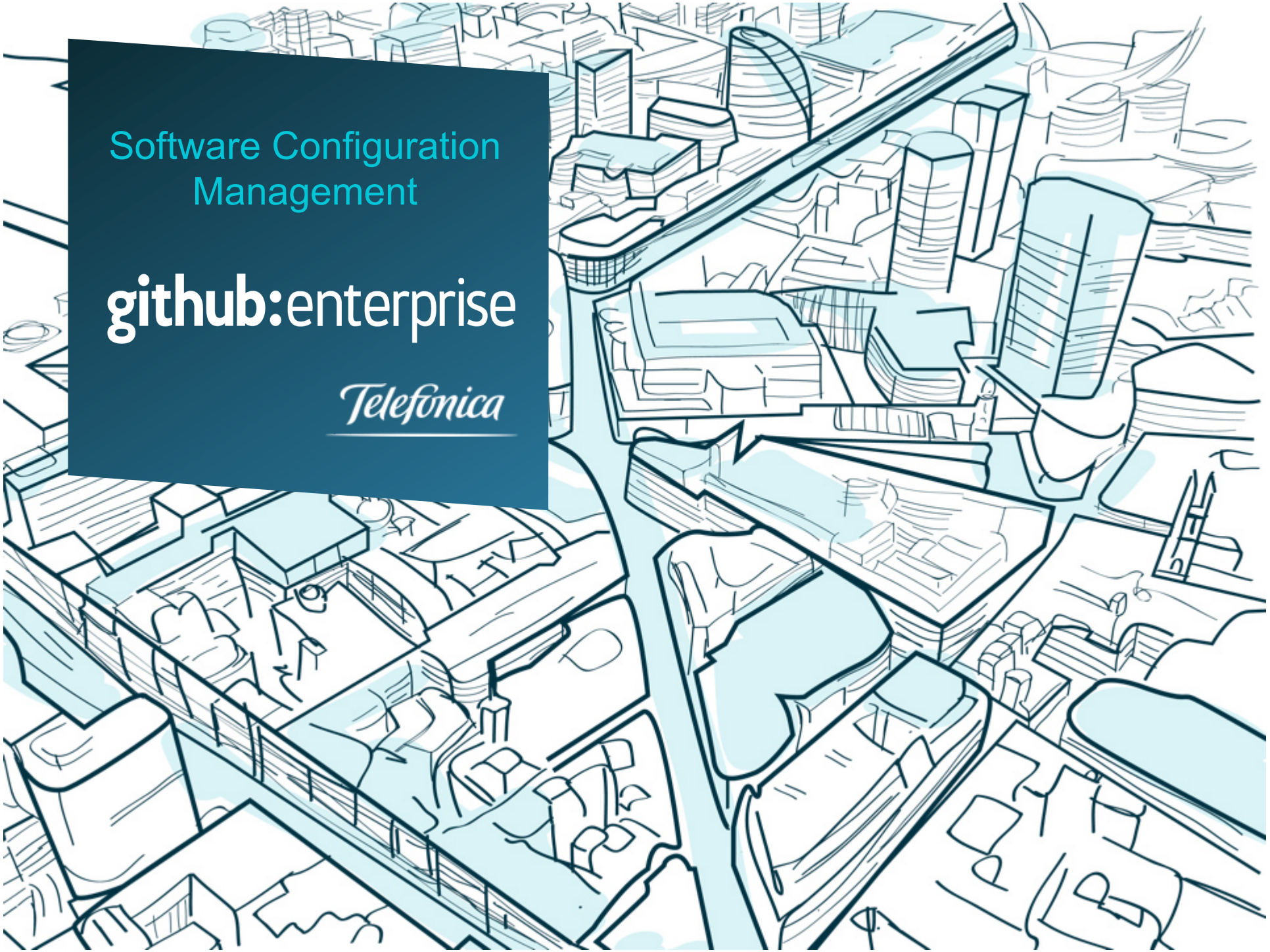
Software Configuration Management

github:enterprise

Telefonica

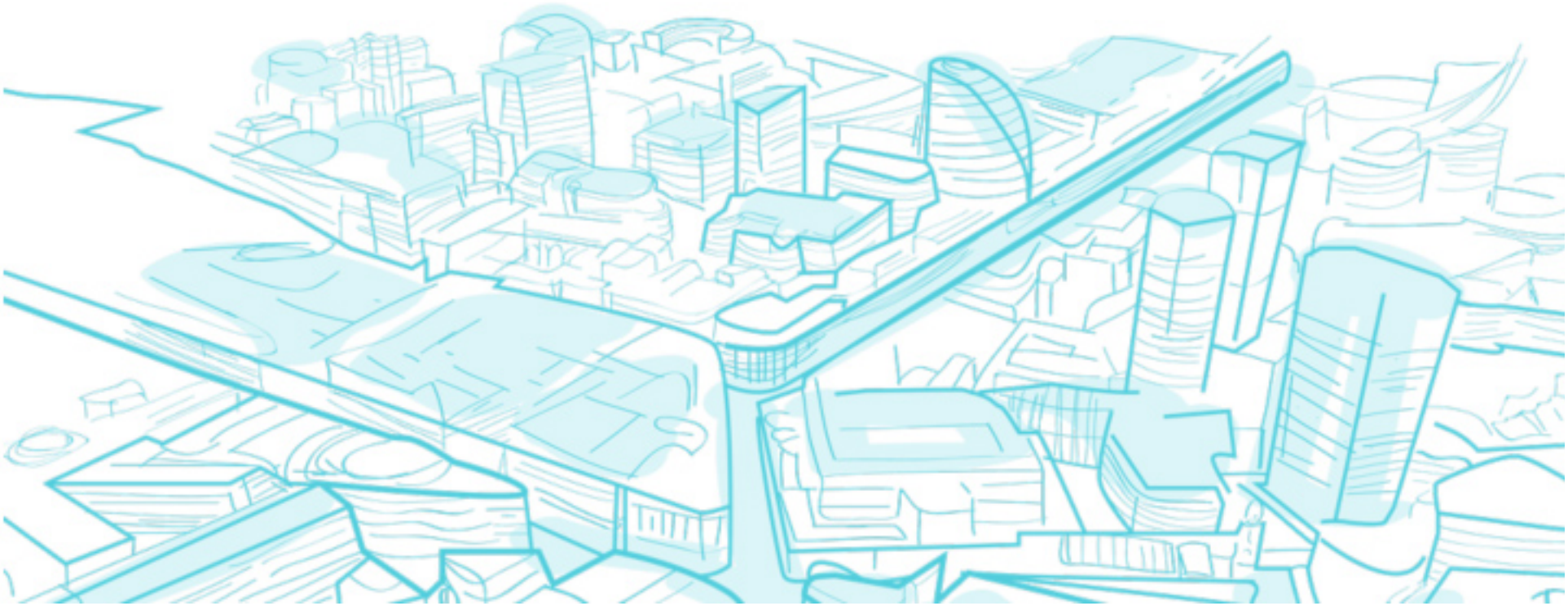


Index

Telefónica

01

Introduction



Presentation

Borja Martín Fernández:

- GIT-Coach
- PDIHUB/GitHub.com Administrator

SCM Main Goals

- Define conventions & policies managing initiatives in PDIHUB
- Establish procedures to manage repositories
- Establish contexts and development branches
- Define responsibilities and roles.

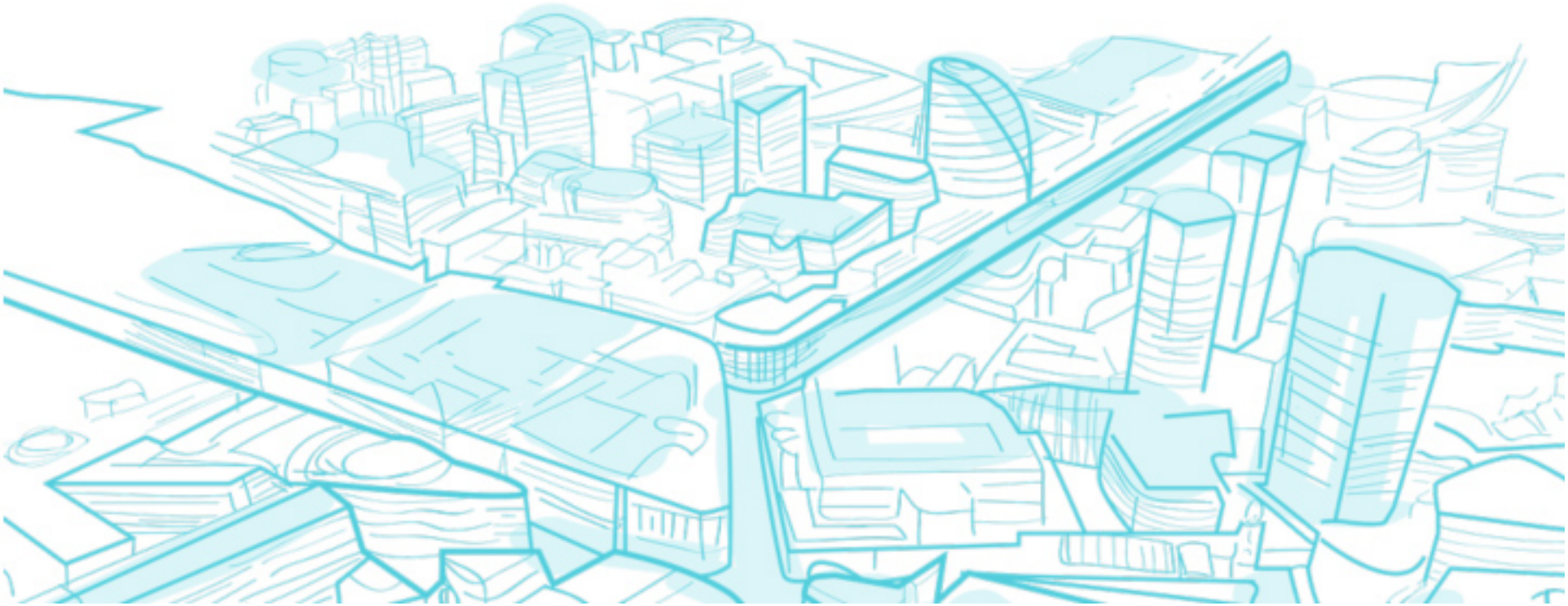
PDIHUB: GitHub Enterprise License

Social Coding

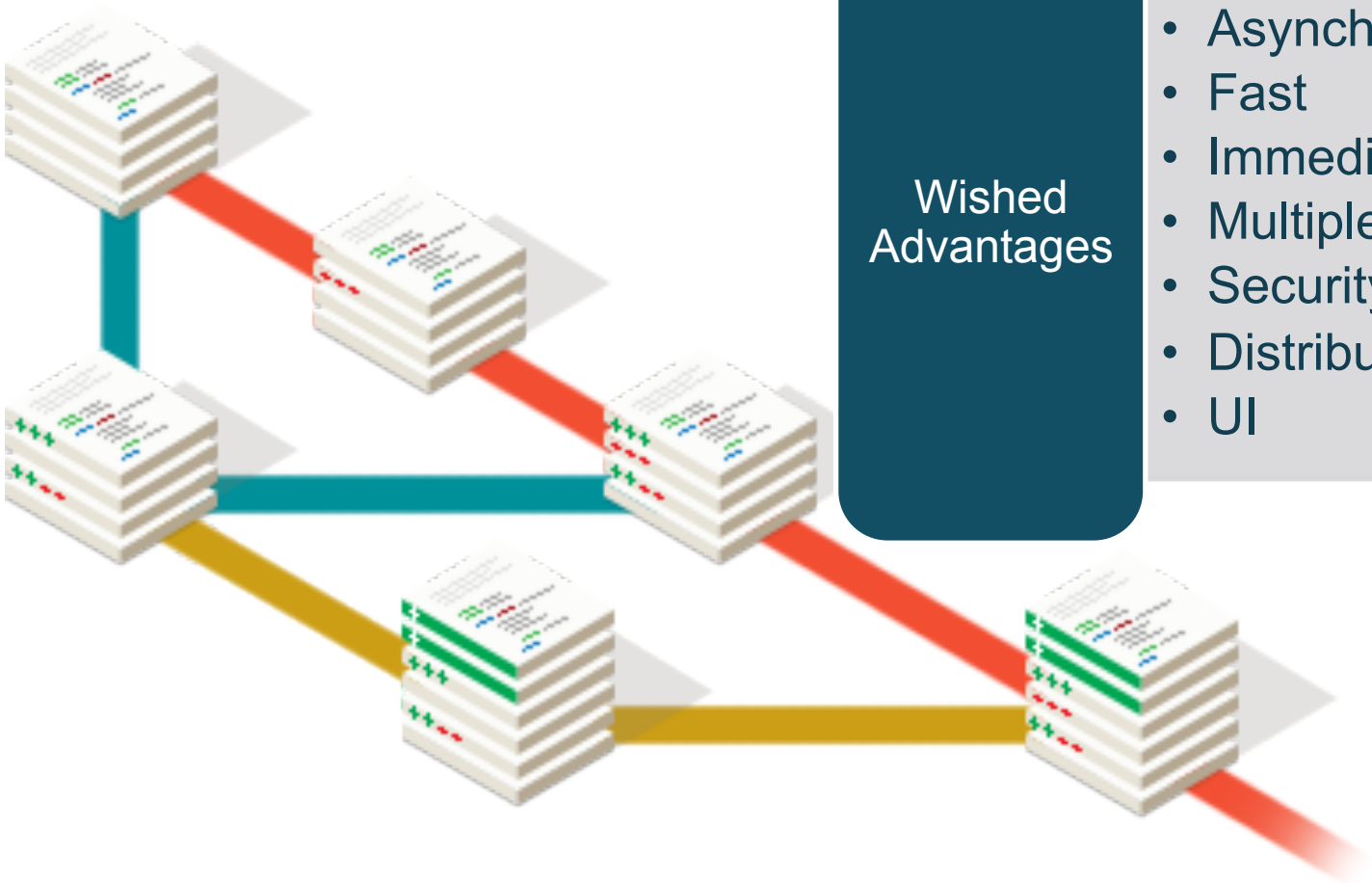
https://pdihub.hi.inet

02

Why GIT?



How to improve version control systems?



Wished Advantages

- Asynchrony
- Fast
- Immediate
- Multiple contexts
- Security
- Distribution
- UI

Basic unit: commit

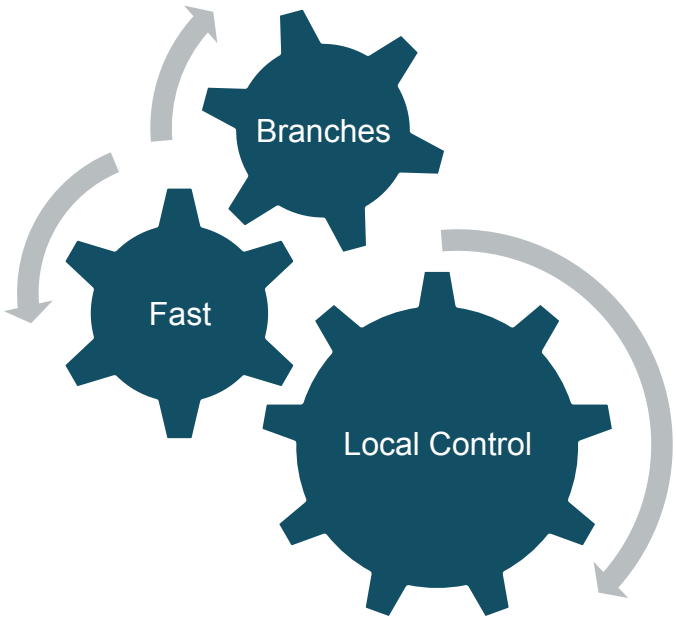
2 apps/sms/js/utils.js View file @ 4b936f1

```
@@ -98,8 +98,8 @@ var Utils = {
   },

   getPhoneDetails: function ut_getPhoneDetails(number, callback) {
-    var details = {};
     ContactDataManager.getContactData(number, function gotContact(contact) {
+      var details = {};
       //TODO what if return multiple contacts?
       if (contact.length > 0) { // we have a contact
         var name = contact[0].name,
```

Main properties

- Git stores added/removed code lines
- Snapshots are ordered
- Each snapshot is a commit

Advantages



Distributed

- Small file system
- Simple file formats
- Backup clones
- Cheaper rollback
- [forks]: Access control

Branching

- Minor repository size (30x)
- Better merging revision
- Context vs trunk
- Integration workflow

Social Advantage

Trust Networking

- Context mails
- Binary packages
- Release freezing
- Scalable
- Multi stage
- Assign issues

Collaboration

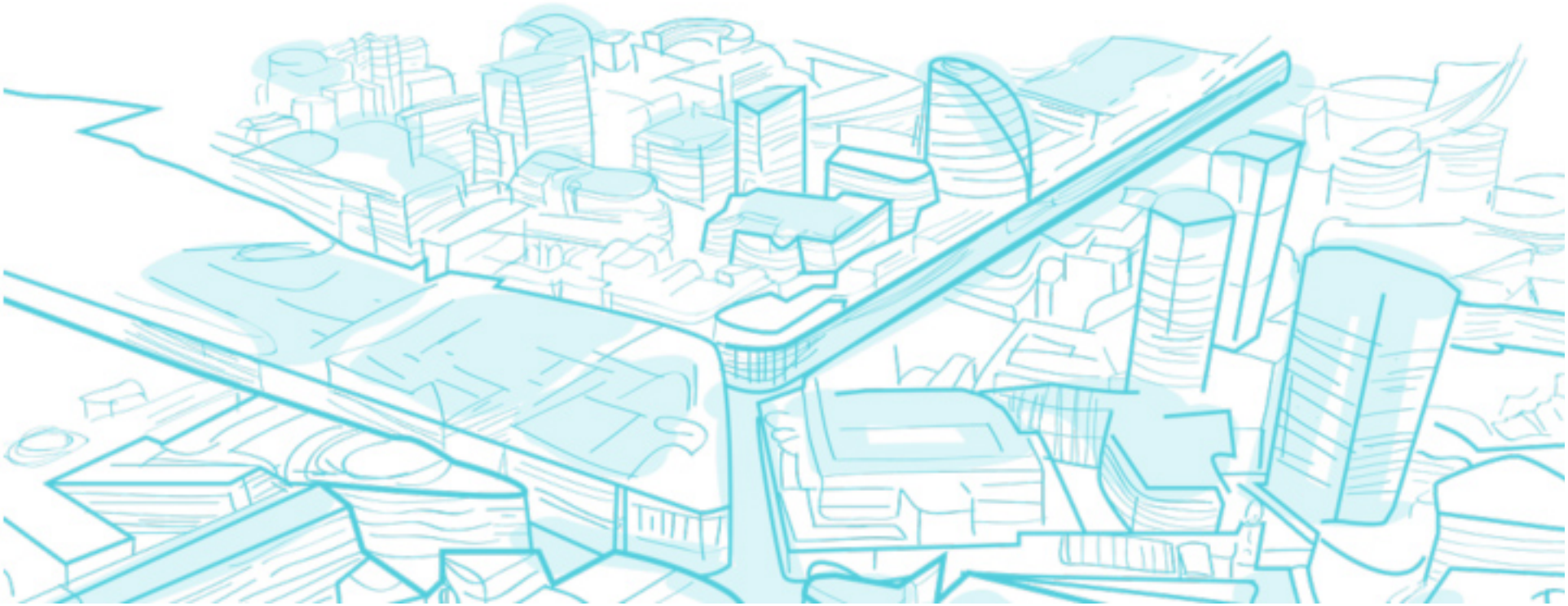
- Featuring
- External bugfixing
- Adhocracy
- Dynamic resources
- Manifests

Why GitHub?

