



Versioning & Repository

Docente: Andrea Francesco Speziale

Chi Sono e Contatti



Andrea F. Speziale

Fotografo // Film-maker
Designer // Developer



andrea.speziale@top-ix.org



andreafspeziale

Front-end vs Back-end



Necessitano di strumenti per lavorare in team!
Altrimenti...

Front-end vs Back-end



Versioning & Repository <3



- tutti gli effetti un software
- effettuare delle operazioni sui nostri contenitori
- gestire in modo comodo le versioni dei nostri contenitori
- gestire in modo comodo i conflitti che possono crearsi quando i due sviluppatori effettuano delle modifiche sugli stessi file



- contenitori per il vostro codice
- comode per condividere il codice con il team di sviluppo
- comode per creare degli ambienti di sviluppo separati

Repository <3

Abbiamo detto che sono utili per **condividere** il codice con il team di sviluppo



github
SOCIAL CODING

Come?

Servizi online che permettono di ospitare gratuitamente o a pagamento le nostre **repository**.

Non sono altro che dei siti web sui quali tramite il **software** di **versioning** possiamo **uploadare/scaricare** e quindi **condividere** con altri le nostre **repository** e quindi il **nostro codice**



Bitbucket

Repository <3



Ora che abbiamo i nostri account tramite linea di comando spostiamoci sulla cartella di **lavoro** (se non l'abbiamo ne creiamo una sul desktop) e :

- Al suo interno creiamo una cartella **repo_vostroCognome**
- Al suo interno creiamo un file txt chiamato **README.txt**

repo_vostroCognome sarà la nostra *repository locale*, che andremo ad “uplodare” su **Github**



Versioning <3

Abbiamo detto che
è un **software** quindi dovremo installarlo
(nel caso non lo sia)

ci serve per poter eseguire delle **operazioni**
sulle nostre repository come ad esempio
l'**upload** e il **download** di quest'ultime

Verifichiamone l'installazione di entrambi

```
$ git - -version
```

```
$ hg - -version
```



Git <3



- non ha una **GUI** quindi una graphic user interface
- essendo un software a tutti gli effetti ha comunque dei comandi che posso essere eseguiti e con i quali operiamo sulle repository
- ci mette a disposizione un **helper**, ovvero una lista di comandi con tanto di descrizione degli stessi

\$ git oppure \$ git - -help

- inoltre ci dà la possibilità di visualizzare i concetti base per ciascun comando o di aprire una guida organizzata per categorie

\$ git help nomeComando e \$ git help -g

Git - configurazione <3

- abbiamo creato il nostro account su github.com
- agganciamo alla nostra configurazione git locale il nostro account in questo modo



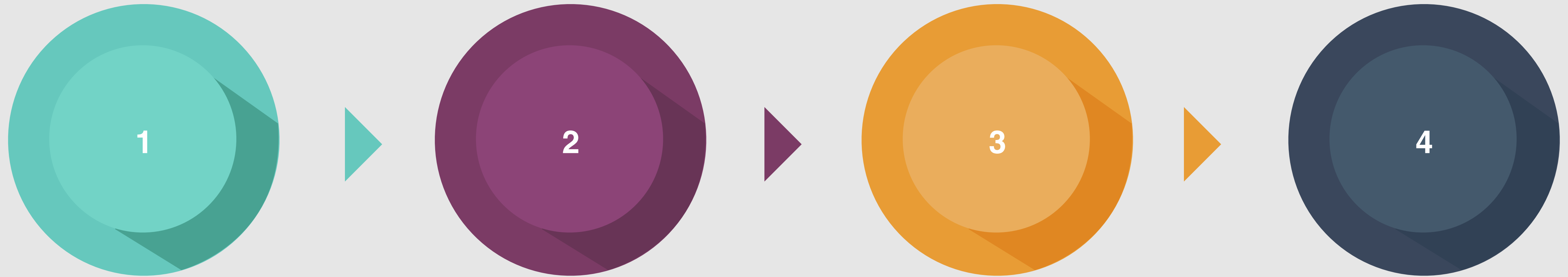
```
$ git config --global user.name <NAME>  
$ git config --global user.email <EMAIL>
```

A questo punto siamo pronti per mettere online la nostra prima repository tramite i comandi del sistema di versioning Git!

Esercitazione 1



Esercitazione 1 - obiettivi



Creare una
repository remota

Associare la repository
remota alla nostra
repository in locale

Tracciare i file della
nostra repository

Pusshare (uploadare)
online i file di cui
stiamo tenendo traccia

Esercitazione 1 - tasto ‘+’

- Apriamo **Github** e **logghiamoci** con il nostro account
- Una volta loggati in maniera molto intuitiva clicchiamo il tasto **+** in alto a destra per cominciare la creazione di una nuova **repository remota**
- **Utils for Dummies**
<http://andreafspeziale.github.io/utils/>

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name

Great repository names are short and memorable. Need inspiration? How about **petulant-quack**.

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▼

Add a license: **None** ▼



Create repository

Esercitazione 1 - tasto ‘+’

Compiliamo i vari campi:

- **name** buona norma chiamare la repository remota come il progetto in locale alla quale vogliamo agganciarla (quindi inseriamo)
- **descrizione** buona norma inserire sempre una descrizione brevissima e significativa, in questo modo quando andremo sulla lista delle nostre repo avremo subito un'idea di cosa c'è dentro ognuna di esse
- **pubblica / privata** come vi accenavo Github e Bitbucket alla creazione di ogni repo vi daranno la possibilità di creare delle repo pubbliche o private quindi con diversi permessi di accesso e visibilità da parte di altri utenti

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name

Great repository names are short and memorable. Need inspiration? How about **petulant-quack**.