- Pull Requests
- Social Coding
- Collaboration
- Forks
- Organizations
- Wikis

03

My workspace



Create a local repository

mkdir

- Create new directory

```
$ mkdir Madrid && cd Madrid
```

Init

- Convert directory into repository

```
$ git init .

Initialized empty Git repository in
…/Madrid/.git/
```

Directory vs Repository

Difference between a repository and a directory?

Where does git store changes history?

What does .git contain?

How to output changes log

```
$ ls -la

total 4
3 Borja 0    Sep 13 22:26 .
3 Borja 4096 Sep 13 22:26 ..
1 Borja 4096 Sep 13 22:26 .git
```

Review

- Git log

  Shows history changes

- Git(k)

  User Interface

```
$ git log
commit
fc52e47c3908a10b85d7e1a8eAuthor:
Borja Martin Fernandez
<bmartinf@axpe.com>
Date:   Fri Sep 14 00:48:29 2012
    First commit
```

Git Status/Checkout

- Git status

  Show the status, current branch and staging area

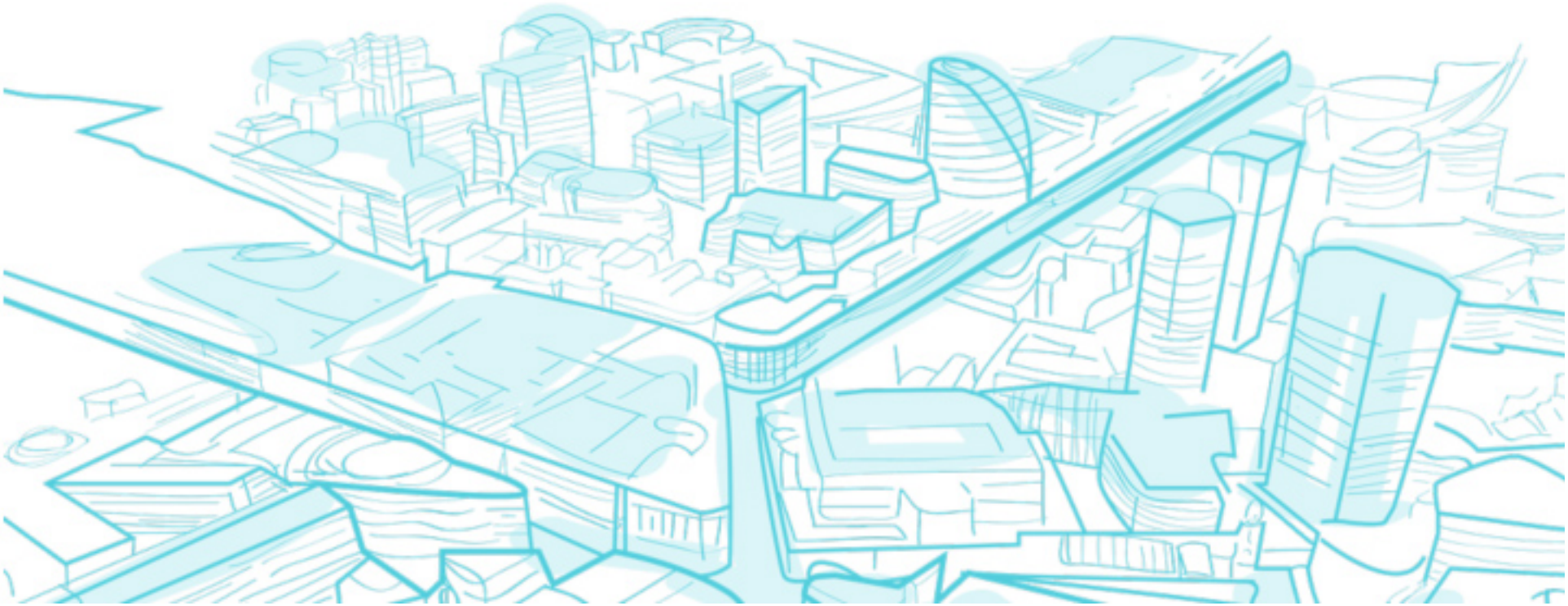
- Git checkout

  It will output the target snapshot in the current directory.

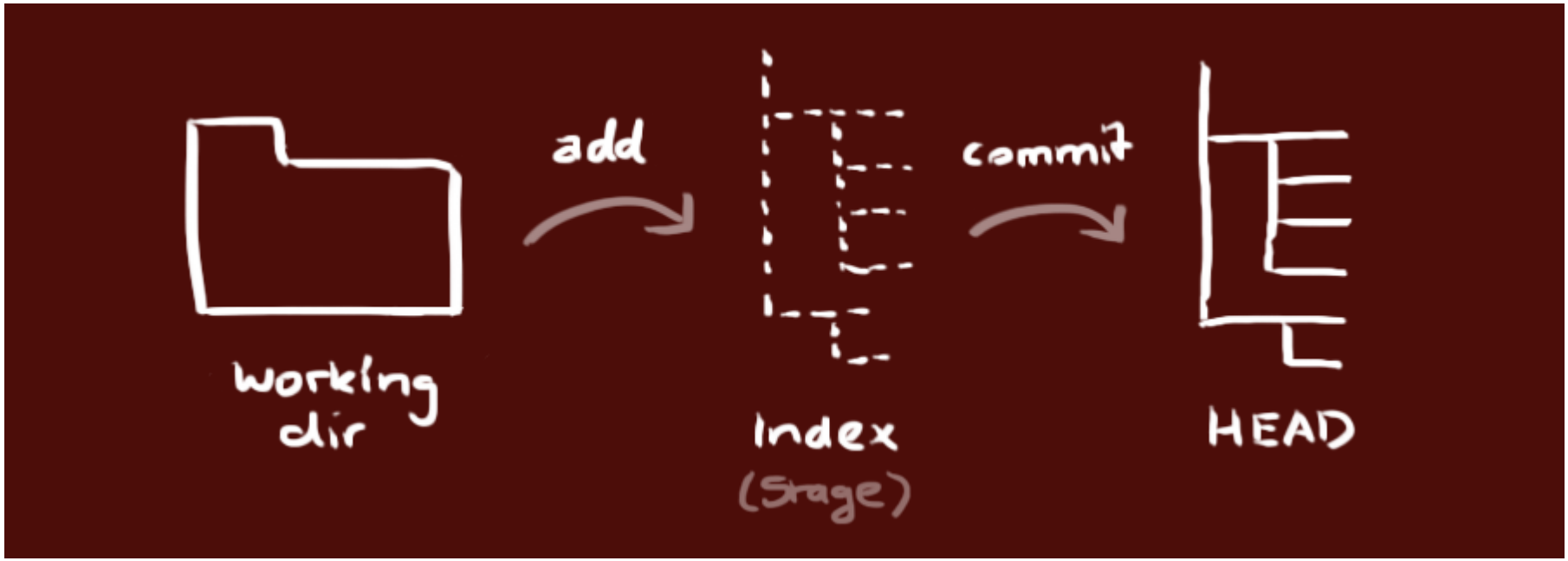
```
$ git status

# On branch master

nothing to commit
(working directory clean)



$ git checkout 311b98
```

04

My first contribution



Create local commit



Git ADD / commit

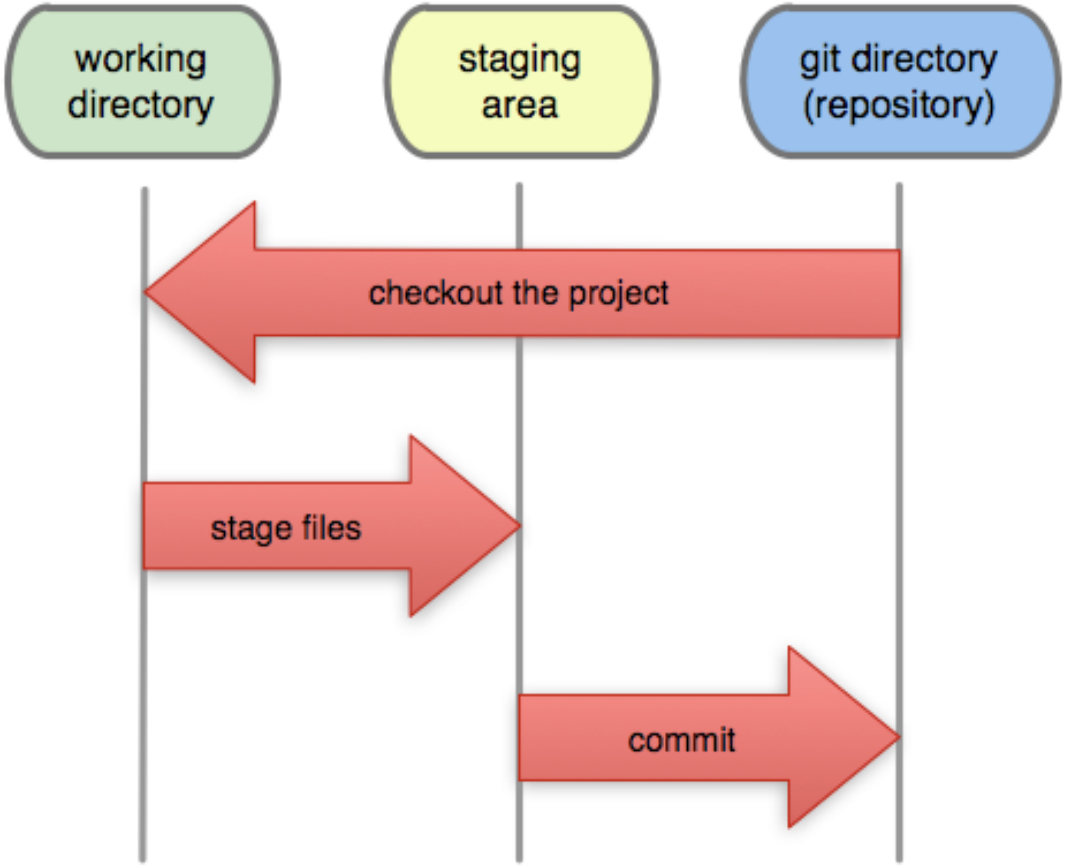
Edit
- Changes are realized in directory

Add
- Changes are added to staging area

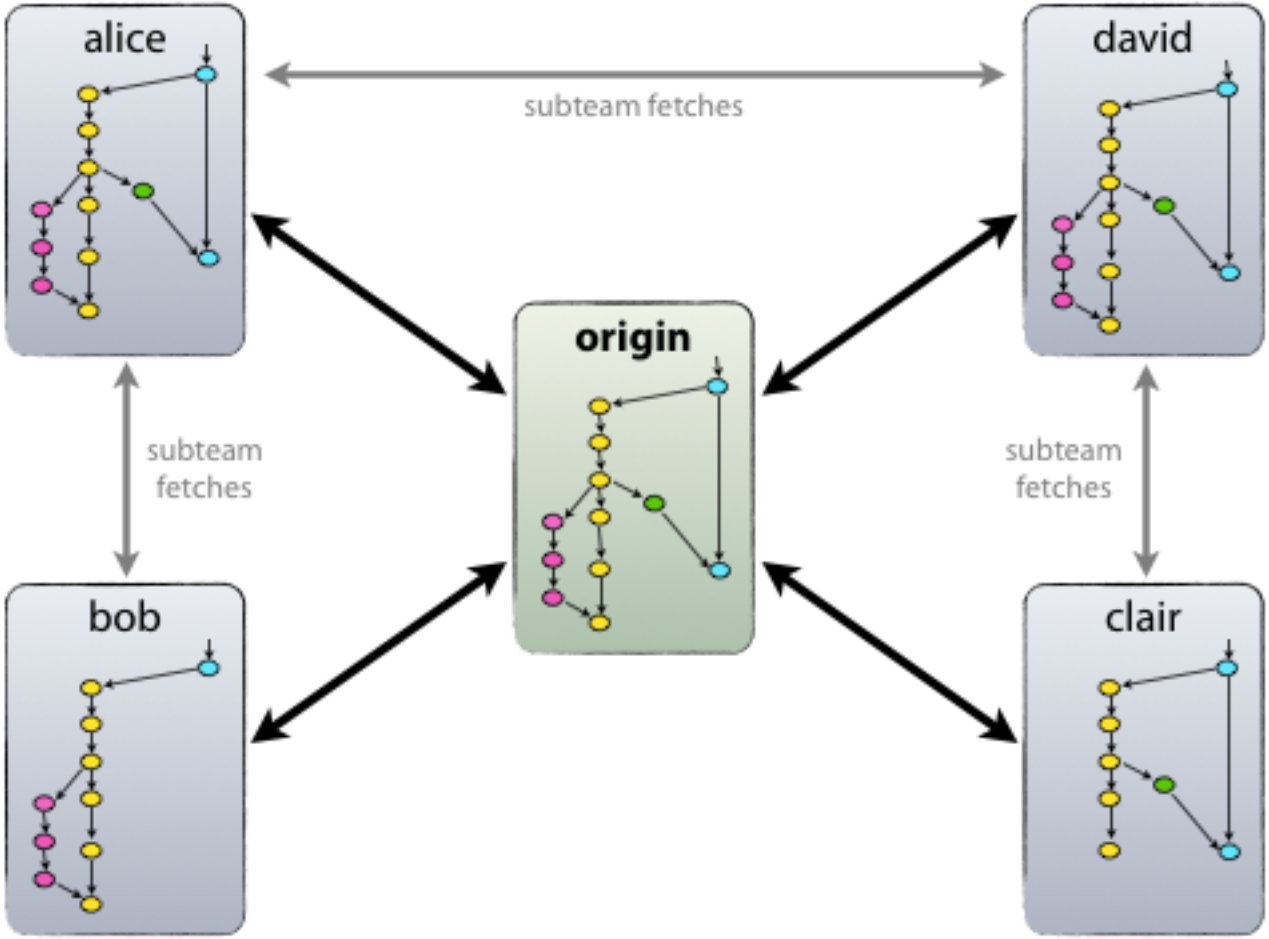
Commit
- Pack changes in a commit object

```
$ touch foo.txt

#       untracked files:   foo.txt


$ git add foo.txt

#       new file:   foo.txt


$ git commit –m “Add foo test-file”

# 1 file changed, +1 insertion(+)
```

Local areas



How to publish my contribution?



Configuration: name & email

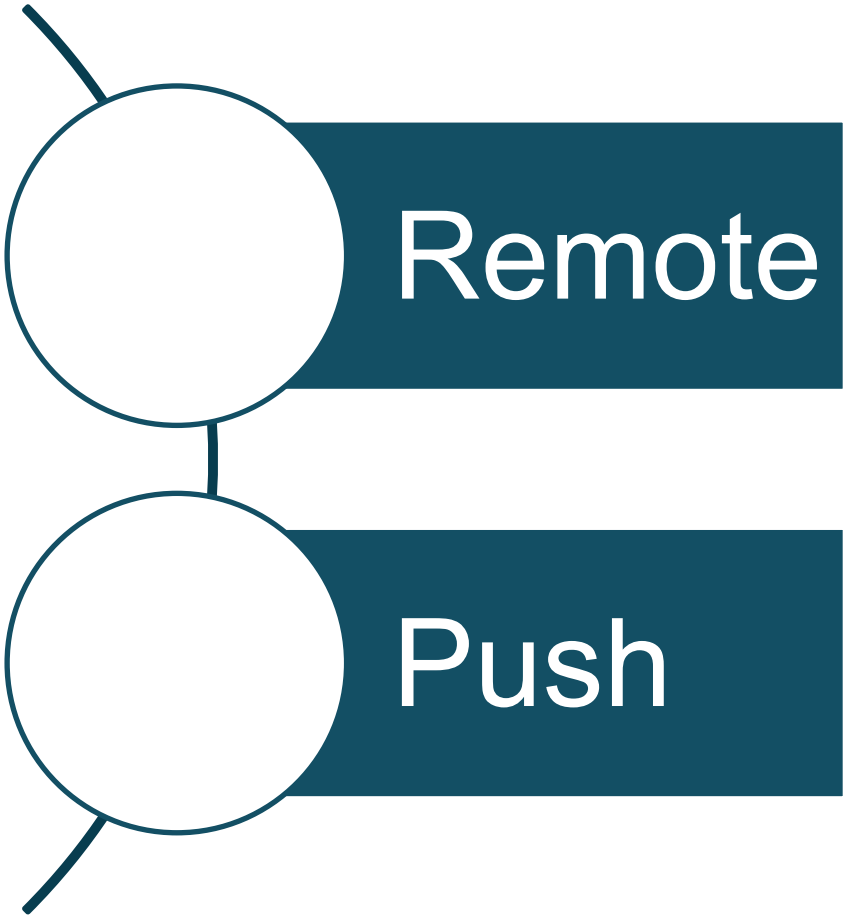
Set username

```
$ git config --global user.name
“Borja Martín Fernández “
```

Set email

```
$ git config --global user.email
“bmartinf@axpe.com"
```

Configuration: remote repository




```
$ git remote add origin https://
github.com/user/repo.git

# Establish “origin” as the name of
the remote repository.
```

```
$ git push origin master

# Push into “origin” (remote repo)
the commits in the “master” branch.
```

Git pull & git push

Pull

- Fetch & update remote changes

Edit

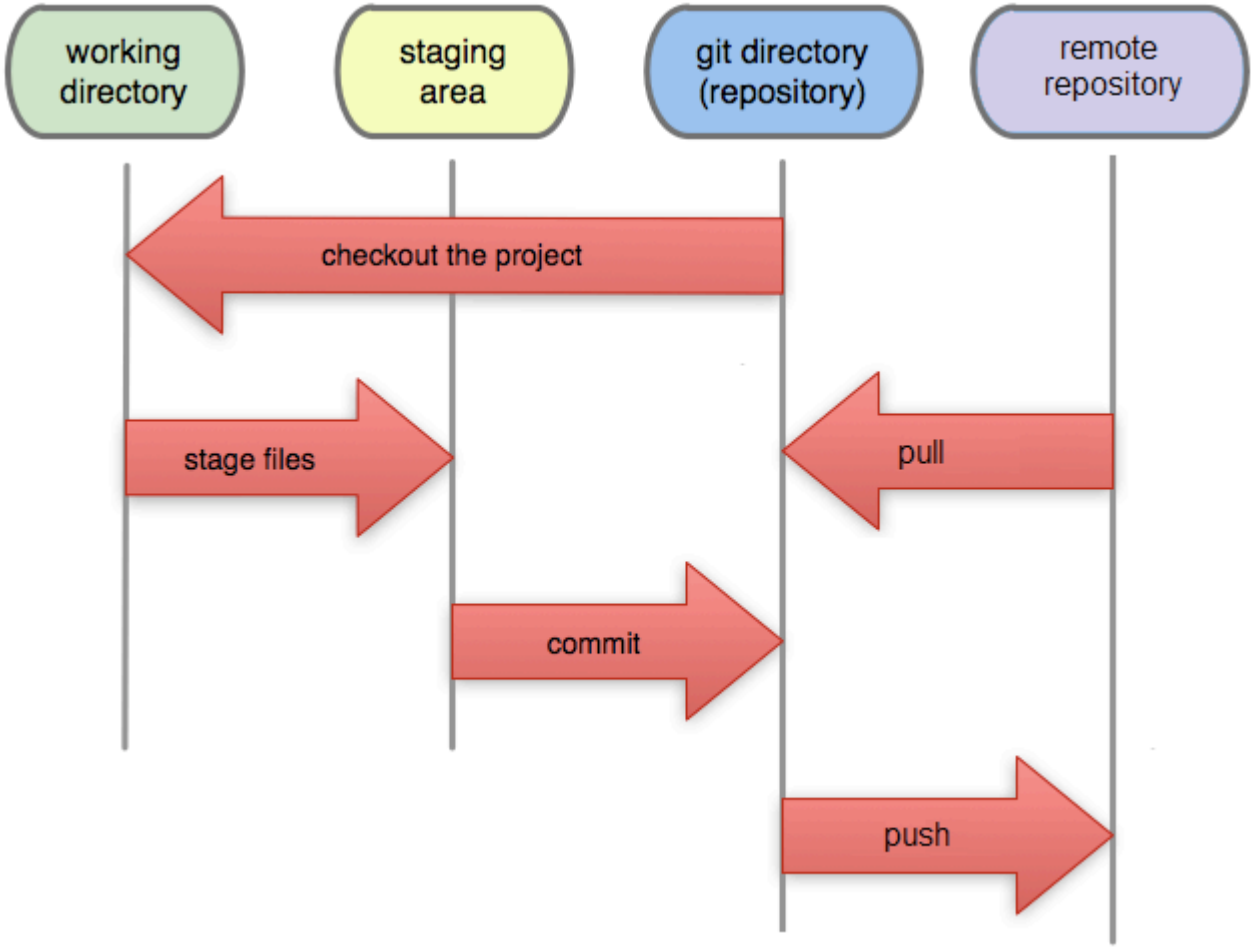
- Work in current directory

Push

- Update remote repository

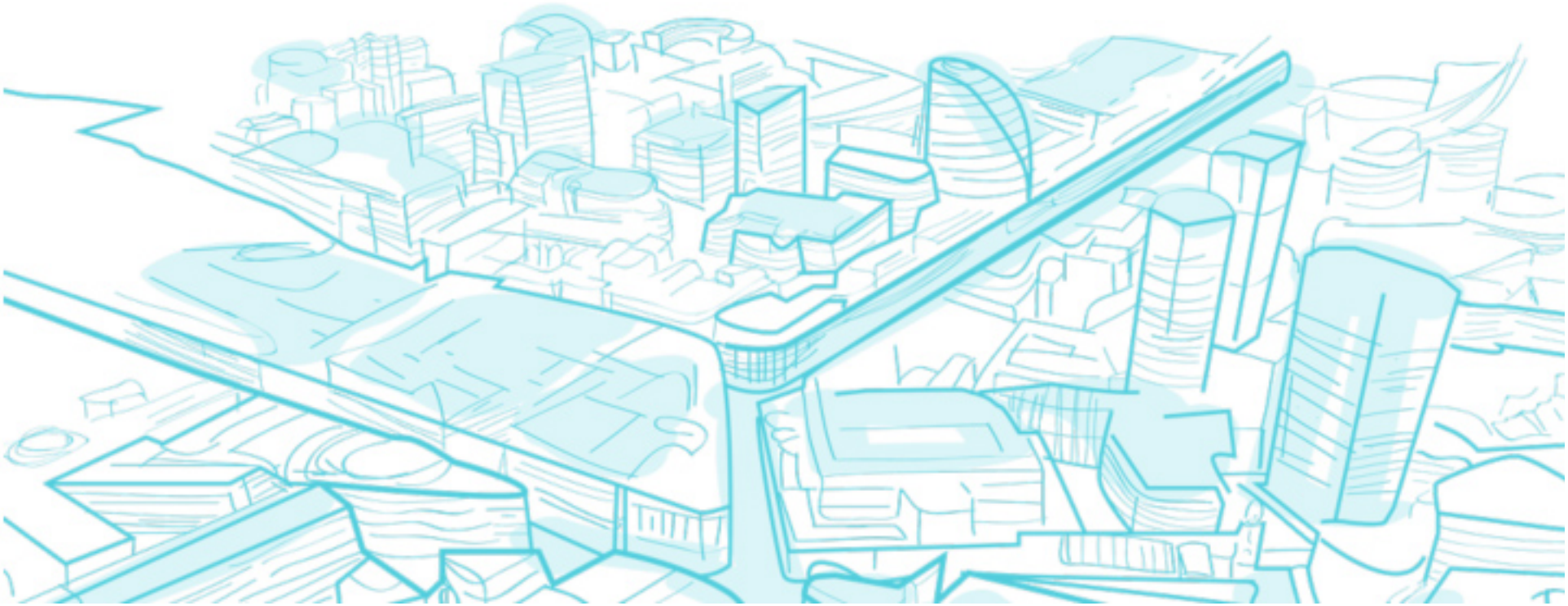
```
$ git pull

Already up-to-date.


$ git add foo.txt

#       new file:   foo.txt


$ git push      # origin
  Counting objects: 25, done.
  Delta compression using up to 2 threads.
  Compressing objects: 100% (25/25), done.
  Writing objects: 100% (25/25), 2.43 KiB, done.
  Total 25 (delta 4), reused 0 (delta 0)
  To git@pdihub.hi.inet:smp/test.git
     * [new branch]      master -> master
```

Working areas: Remote repository (PDIHUB)

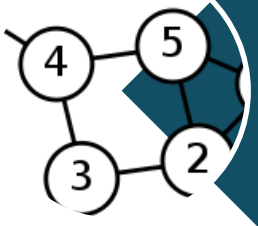


05

Contexts: branches

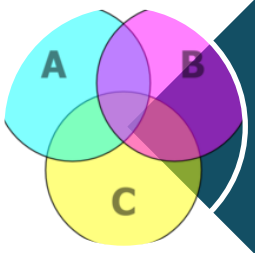


Multiple contexts solution: branching



Graph

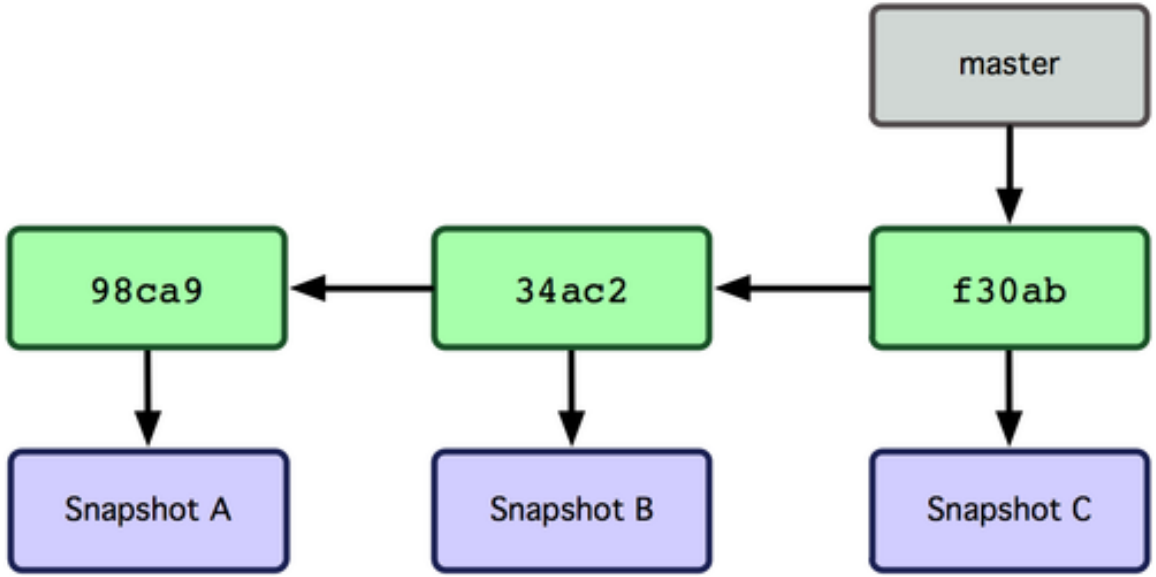
- Multiple sucesors: Branching
- Multiple antecessors: Merge