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



Create repository

Esercitazione 1 - tasto ‘+’

Per il momento non consideriamo le opzioni infondo perché li aggiungeremo noi manualmente, queste opzioni non servono ad altro che aumatizzare il loro inserimento

- **README**
- **gitignore**

sono due file molto utili in cui nel primo scriviamo le informazioni approfondite del nostro progetto e ne secondo inseriamo il nome dei file dei quali non vogliamo che venga tenuta traccia, vedremo poco dopo cosa significa nella pratica

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▼

Add a license: **None** ▼



Create repository

Esercitazione 1 - tasto '+'

PREMIAMO CREATE!

Esercitazione 1 - tasto '+'




Esercitazione 1 - post 'CREA'

Dovremmo ritrovarci a questa schermata

Introducing a new repository design

Hey there! We're [rolling out](#) a faster, more streamlined repository experience and would love to give you early access.

Switch to new design

 [andreafspeziale](#) / [prova](#)

Unwatch 1

Star 0

Fork 0

Quick setup — if you've done this kind of thing before

Set up in Desktop

 or

HTTPS

SSH

<https://github.com/andreafspeziale/prova.git>

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# prova" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/andreafspeziale/prova.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/andreafspeziale/prova.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

<> Code

Issues 0

Pull requests 0

Wiki

Pulse

Graphs

Settings



Esercitazione 1 - HTTPS

In questa schermata **Github** ci proporrà ancora dei suggerimenti per la nostra repository remota ma noi impareremo a fare tutto a mano in questo modo avremo il pieno controllo della creazione

L'unica informazione utile che ci servirà è il link:

<https://github.com/nomeVostroAccount/nomeRepository.git>

Quick setup — if you've done this kind of thing before

 **Set up in Desktop** or **HTTPS** **SSH** 

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

Esercitazione 1

creare una
repository remota
ARCHIVED

Esercitazione 1 - remote & local

Il prossimo passo è agganciare la nostra repository locale a quella remota.
Per effettuare ciò utilizzeremo a pieno le funzionalità del **sftw di versioning Git**

Spostiamoci sul terminale e navighiamo fino alla root della nostra cartella repo_vostroCognome ed eseguiamo il comando

git init .

git → specifico che sto invocare un comando del sftw di versioning git

init → inizializzo da quel momento Git guarderà lo stato dei miei file

. → indica 'nell posizione in cui mi trovo'

Esercitazione 1 - remote & local

Ora che abbiamo attivato Git nella root del nostro progetto agganciamo la nostra copia locale della repository con il seguente comando

```
git remote add origin git@github.com:MyUsername/nameOfTheOnlineRepo.git
```

Un comando molto utile e fondamentale è

```
git status
```

Ogni volta che avete qualche dubbio sullo stato della vostra repository utilizzate questo comando e lo scoprirete!

Provate subito!

Esercitazione 1

*Associare la repo remota alla
nostra repo locale*

ARCHIVED

Esercitazione 1 - remote & local

Ora vi ricordate che alla creazione della repository, sul sito di Github, il wizard ci chiedeva se volevamo creare i file **.gitignore** e **readme**?

Bene proviamo a crearli ora a mano

README.txt

Già fatto! Questa sarà una descrizione dettagliata del nostro progetto che ci permetterà di ricordarci ad esempio come installare e runnare l'applicativo che stiamo sviluppando

.gitignore

Dobbiamo crearlo! Questo file ci permette di specificare al nostro sfw di versioning quali file della nostra repository ignorare. Tutto quello che git non andrà a tracciare rimarrà in locale sul nostro computer e attenzione, di conseguenza, tutte le modifiche che subiranno quei file nel tempo non saranno recuperabili.

Esercitazione 1 - remote & local

Per sperimentare il funzionamento di **.gitignore**
(ovvero tutto quello vorrete venga ignorato dal nostro software di versioning, git)

creiamo una cartella chiamata assets
all'interno di assets creiamo due cartelle
css → vim css.txt 'Sono un file di stile'
lib → vim lib.txt

Provate nuovamente un **git status**!

Noterete che git non sta ancora tenendo traccia di nessuno dei file che abbiamo dentro la nostra repository!

Esercitazione 1 - remote & local

Detto ciò creiamo quindi nella *root* della nostra repo il *.gitignore* e diciamogli di ignorare tutto quello che si trova dentro la cartella *lib*

```
vim .gitignore
```

Provate nuovamente un **git status**!

Esercitazione 1 - remote & local

Ora che abbiamo definito cosa vogliamo che git tenga traccia dobbiamo dirgli di farlo! Come? Così

git add .

git → specifico che sto invocare un comando del sftw di versioning git

add → aggiungi/segui/track

. → indica 'nell posizione in cui mi trovo' e quindi tutto quello che c'è da lì all'interno dell'albero del progetto

Controlliamo lo stato con git status e git vi risponderà mostrandovi tutti i file che ha iniziato a seguire.

git status

Esercitazione 1

*Tracciare i file della nostra
repo*

ARCHIVED

Esercitazione 1 - remote & local

Non ci resta che caricare tutti i nostri file, per fare ciò dovremo

associare al caricamento un messaggio utile

git commit -am “*messaggio*”

ed eseguire il caricamento

git push

-u → da ora in poi usa il branch specificato come def.

origin → specifica destinazione

master → branch di destinazione

Esercitazione 1

*Pushare (uploadare)
online i file di cui s
tiamo tenendo traccia*

ARCHIVED

Esercitazione 1

Andiamo tutti su Github e vediamo come effettivamente tutti i file che abbiamo trackato si trovano nella repo.