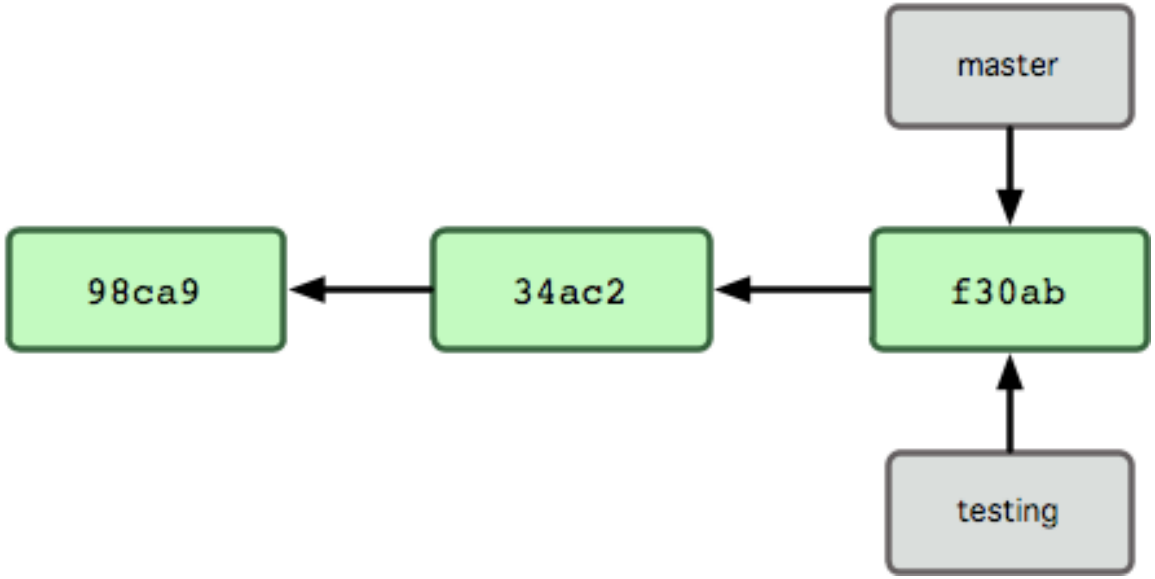
Contexts

- Construction: Features
- Delivery: Bugfixing & Release

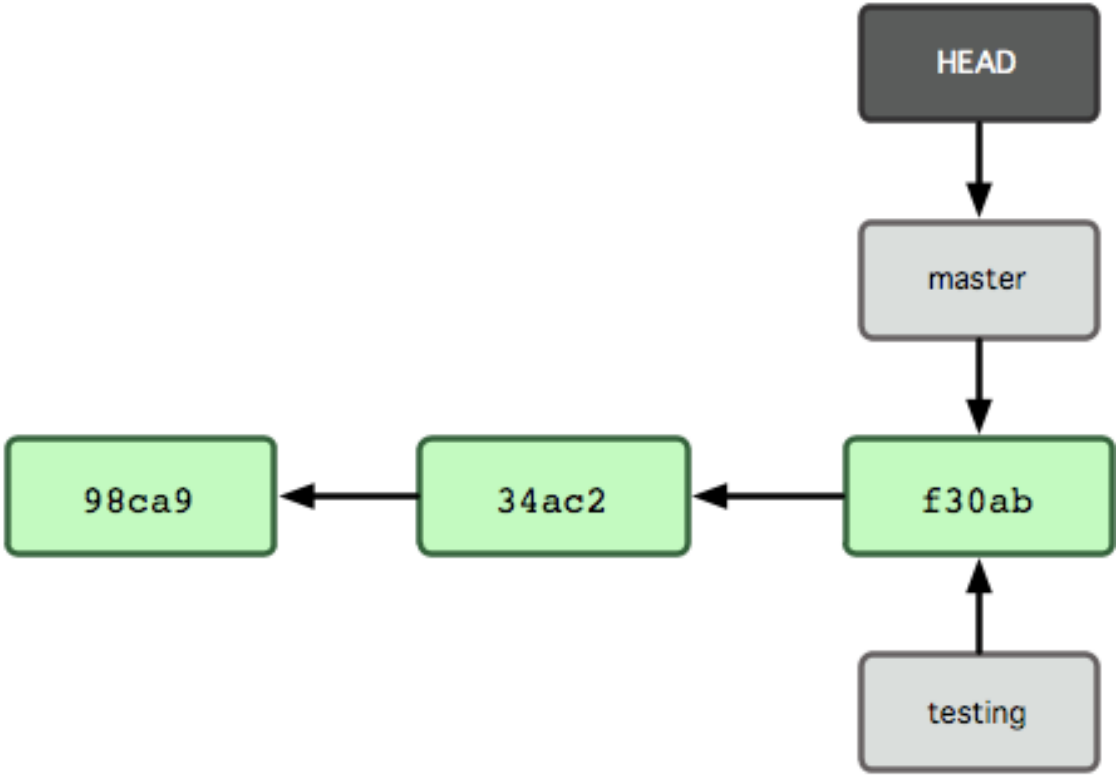
Branch definition: pointer



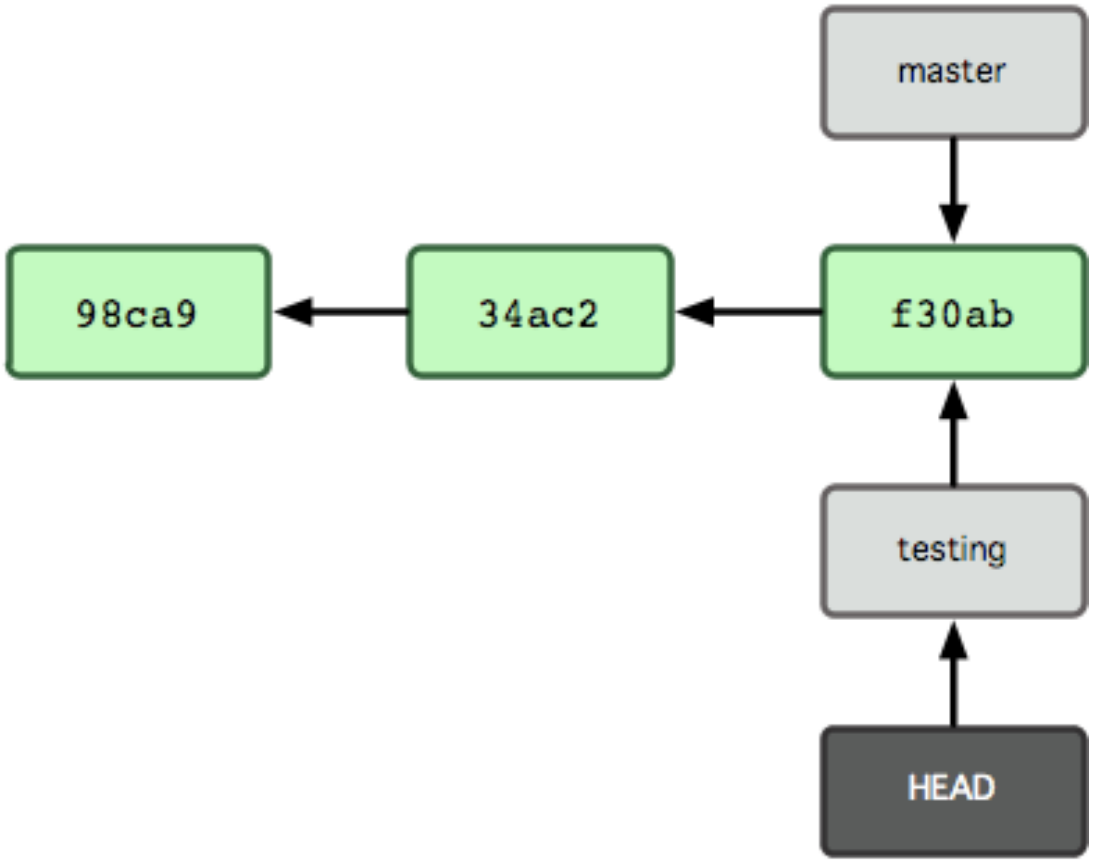
Branch definition: Create new branch



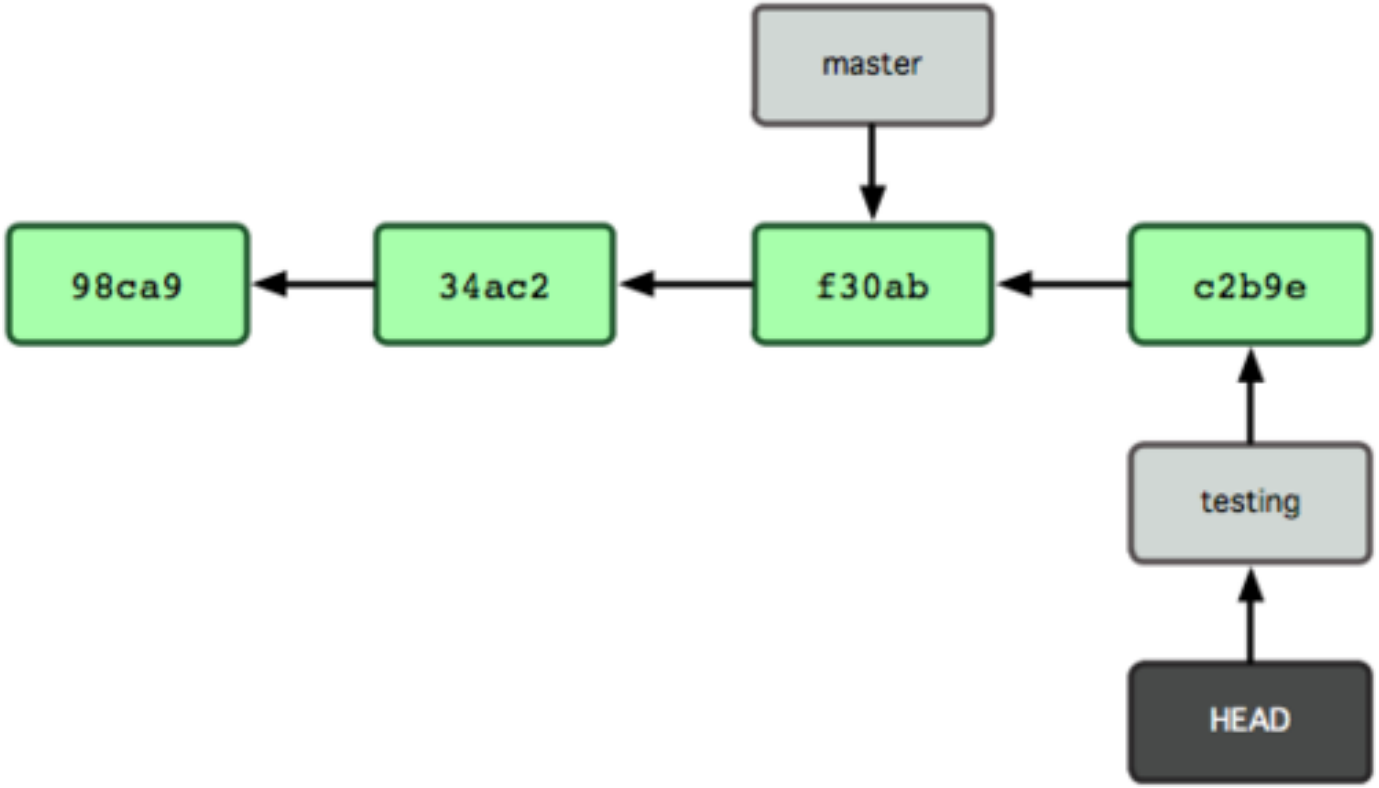
El concept de rama: HEAD



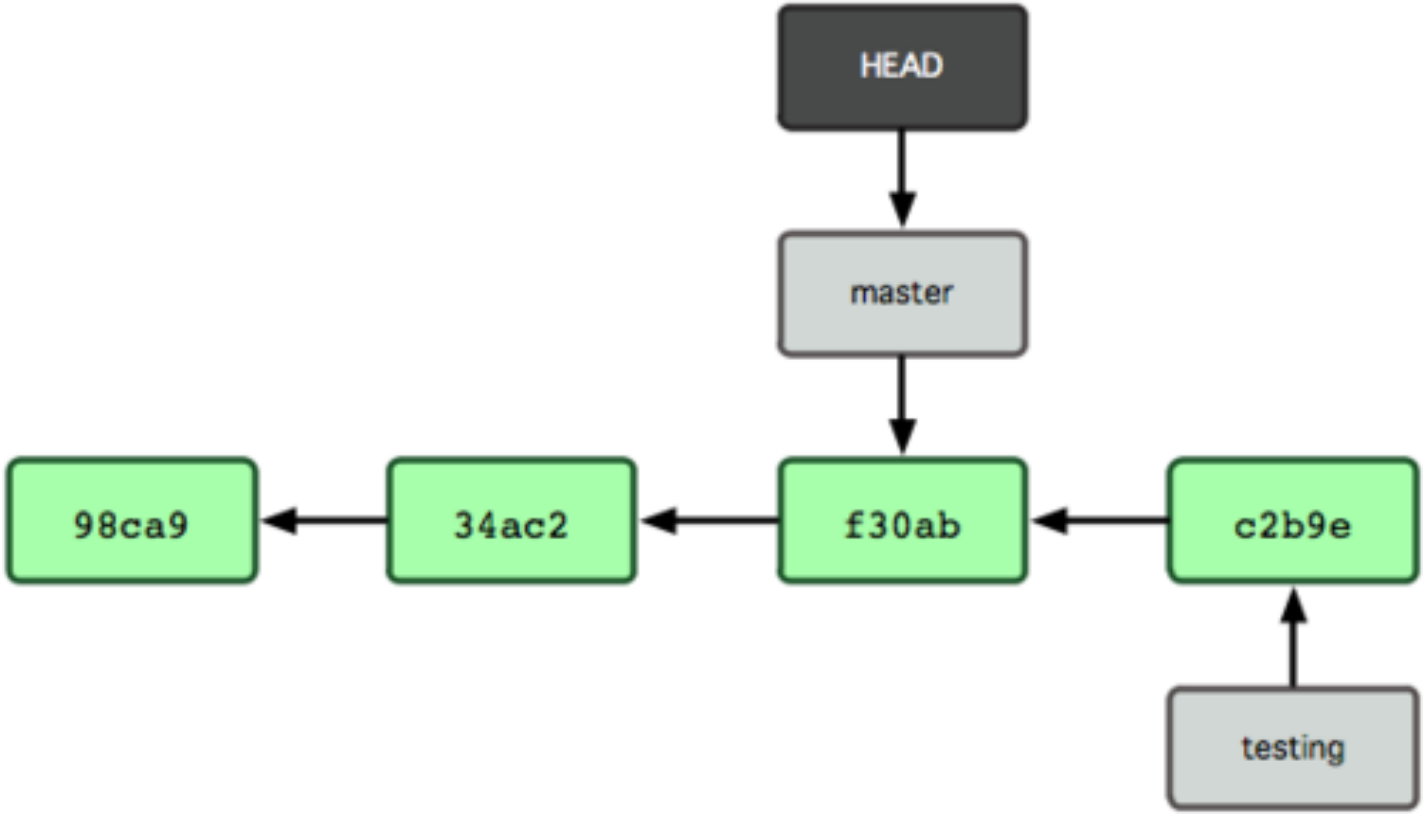
El concepto de rama: checkout



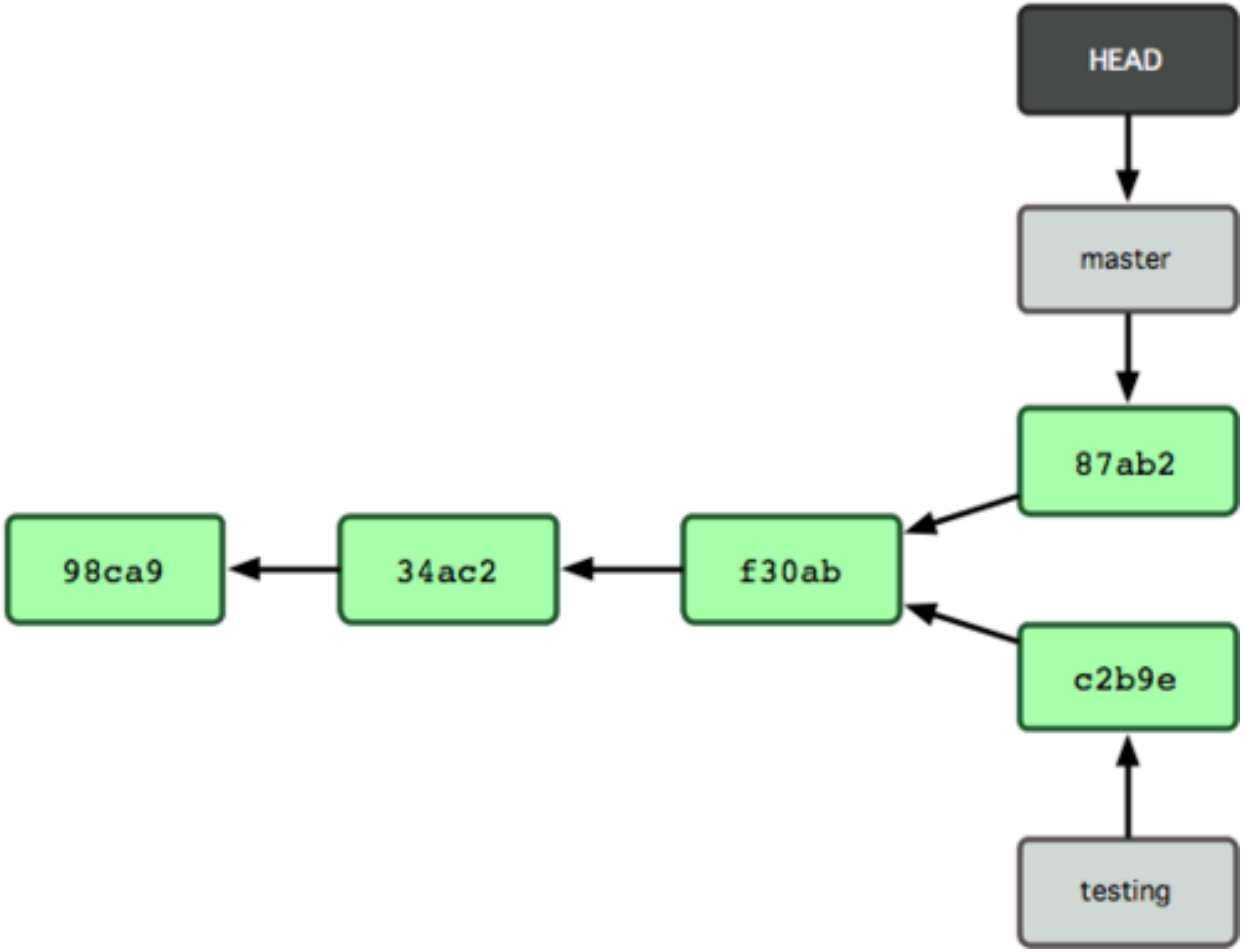
El concepto de rama: desarrollo paralelo



El concepto de rama: checkout y HEAD



El concepto de rama: divergencia de código



Procedure: Branching

Branch

- Create a new branch from the current branch.

```
$ git checkout -b 'hotfix'

Switched to a new branch "hotfix"
```

Edit

- Work

```
$ touch index.html
```

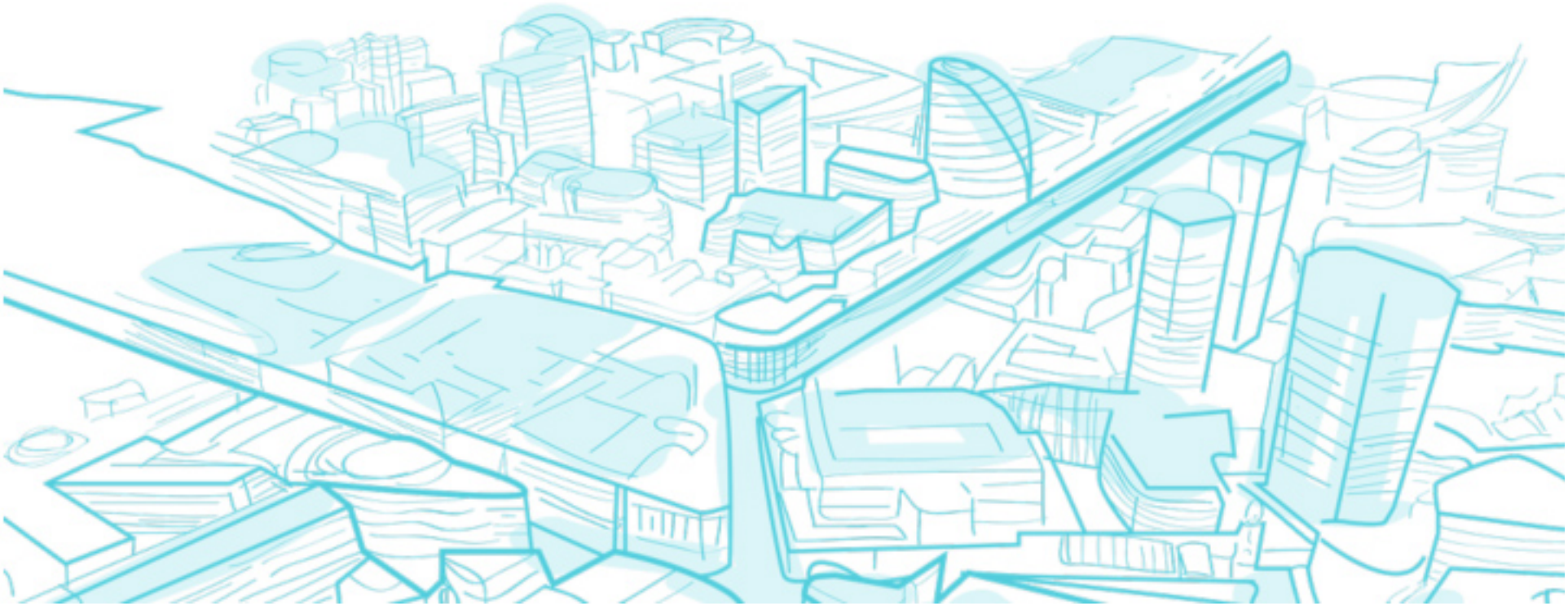
Commit

- Create the commit

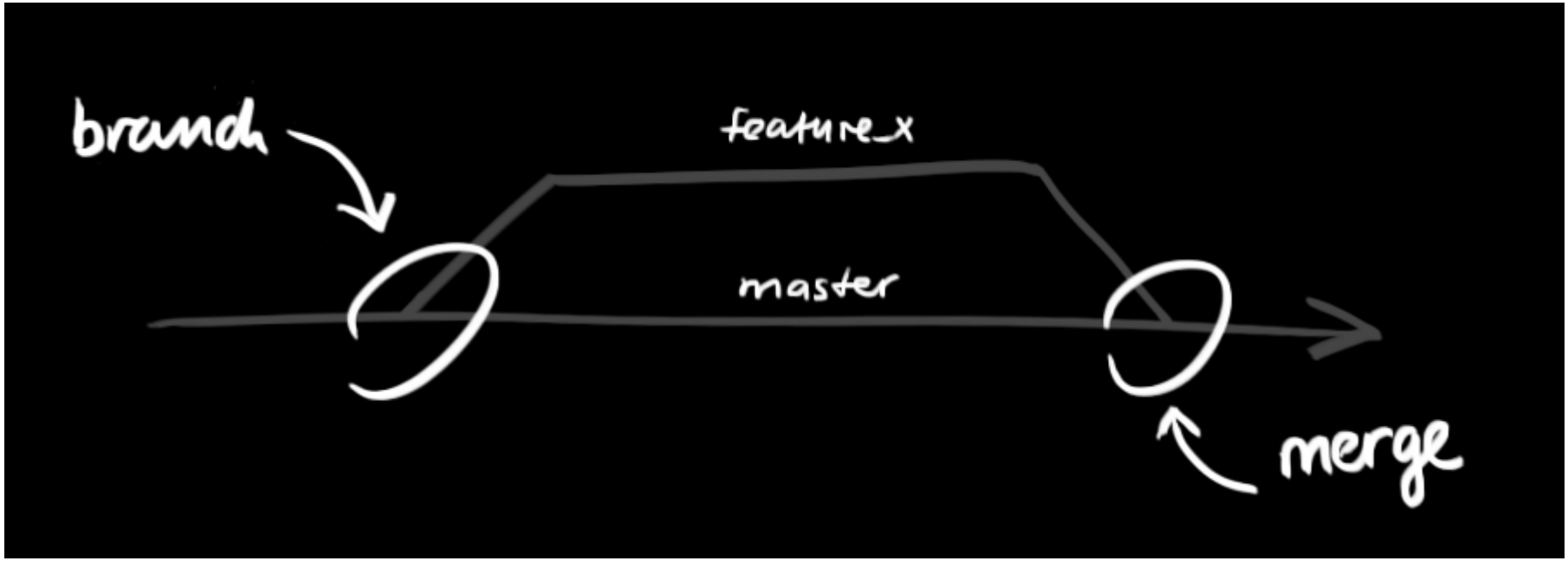
```
$ git commit -a -m 'fixed the broken email address'
[hotfix]: created 3a0874c: "fixed the broken email address"
1 files changed, 0 insertions(+), 1 deletions(-)
```

06

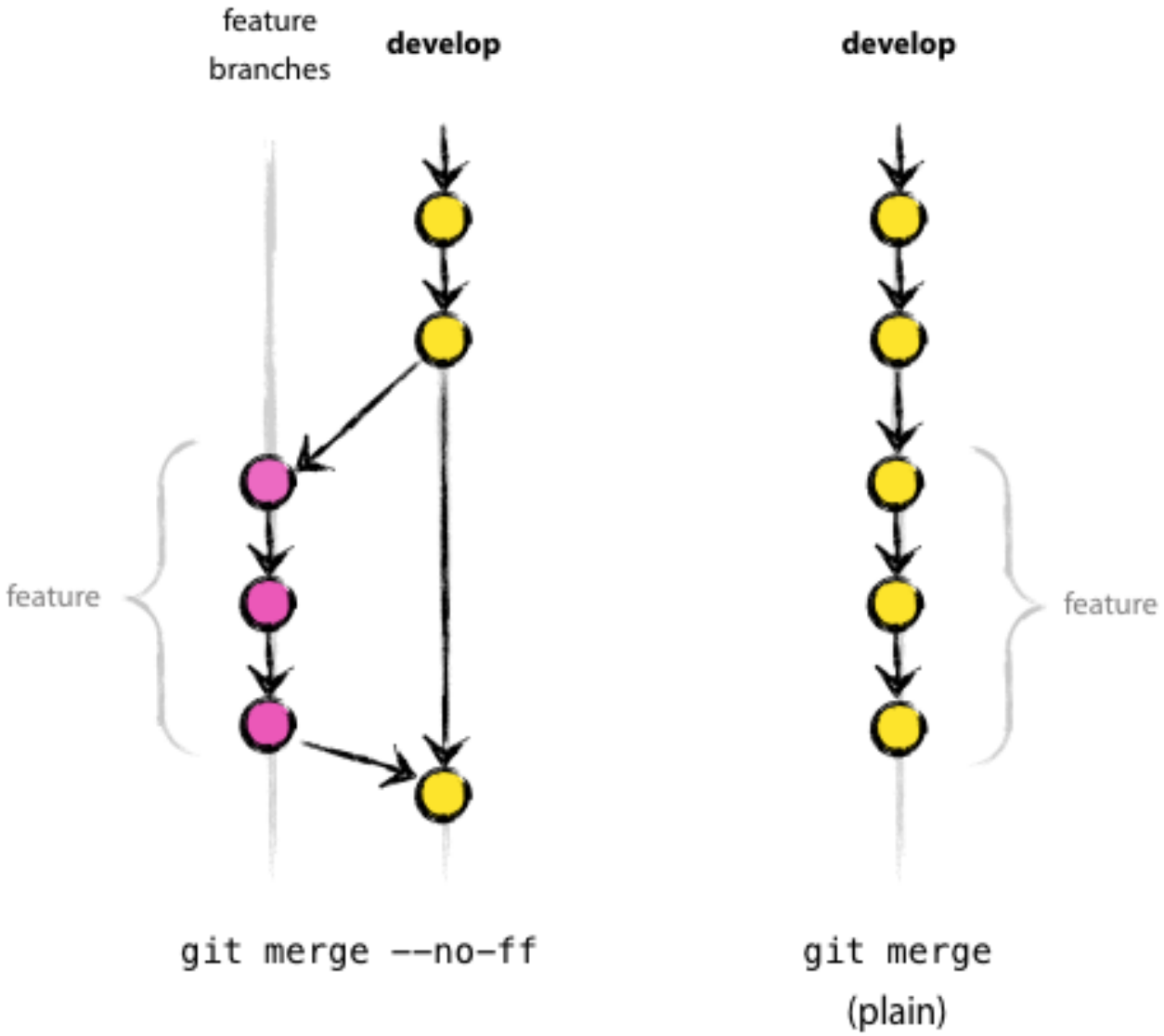
Git Merge



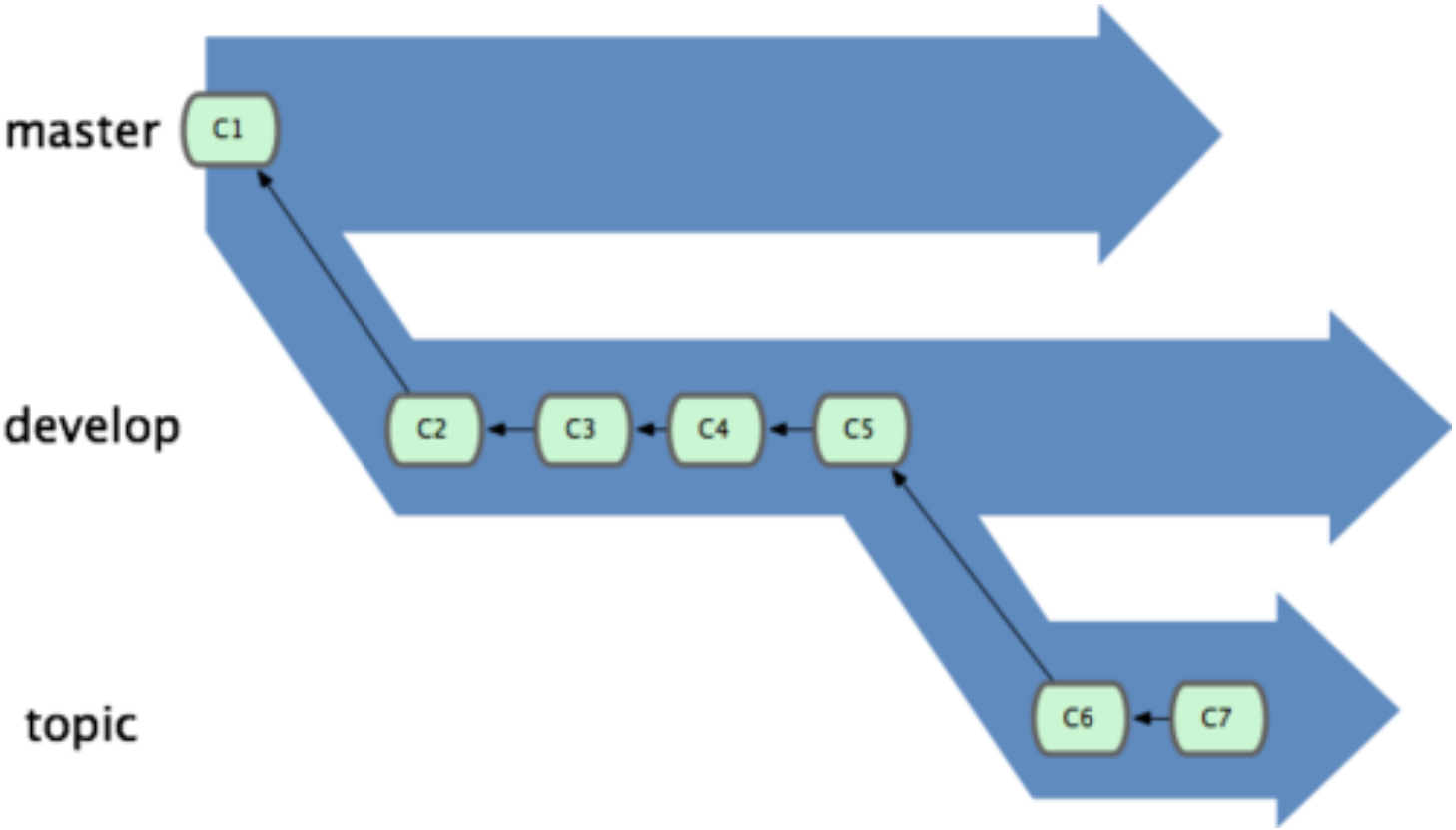
Git Merge: Main Concept



Git Merge: types



Branching & concepts



Procedimiento

Branch

- Branch is created

```
$ git branch feature/nueva
$ git checkout feature/nueva
```

Work

- Work in current branch

```
$ touch nueva
$ git add .
$ git commit –m “Add nueva”
```

Merge

- Target branch is merged in context branch

```
$ git checkout master
$ git merge feature/nueva
```

Git Merge: Conflict

Merge
- We create a conflict while merging

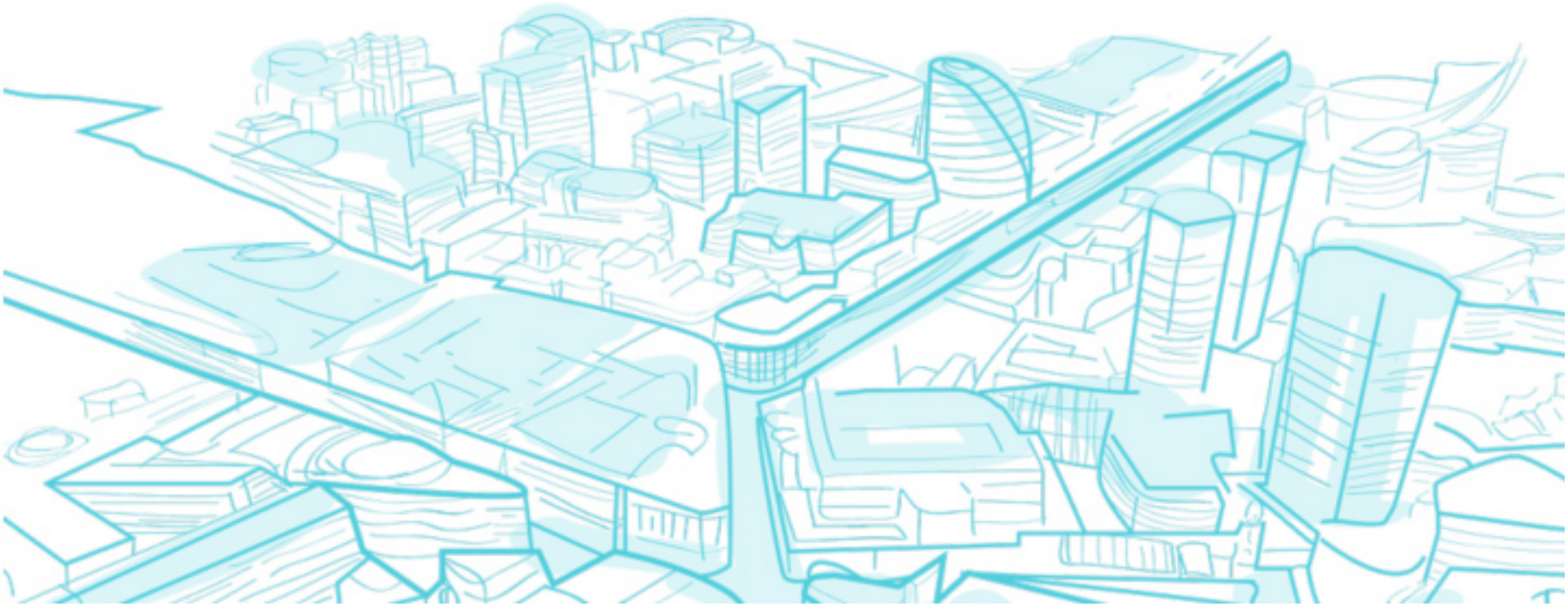
Fix
- Solve the conflict editing the file

Commit
- Create new commit [Solve]

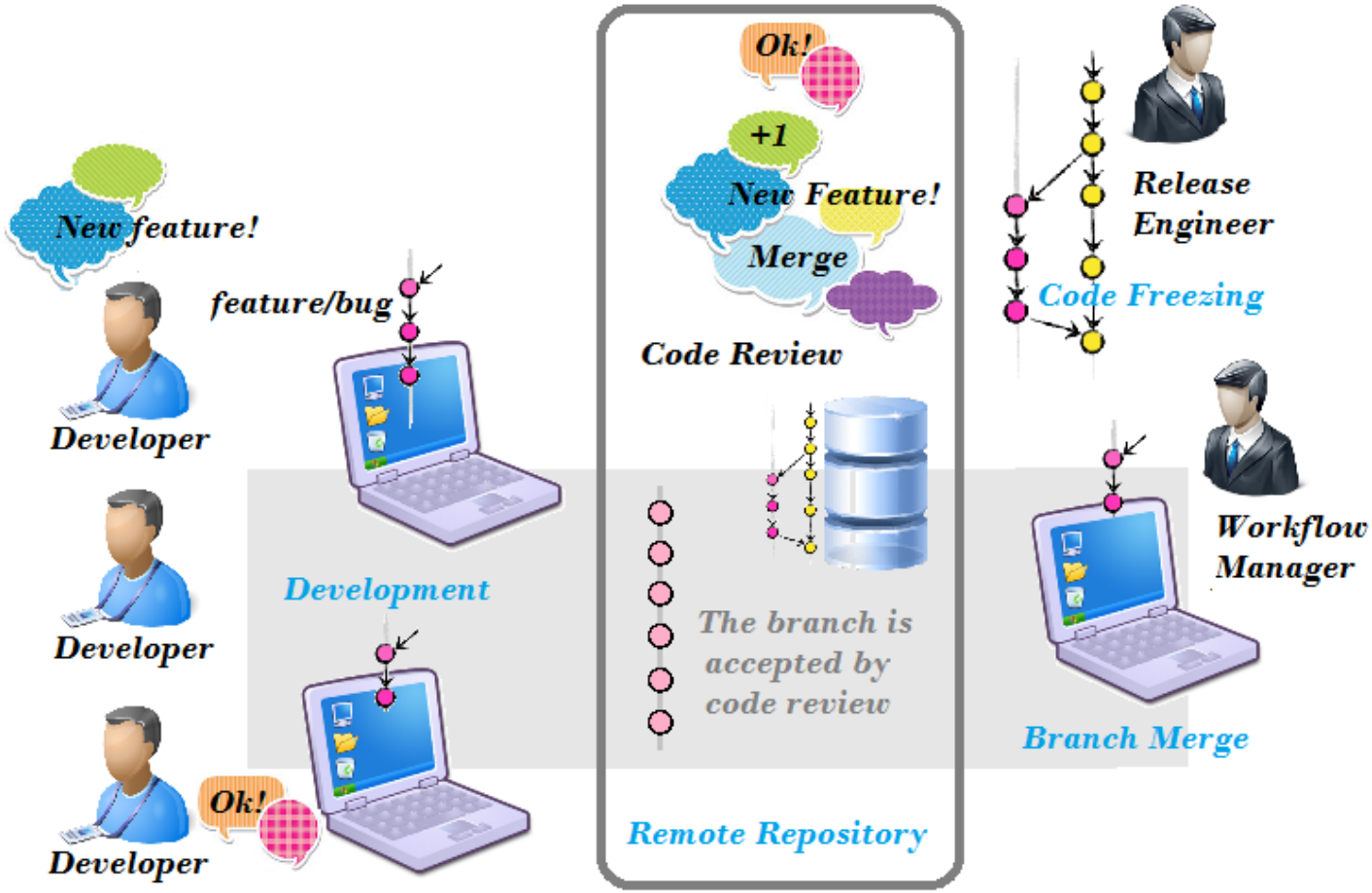
```
<<<<<<< HEAD:index.html
<div id="footer">
        contact : bmartinf@axpe.com
</div>
=======
<div id="footer">
        please contact me at
bmartinf@axpe.com
</div>
>>>>>>> issue:index.html
```

07

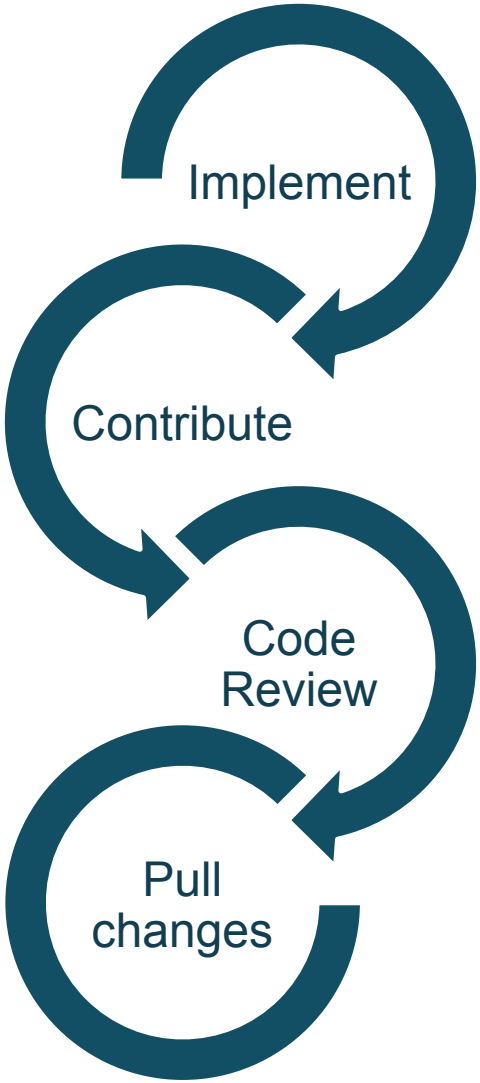
Code Reviews



Pull Request: Diagram



Merge request: “Pull Request”