Notiamo un paio di cose dal nostro Github

commits → vedremo la presenza del nostro messaggio inserito prima di pushare i file

Ora proviamo a rifare tutto da soli e cerchiamo di arrivare allo stesso risultato, ovvero con una nuova repository, con primo commit e push

creiamo repository **repo_vostroCognome_2...**

Molto bene, siamo riusciti tutti a creare una nuova repo, con dei file, agganciarla ad una repo remota e fare il primo commit e push?

**Se non siete in pari
aiutiamoci e chiedete!**

Recap

Come ultima cosa vi ricordate cosa abbiamo fatto ieri!?

git push

- u → da ora in poi usa il branch specificato come def.
- origin → specifica destinazione
- master → branch di destinazione

Recap

- Apriamo **Github** e verifichiamo di essere **loggati** e **esploriamo** la nostra repository

The screenshot shows the GitHub interface for a repository named 'vagrant_test' by the user 'andreafspeziale'. The repository is described as 'My learning vagrant test repo with walkthrough in the readme file' and was updated 7 days ago. It has 27 commits, 2 branches, 0 releases, and 1 contributor. The repository is public and has 0 stars and 0 forks. The repository is categorized as 'HTML'.

The repository contains the following files:

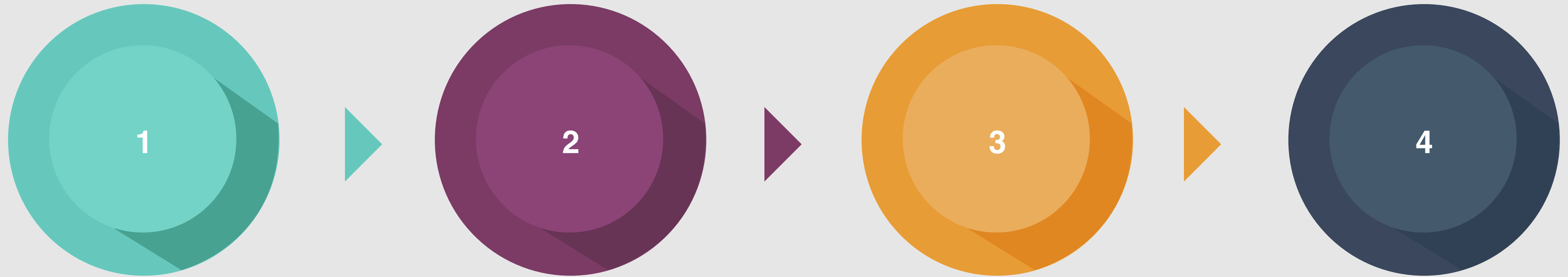
File	Commit Message	Time
roles	added vagrantfile config and script launch - hostmanager added and re...	12 days ago
.gitignore	starting walkthrough - added a test sh runned by the provisioner shell	28 days ago
README.md	added vagrantfile config and script launch - hostmanager added and re...	12 days ago
Vagrantfile	little files clean	7 days ago
index.html	added apache reload task - and index page	14 days ago
info.php	added php installation and php info page sample	16 days ago
playbook.yml	added apache reload task - and index page	14 days ago
test.sh	starting walkthrough - added a test sh runned by the provisioner shell	28 days ago

The repository is categorized as 'HTML' and has 1 star and 1 fork.

Esercitazione 2



Esercitazione 2 - obiettivi



Apportare nuove modifiche
+ **commitarle** e **pusharle**
+ **aggiunta e rimozione** file

**Utilizzare il sistema di
versioning (1)**, quindi dalla
versione attuale tornare
a una precedente e
continuare da lì

**Utilizzare il sistema di
versioning (2)**
Branching e Git Flow.
Creazione nuovi
branch e merge tra
gli stessi

Lavoro in Team
Eseguire i comandi imparati
ed applicarli in coppia

Esercitazione 2 - add a file

Spostiamoci nella root della repository e creiamo 2 nuovi file **index.html** e sotto assets/css/**style.css**

Lanciamo sempre d'approccio un
git status

git si accorgerà che ha un file non tracciato e a questo punto per aggiungerlo abbiamo 2 possibilità

git add nomeFile (oppure *git add path/nomeFile*)

oppure

git add .

Differenza!?

Esercitazione 2 - remove a file

Rimuoviamo dalla nostra repository il file **css.txt**

Lanciamo sempre d'approccio un
git status

git si accorgerà che è avvenuto un cambiamento

git commit -am “index html added - css.txt removed” (oppure *git add path/nomeFile*)

e

git push

Esercitazione 2

ADD

REMOVE

EDIT

COMMIT e PUSH













ARCHIVED








Esercitazione 2 - commit (versioning)

- Apriamo **Github**, verifichiamo di essere **loggati**, **esploriamo** la nostra repository e clicchiamo su **commit**

Branch: **master** ▼

Commits on Nov 4, 2015

	readme walk pagination updated andrea speziale committed 5 days ago		87bde49	
	readme walk pagination updated andrea speziale committed 5 days ago		75c8531	
	readme walk pagination updated andrea speziale committed 5 days ago		e1e48bb	
	starting walkthrough - added a test sh runned by the provisioner shell andrea speziale committed 5 days ago		e64ad0b	

Esercitazione 2 - commit (versioning)

little files clean

master

andrea speziale committed 7 days ago

1 parent [ca1c529](#)

commit [36575d8ce52dbb79ea3c3f6c3f8187ae83c50d84](#)

[Browse files](#)

Showing **2 changed files** with **3 additions** and **11 deletions**.

Unified

Split

6 Vagrantfile

[View](#)

...	...	@@ -1,7 +1,7 @@
1	1	# Run on vagrant up command
2	2	system(")
3	3	if [\${ARGV[0]} = 'up']; then
4		- echo 'You have just run 'vagrant up''
	4	+ echo 'You have just run vagrant up'
5	5	fi
6	6)
7	7	
		@@ -13,7 +13,7 @@ system(")
13	13	require 'yaml'
14	14	
15	15	current_dir = File.dirname(File.expand_path(__FILE__))
16		-configs = YAML.load_file("#{current_dir}/vagrantfile_config.yaml")
	16	+configs = YAML.load_file("#{current_dir}/vagrantfile_config.yml")
17	17	vagrant_config = configs['configs'][configs['configs']['use']]

Esercitazione 2 - edit again a file

Spostiamoci nuovamente nella cartella assets/lib e cambiamo il contenuto del file lib in

“Culo”

Lanciamo sempre d’approccio un
git status

git si accorgerà che è avvenuto un cambiamento

git commit -am “lib.txt content updated”

next

git push

Esercitazione 2 - commit (versioning)

Andiamo su **Github** guardiamo tutti l'ultimo commit e diciamo insieme

OH NO LIB.TXT è SBAGLIATO ERA GIUSTO PRIMA!

Torniamo ad una vecchia versione della nostra repository con il comando

git checkout <CODICE VERSIONE>

Lanciamo sempre d'approccio un

git status

A questo punto potrete notare che ci troviamo nella versione della nostra repository dove il file **lib** **non aveva ancora subito modifiche**

Verifichetelo! Proviamo a modificarlo nel modo corretto!

editate il file come preferite, aggiungeteci magari un altro aggettivo

Esercitazione 2 - commit (versioning)

Editato il file

git status

Dobbiamo comminare!

git commit -am “fix on lib.txt file”

Provate a fare quindi come da prassi un

git push

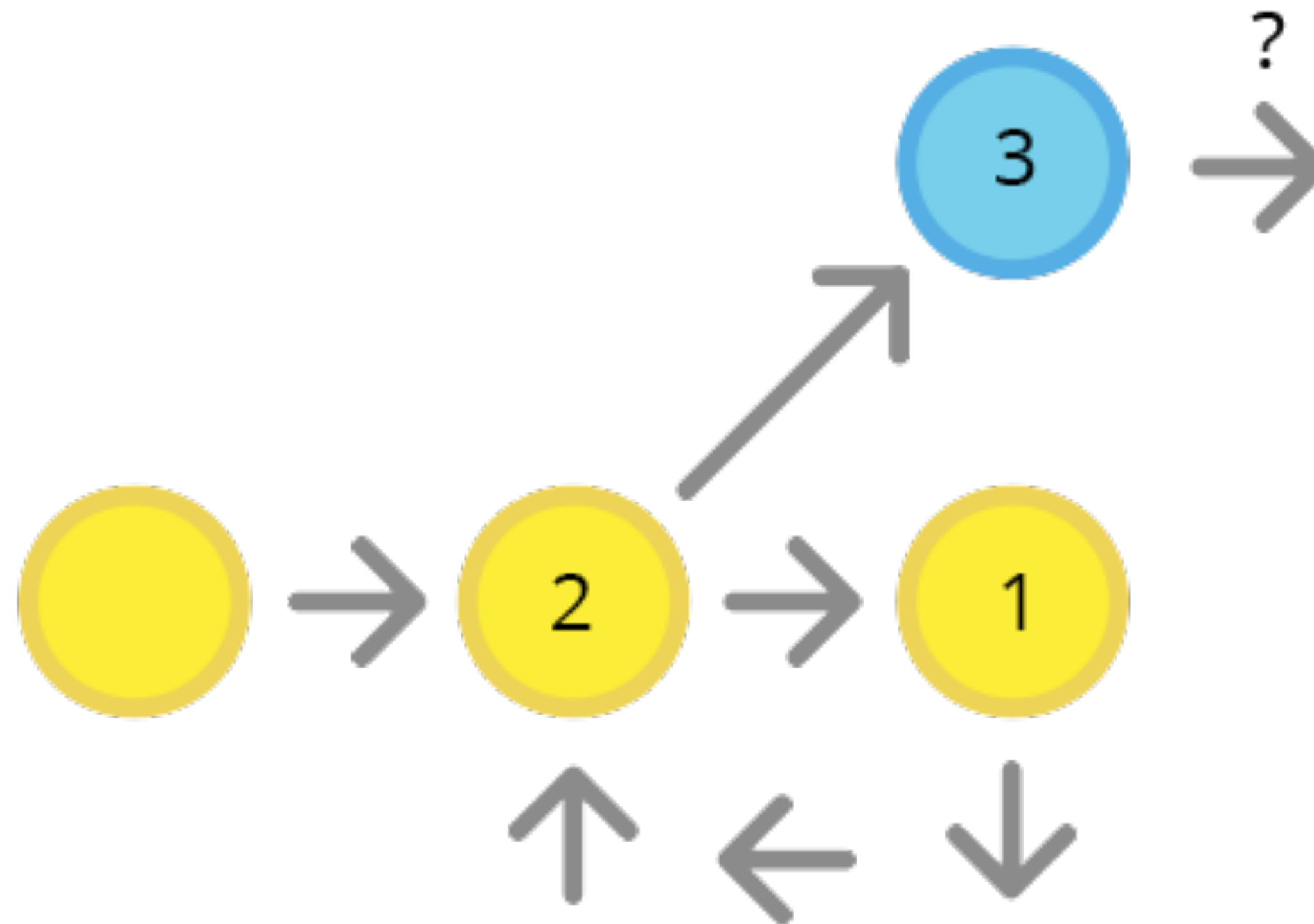
COSA SUCCEDE!?