Pull Request: Procedure

Push

- Push local changes to remote repository

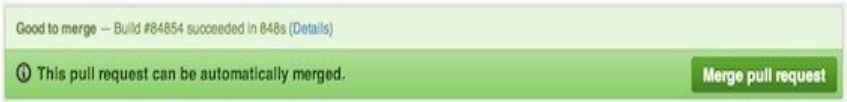
```
$ git push origin <branch>
```

PR

- Request to the repository owner by code review

Merge

- Merge is done by the owner



Forking a repository: Trust Network

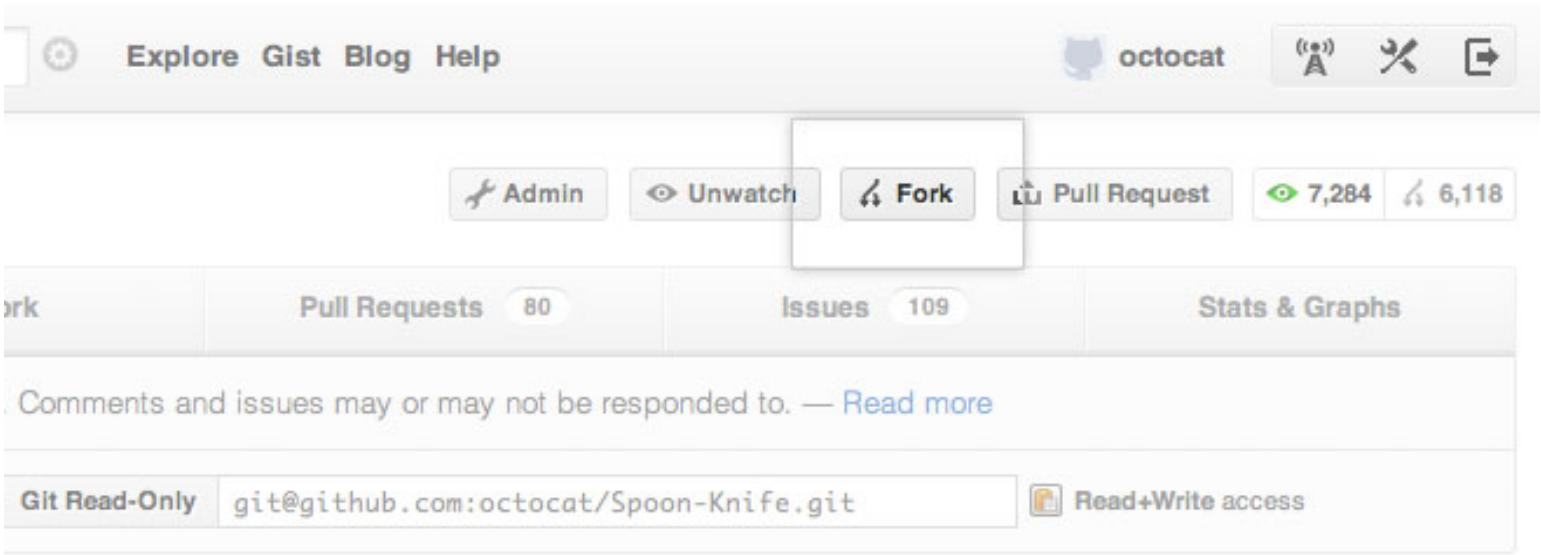
Each developer works in his fork

- It is a “cloned” repository
- Different sha1
- Owned by its creator
- Same privacity as the upstream.

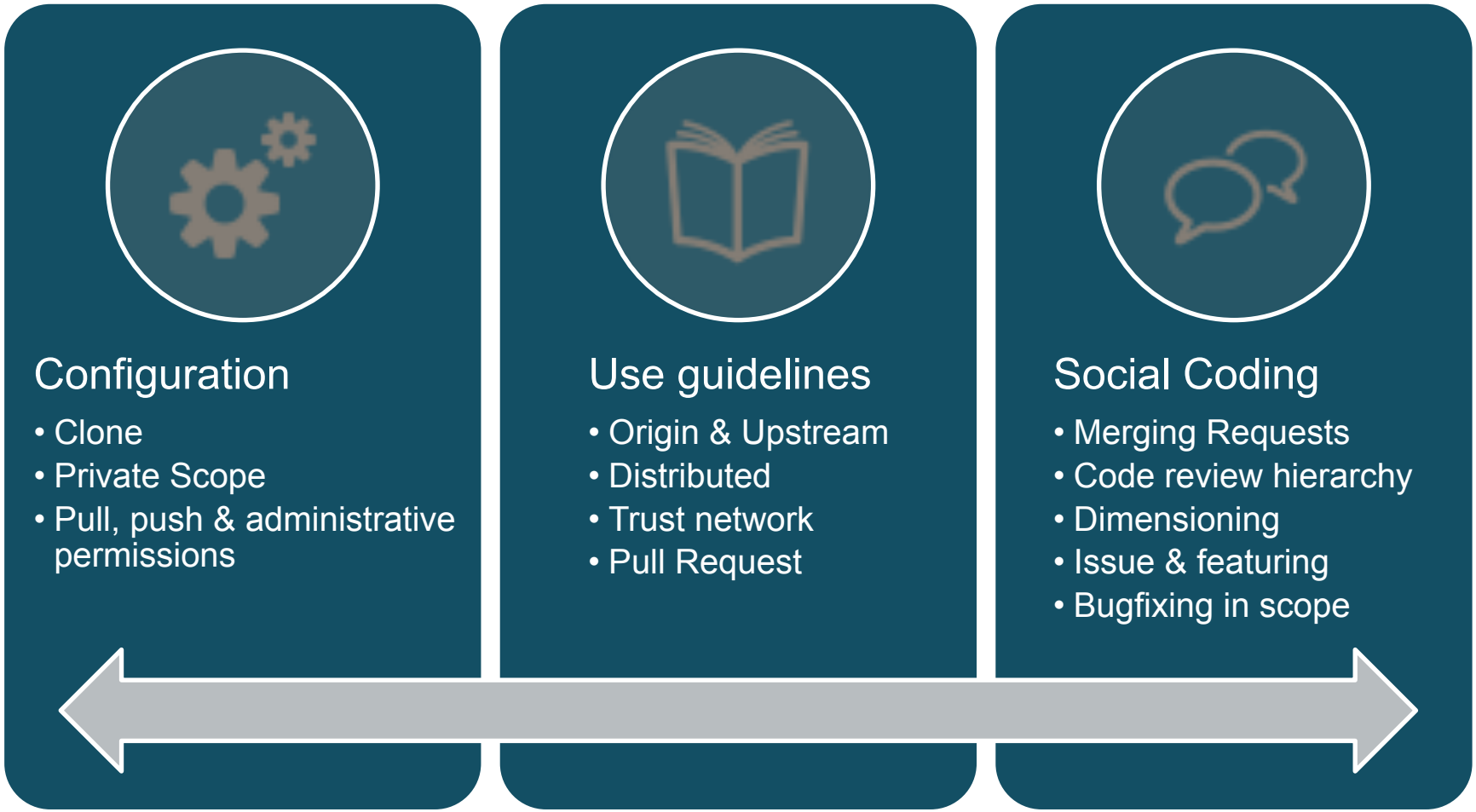
Pull Request afterwards

- Merged by the repository owner
- Each developer can participate by code review
- Automatic no-ff marging.