Esercitazione 2 - commit (versioning)



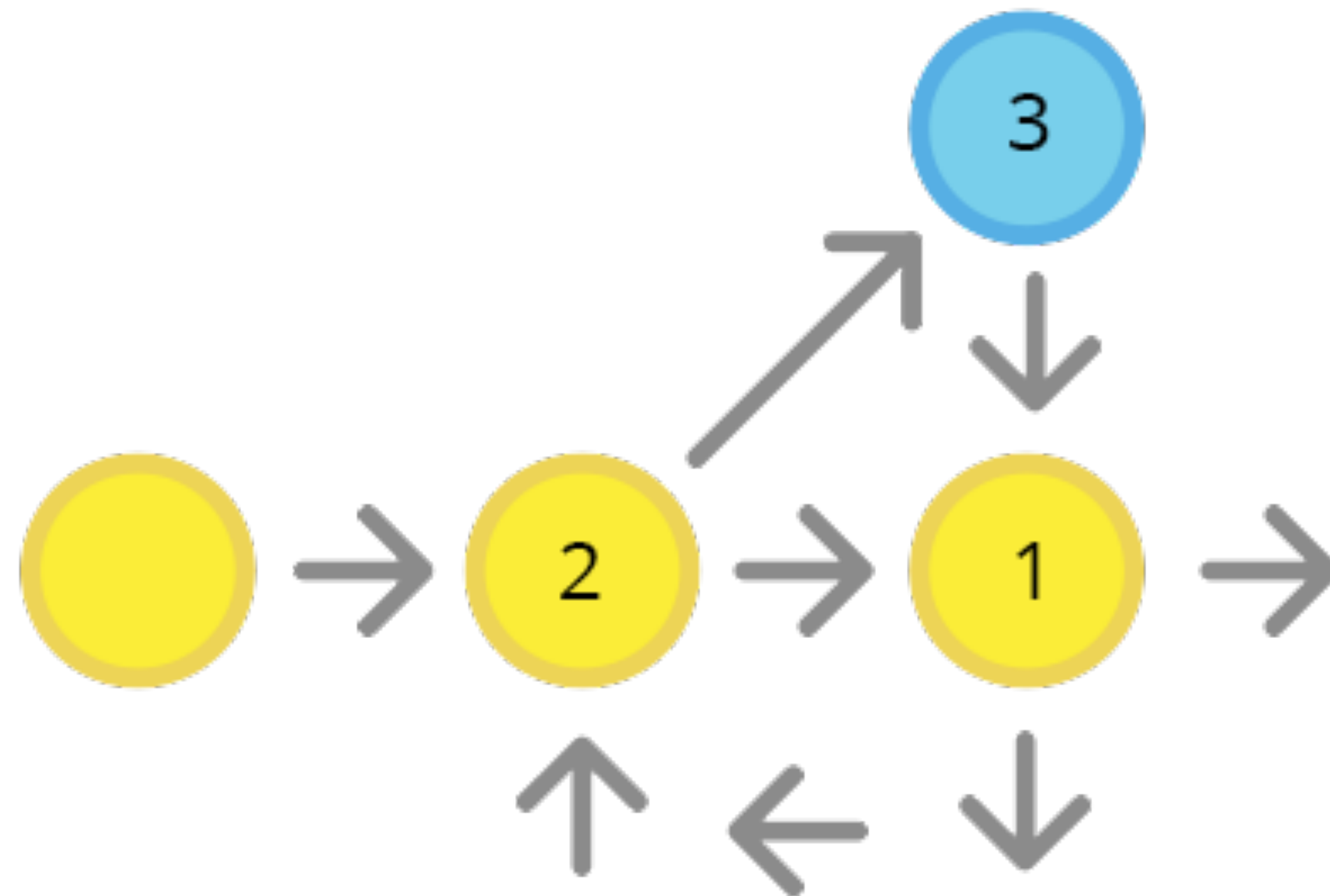
Esercitazione 2 - commit (versioning)

Non vi permetterà di fare nessun push perché dovrete specificargli in che direzione proseguire!



Esercitazione 2 - commit (versioning)

Caso 1 gli diremo di proseguire direttamente da dove avevamo lasciato



Esercitazione 2 - commit (versioning)

Caso 1 per fare cio' dovremo spostarci al punto 1 e dirgli
“punto 1 prenditi in pancia il punto 3”

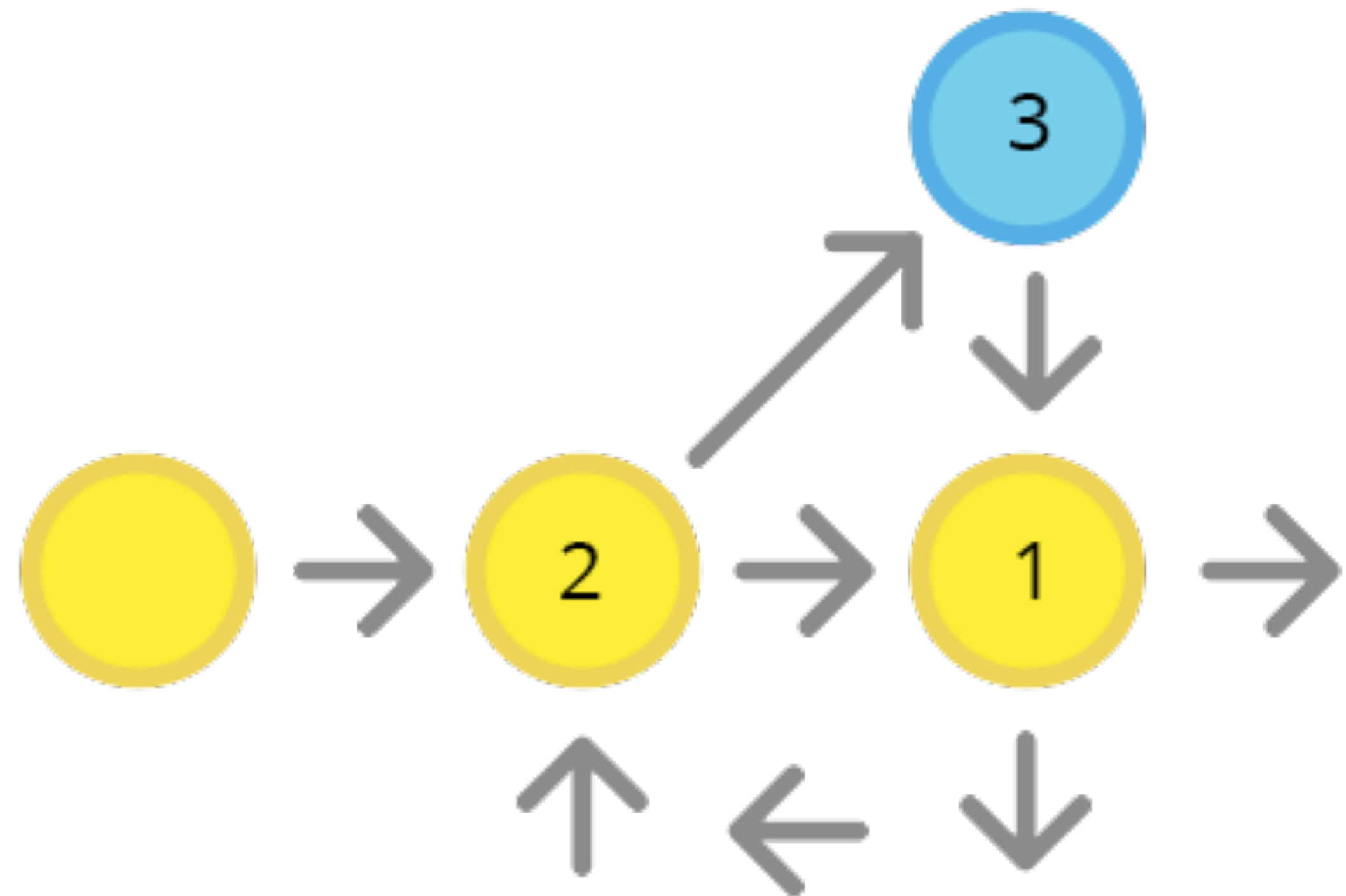
Spostiamoci sul punto 1, quindi la nostra
HEAD, quindi **master**

git checkout master

git ci ricorderà che stiamo lasciando indietro
dei **commit ovvero** delle **modifiche** che ha
tracciato targate con un certo **codice** o **tag**

A questo punto essendoci spostati su **master**
dovremo dirgli di prenderti in pancia le
modifiche con quel tag