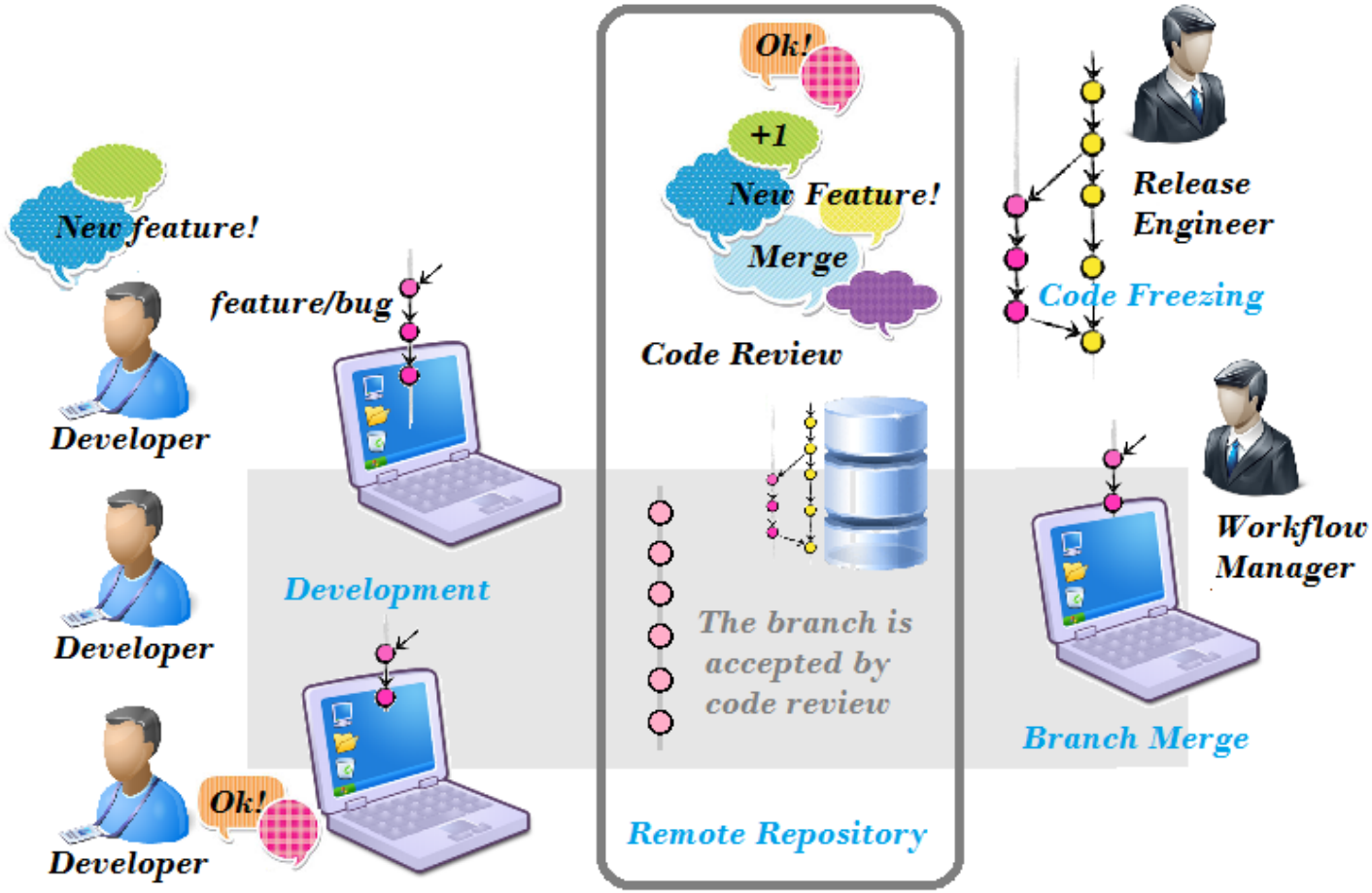
Fork de un repositorio: Clon remoto



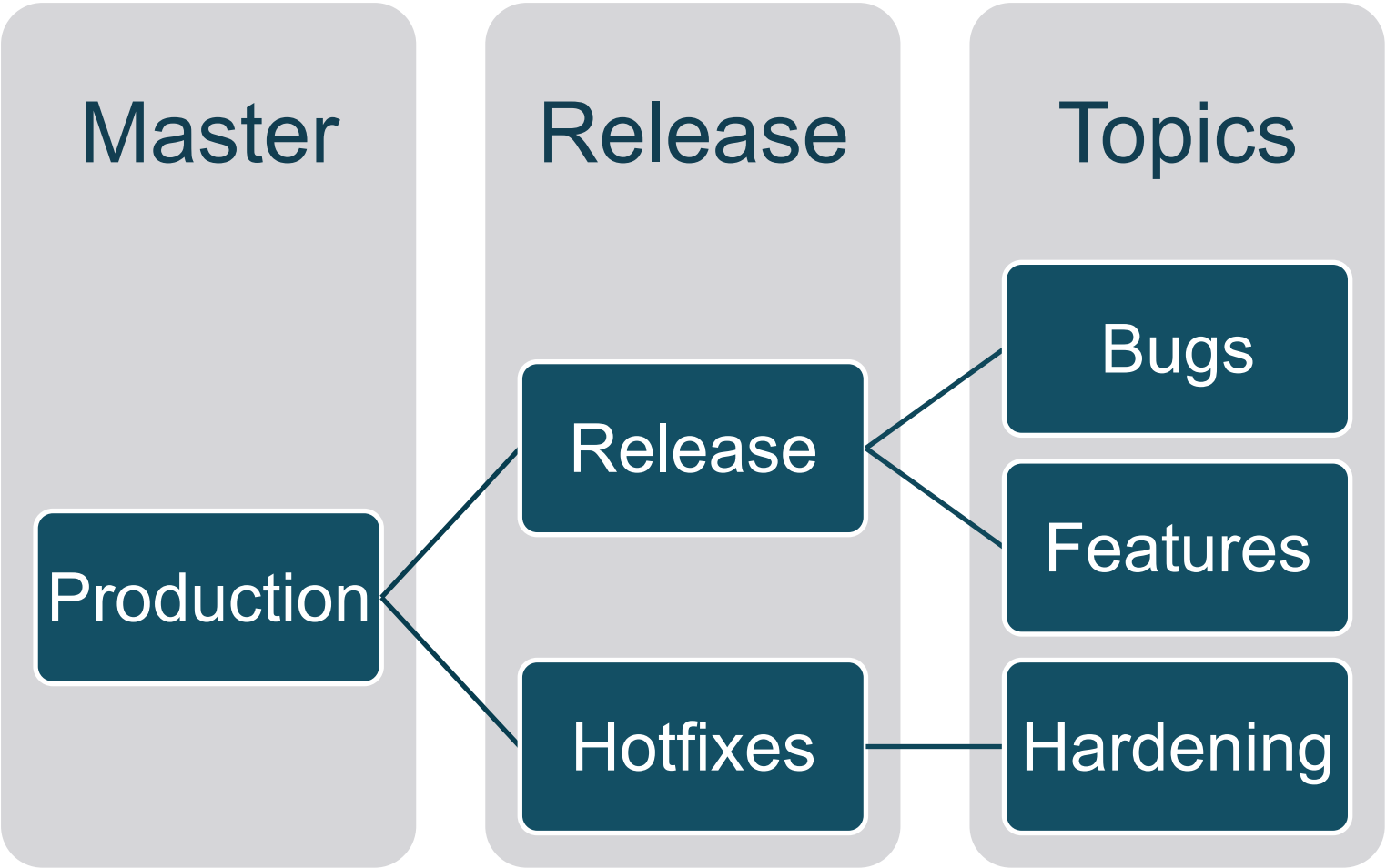
Fork: Distribution



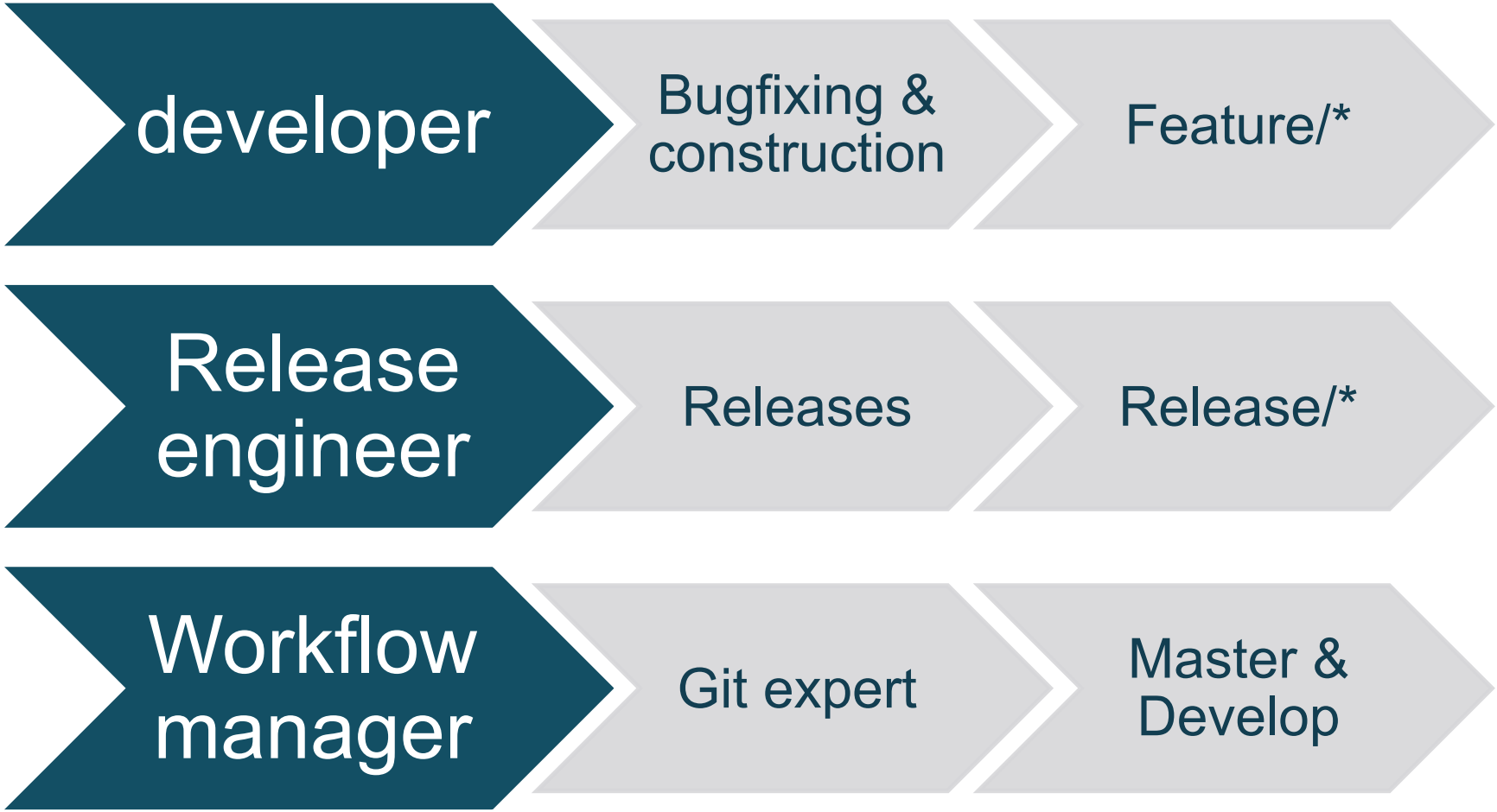
Pull Request: Diagram



Versioning procedure

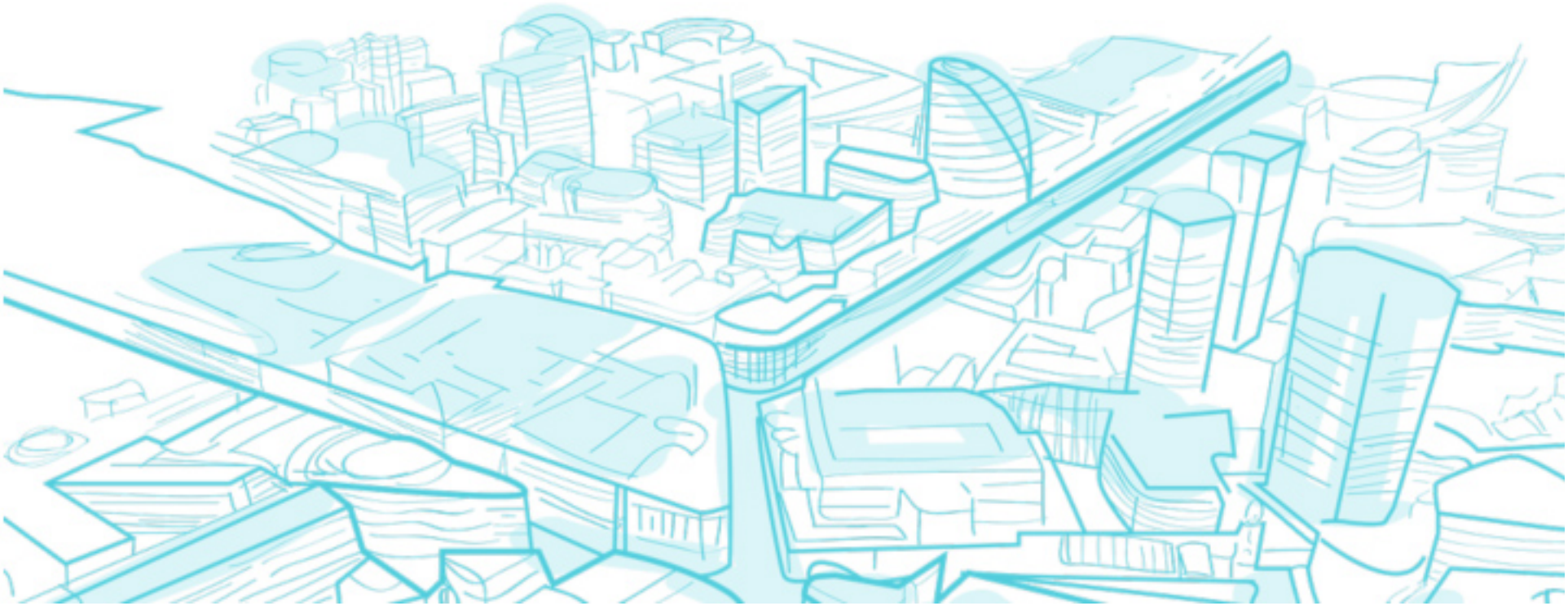


Roles, contexts and main branches

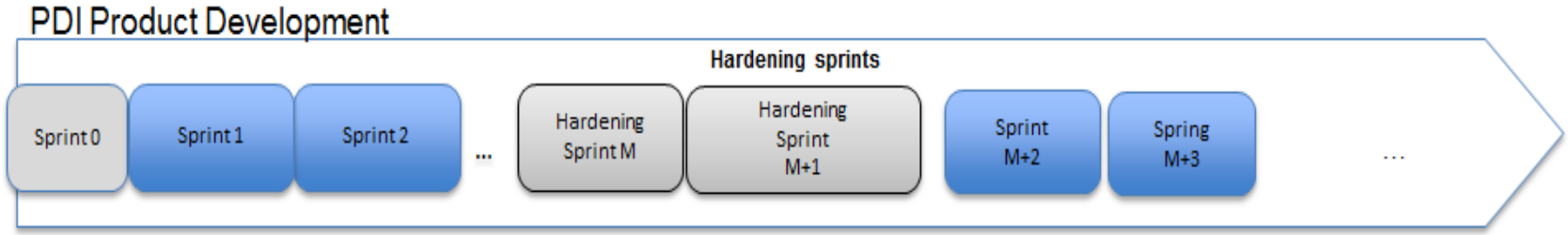


08

Main procedures



Process elements



Development team Checkpoint

Sprint

- Es un período de trabajo de la fase de construcción del ciclo de vida de un producto (PLC). En el flujo de trabajo deben aparecer marcados los hitos que diferencian los diferentes sprints, desde Kick Off, hasta que se declara Code Complete.

Hardening

- Es el proceso de refuerzo del código tras los sprints de implementación de las *features*. Nunca será utilizado para implementar funcionalidades nuevas. Únicamente será utilizado para editar el código: comentarios de código, reordenación, procesos de refactorización y optimización de los elementos y funcionalidades ya recogidas.

Process elements

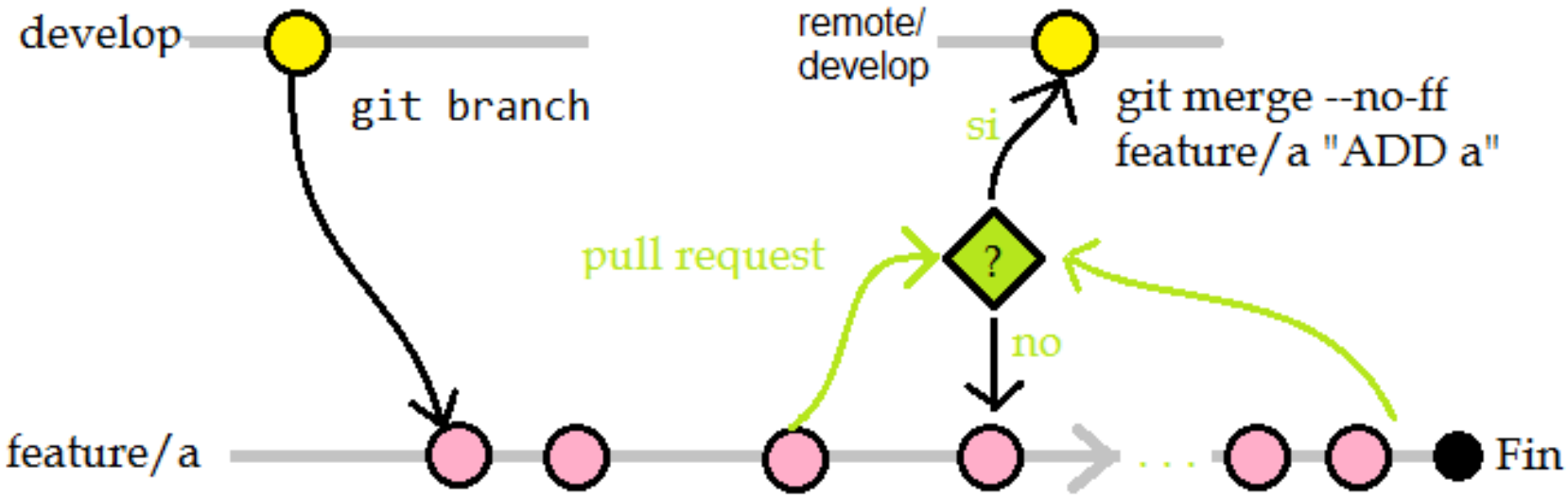
Feature

- Son características distintivas de un elemento del software: implementaciones, portabilidades, funcionalidades, etc.

Bug

- Los bugs se clasifican según el área de trabajo en el que hayan sido localizados, y se corregirán según las siguientes pautas:
  - Bugs locales: Deben ser corregidos en la rama local del desarrollador antes de subirlo a repositorio remoto y una vez solucionados, eliminarse del histórico.
  - Bugs en ramas feature: Surgen tras revisar un pull request hacia la rama develop. Deben resolverse directamente en la propia rama feature, utilizando los comentarios en la interfaz del pull request.
  - Bugs en la rama develop: Surgen tras ejecutar las tareas periódicas de revisión. En caso de fallo, debe crearse un issue y resolverse en una rama bug cuyo nombre coincidirá con el número del issue (ejemplo: bug/44, para el issue #44)
  - Bugs en las ramas release: Es el caso más común, surgen durante la validación del código. Deben corregirse en ramas bugs, con el número del issue que generan. Su corrección debe mergearse tanto a la rama develop, como a la rama release, para mantener la consistencia del repositorio.

Featuring



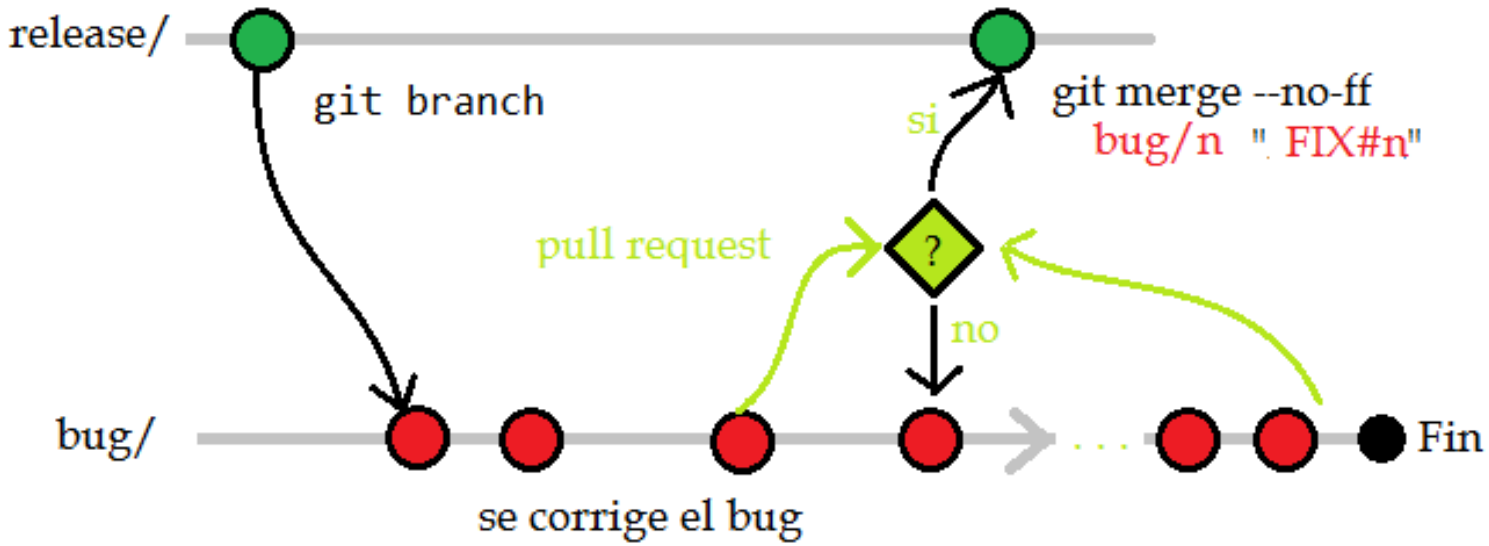
Branch

- New feature/#user_story (JIRA) will be created.
- Developers will work in the feature branches

PR

- Afterwards, it will be merged after pull request

Bugfixing

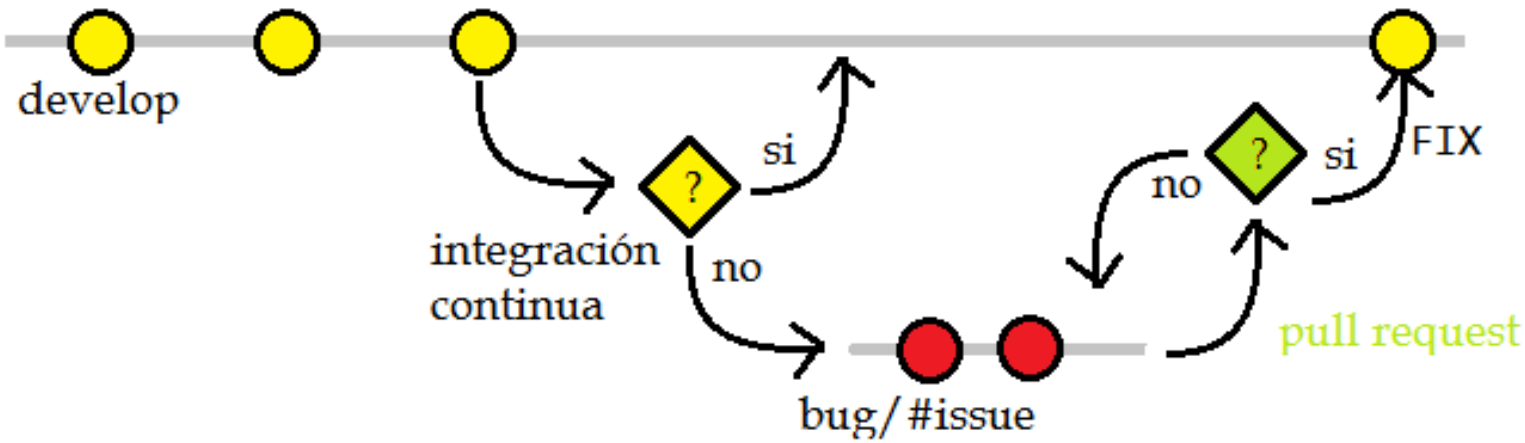


Bug
- New bug/#issue branch is created after being detected.

Fix
- Bugfixing in the bug/* branch.

PR
- Pull Request to merge bug/* branch is requested.

Continuous Integration (develop branch)



Hook
- New commit triggers a hook.

Jenkins
- CI is activated by the hook

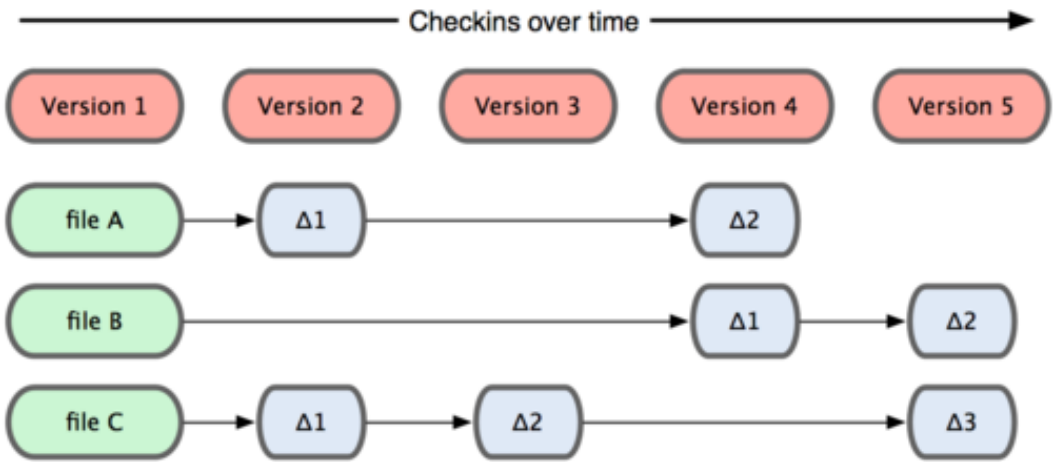
Bug
- If it fails, a bug/* branch will be created/merged by pull request.

Telefónica

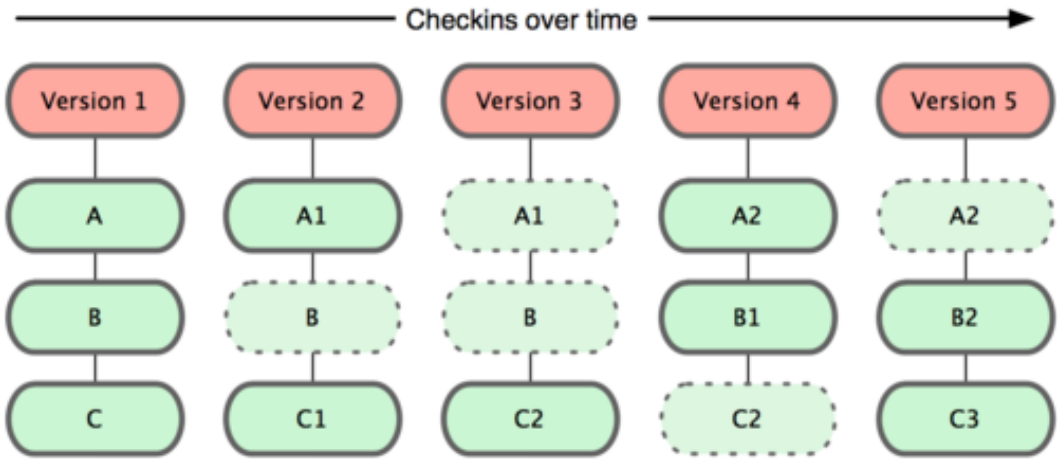
Git Merge: Consideraciones sobre los conflictos

¿Qué ocurre cuando git no puede identificar qué versión debe almacenar?	• Genera un conflicto
¿Es una decisión mecánica?	• No, debe resolverse de forma “humana”
¿Ocurre siempre que realizamos merge?	• No, solo cuando tiene ramas paralelas cuyo conflicto afecta al mismo archivo y no puede garantizar la secuencia de commits.
¿Si propagamos los conflictos por el repositorio, debemos resolverlos de la misma manera?	• Sí, el operador “rerere” nos permite guardar la resolución utilizada. (Reuse-Recorded-Resolution) De esta manera los conflictos solo se resuelven una vez.