git merge <codice versione // tag>



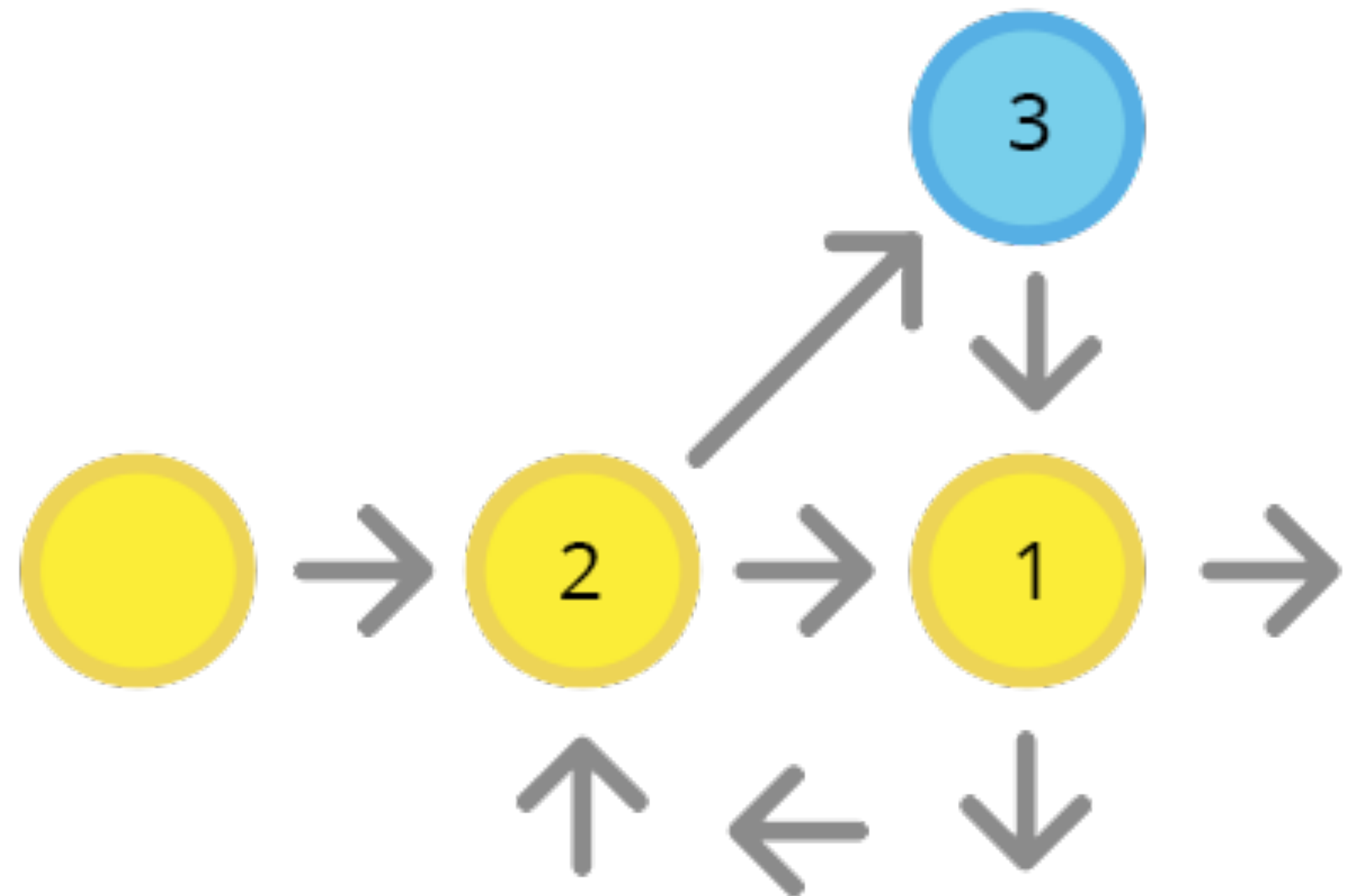
Esercitazione 2 - commit (versioning)

A questo punto come da prassi **committiamo**

git commit -am "merge and code update"

git push

Andiamo tutti su Github e noterete che effettivamente vi ritroverete in linea con la vostra repository aggiornata