Snapshots en git: Minisistema de archivos

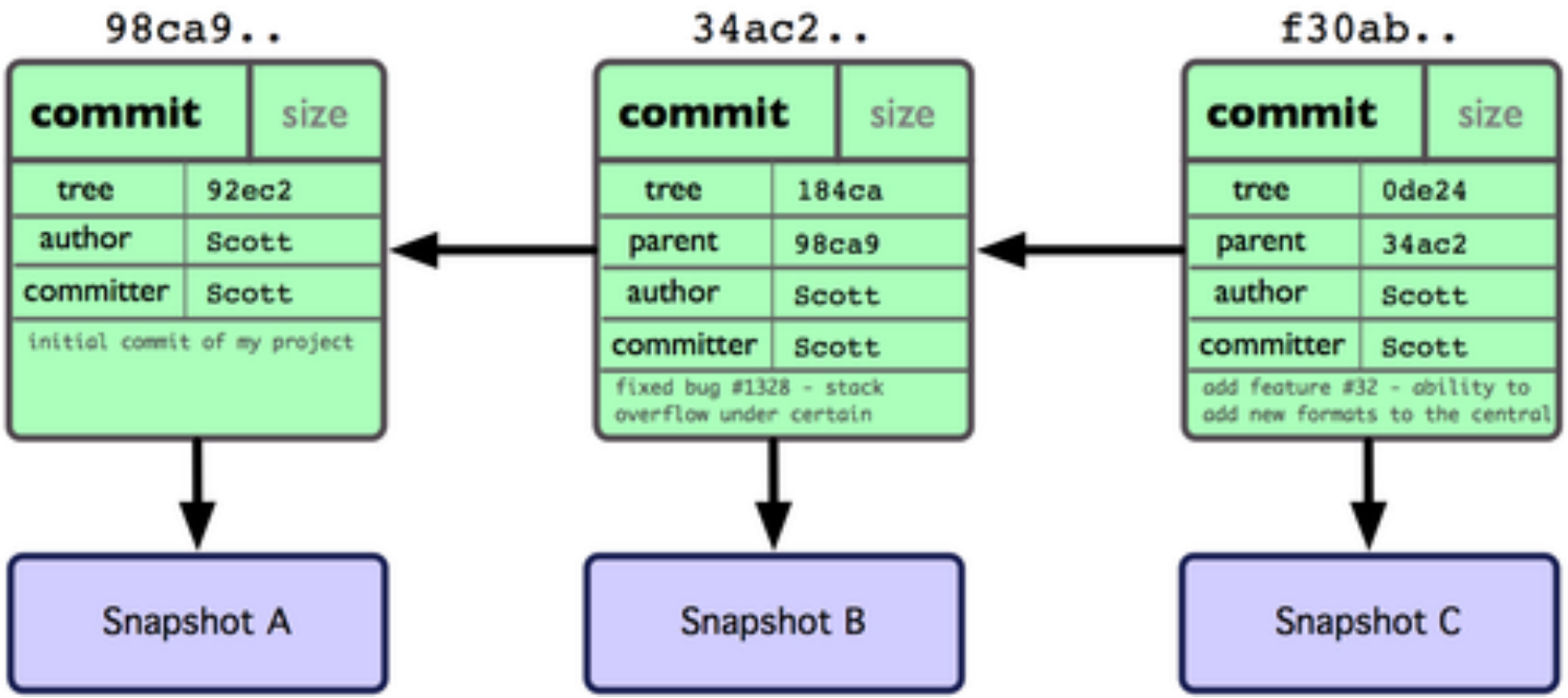


Otros controles de versiones



Git guarda las modificaciones anteriores en caso de no detectar cambios

El concepto de rama: grafo



Agradecimientos

Borja Martín Fernández:

- GIT-Coach
- PDIHUB/GitHub.com Administrator

Objetivos principales del SCM

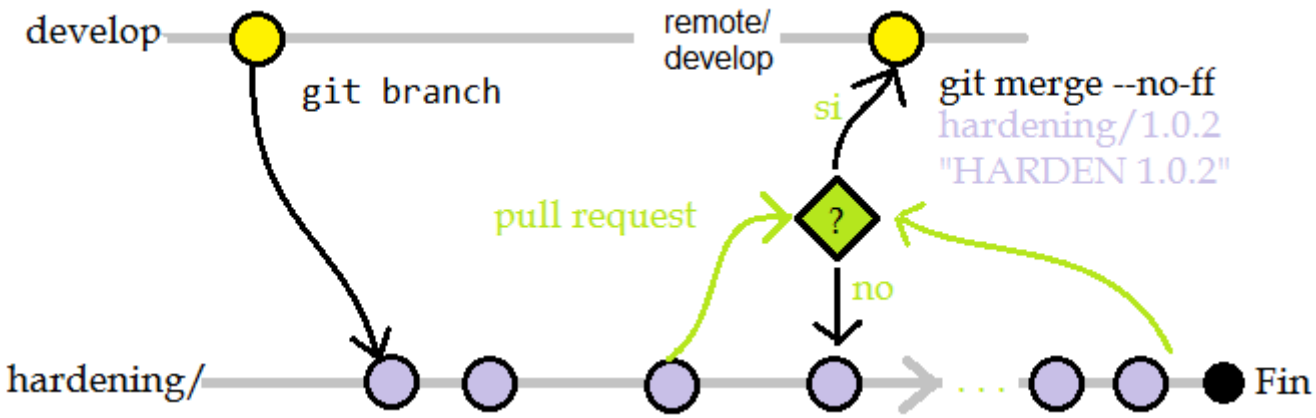
- Definir los convenios y políticas para la gestión de las iniciativas en PDIHUB
- Establecer los procesos para gestionar los repositorios
- Establecer los contextos y ramas de desarrollo
- Definir las responsabilidades y roles

PDIHUB: GitHub Enterprise License

Social Coding

https://pdihub.hi.inet

Hardening



FC

- Se declara Functionality Complete

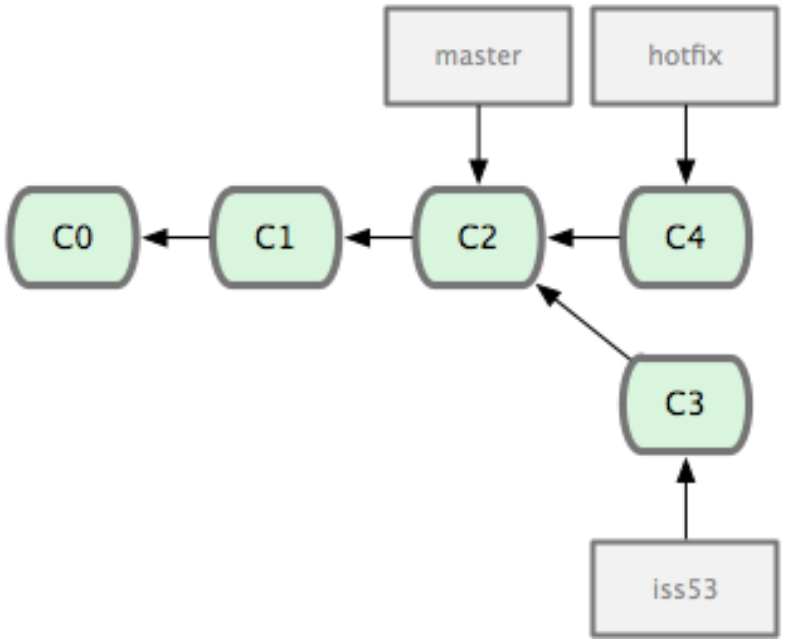
Harden

- Se crea una rama hardening/* y se comienza a refactorizar aquéllos elementos deseados.

PR

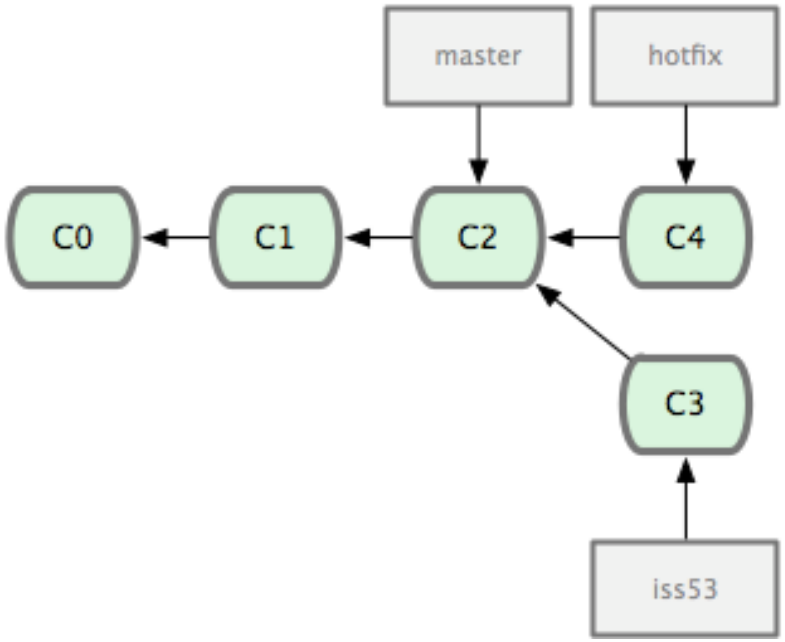
- Se solicita Pull Request para declarar el código “Code Complete”

Git Merge: Procedimiento



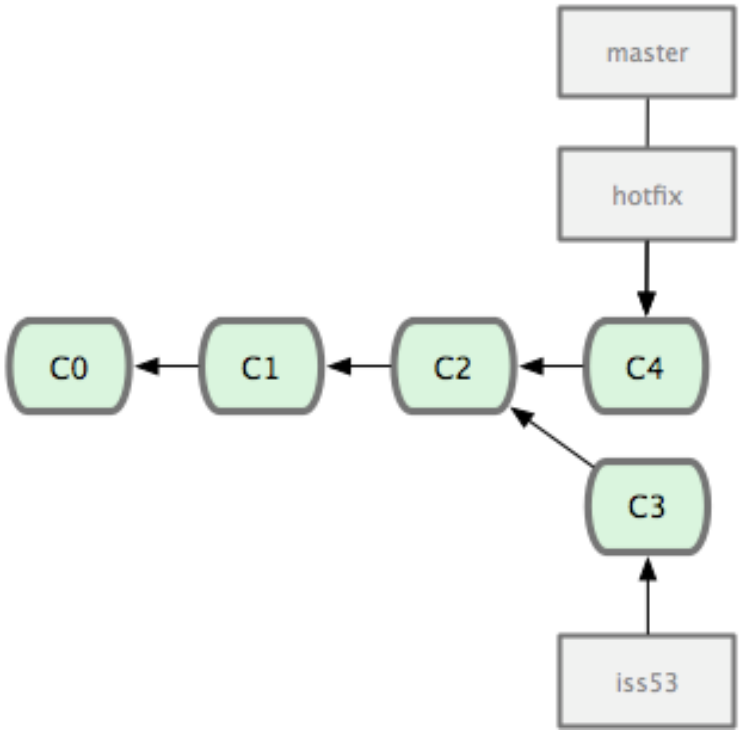

```
$ git checkout -b 'hotfix'

Switched to a new branch "hotfix"



$ touch index.html



$ git commit -a -m 'fixed the broken
email address'
[hotfix]: created 3a0874c: "fixed
the broken email address"
1 files changed, 0 insertions(+),
1 deletions(-)
```

Git Merge: Procedimiento



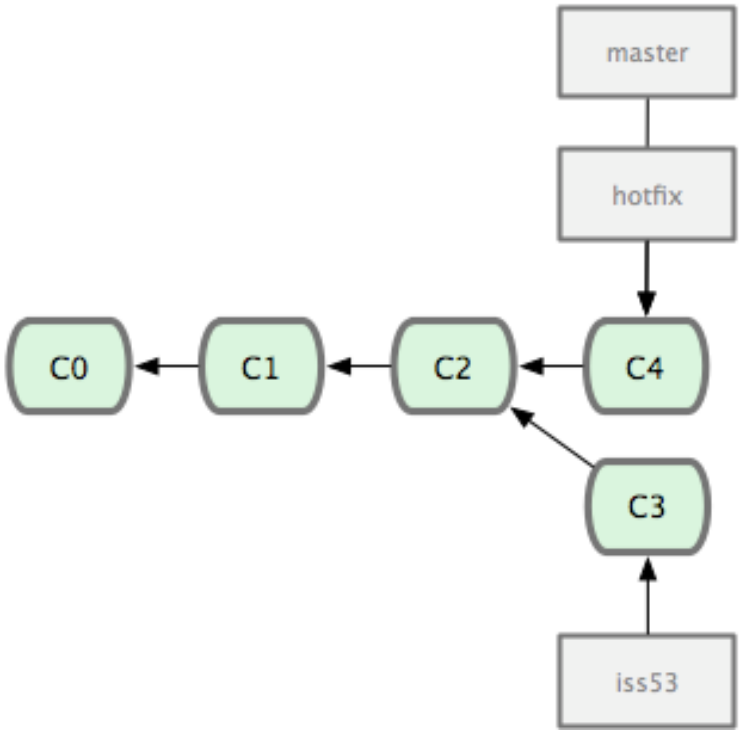

```
$ git checkout master

$ git merge hotfix Updating
f42c576..3a0874c
Fast forward

README | 1 - 1 files changed, 0
insertions(+), 1 deletions(-)
```

Git Merge: Procedimiento



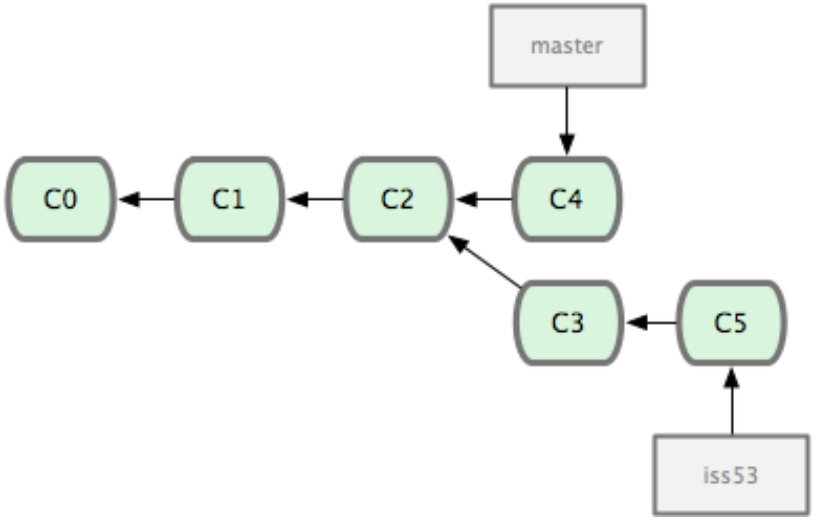

```
$ git checkout master

$ git merge hotfix Updating
f42c576..3a0874c
Fast forward

README | 1 - 1 files changed, 0
insertions(+), 1 deletions(-)
```

Git Merge: Procedimiento



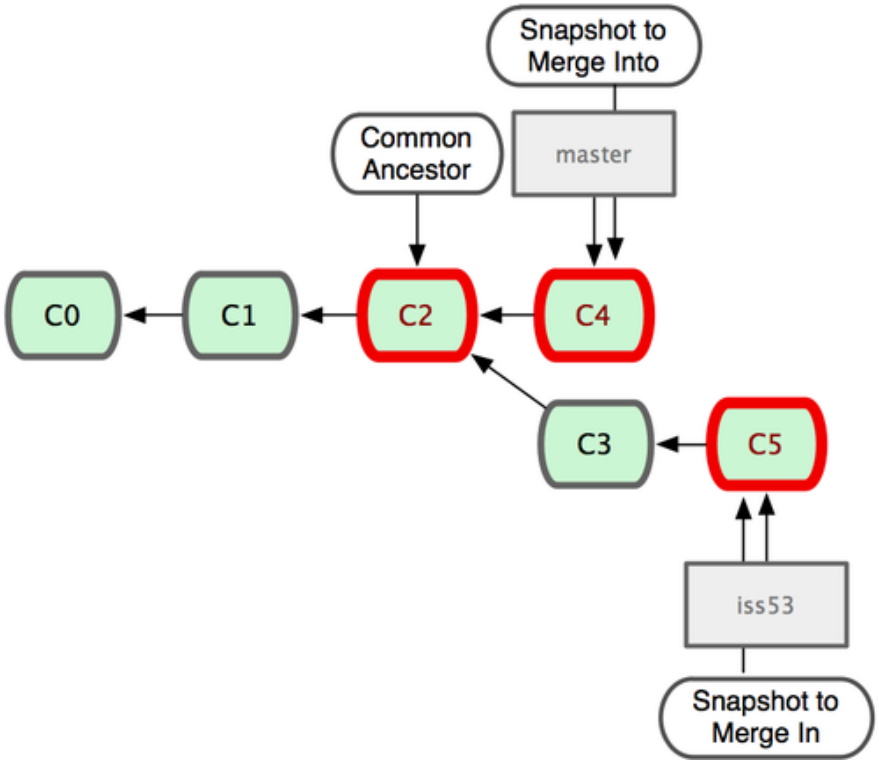

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).

$ git checkout iss53
Switched to branch "iss53"

$ vim index.html $ git commit -a -m
'finished the new footer [issue 53]‘

[iss53]: created ad82d7a: "finished
the new footer [issue 53]"
1 files changed, 1 insertions(+), 0
deletions(-)
```

Git Merge: Procedimiento




```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).

$ git checkout iss53
Switched to branch "iss53"

$ vim index.html $ git commit -a -m
'finished the new footer [issue 53]‘

[iss53]: created ad82d7a: "finished
the new footer [issue 53]"
1 files changed, 1 insertions(+), 0
deletions(-)
```

Git Merge: Resultado




```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).

$ git checkout iss53
Switched to branch "iss53"

$ vim index.html $ git commit -a -m
'finished the new footer [issue 53]‘

[iss53]: created ad82d7a: "finished
the new footer [issue 53]"
1 files changed, 1 insertions(+), 0
deletions(-)
```

Commits: Elementos

Elementos

- Sha1 (ID)
- Cambios
- Autor
- Timestamp
- Comentario
- Antecesor