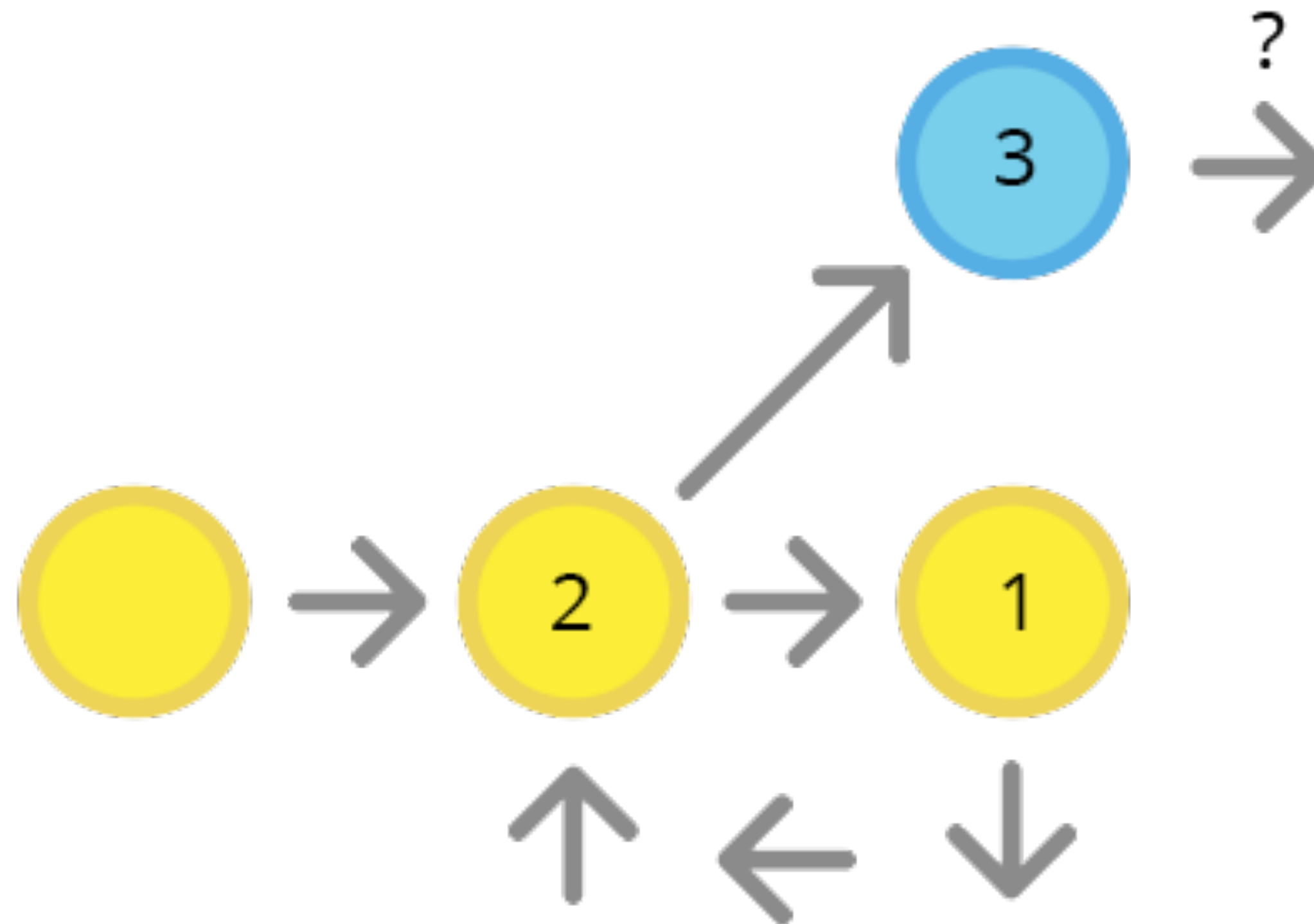


Esercitazione 2

Utilizzare il
SISTEMA DI VERSIONING
(1)
ARCHIVED

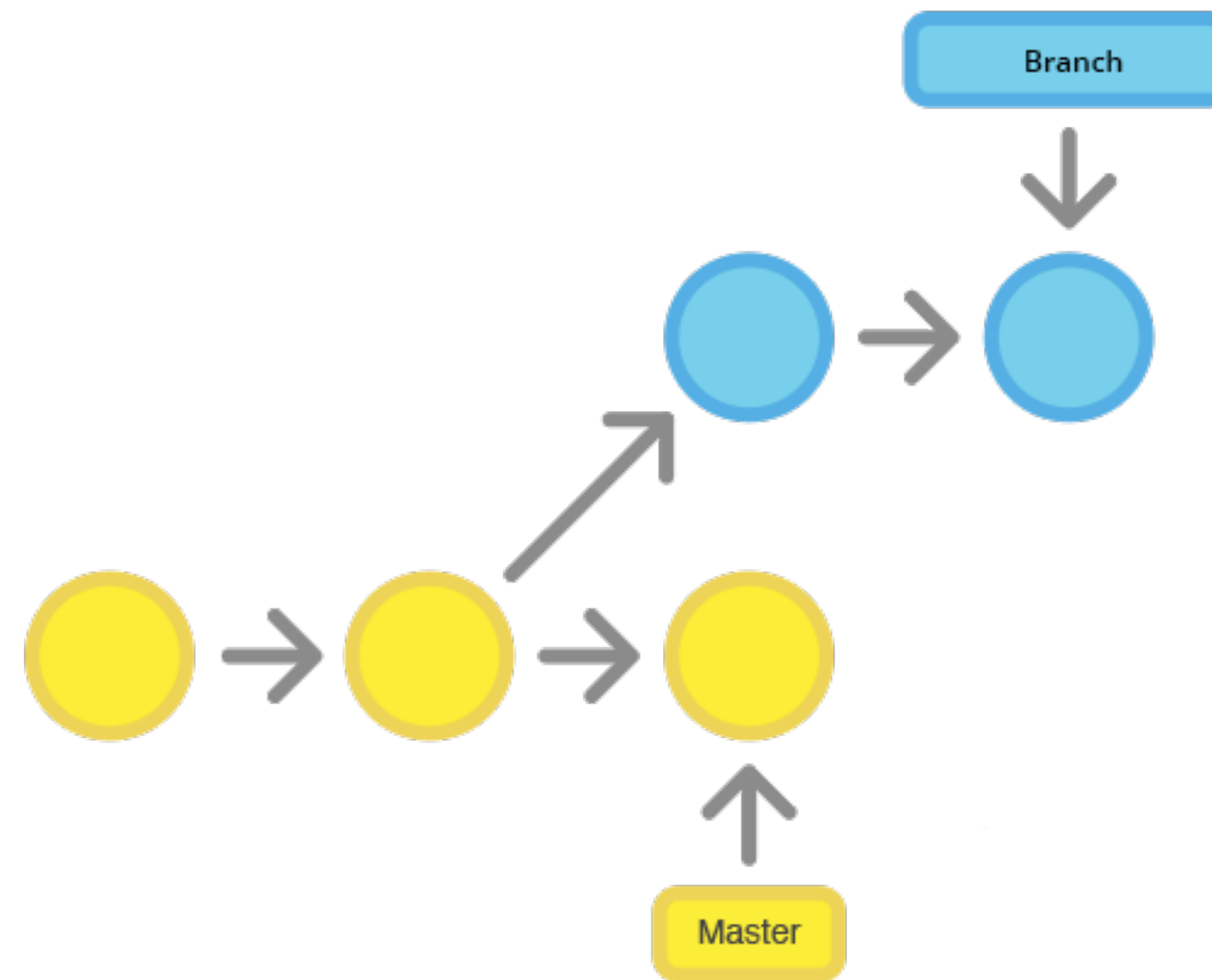
Esercitazione 2 - branching

*Vi ricordate quando **git non** vi permetteva di fare alcun push perché dovevate specificargli in che direzione proseguire?*



Esercitazione 2 - branching

Caso 2 invece di proseguire da dove avevamo lasciato, proseguiremo su un'altra strada, un nuovo ramo ovvero **branch**



Esercitazione 2 - branching

Possiamo creare un merge sia dal nostro ultimo commit, sia da un commit precedente

Noi proveremo a farlo dal nostro ultimo commit quindi, verifichiamo lo stato della nostra repo

git status

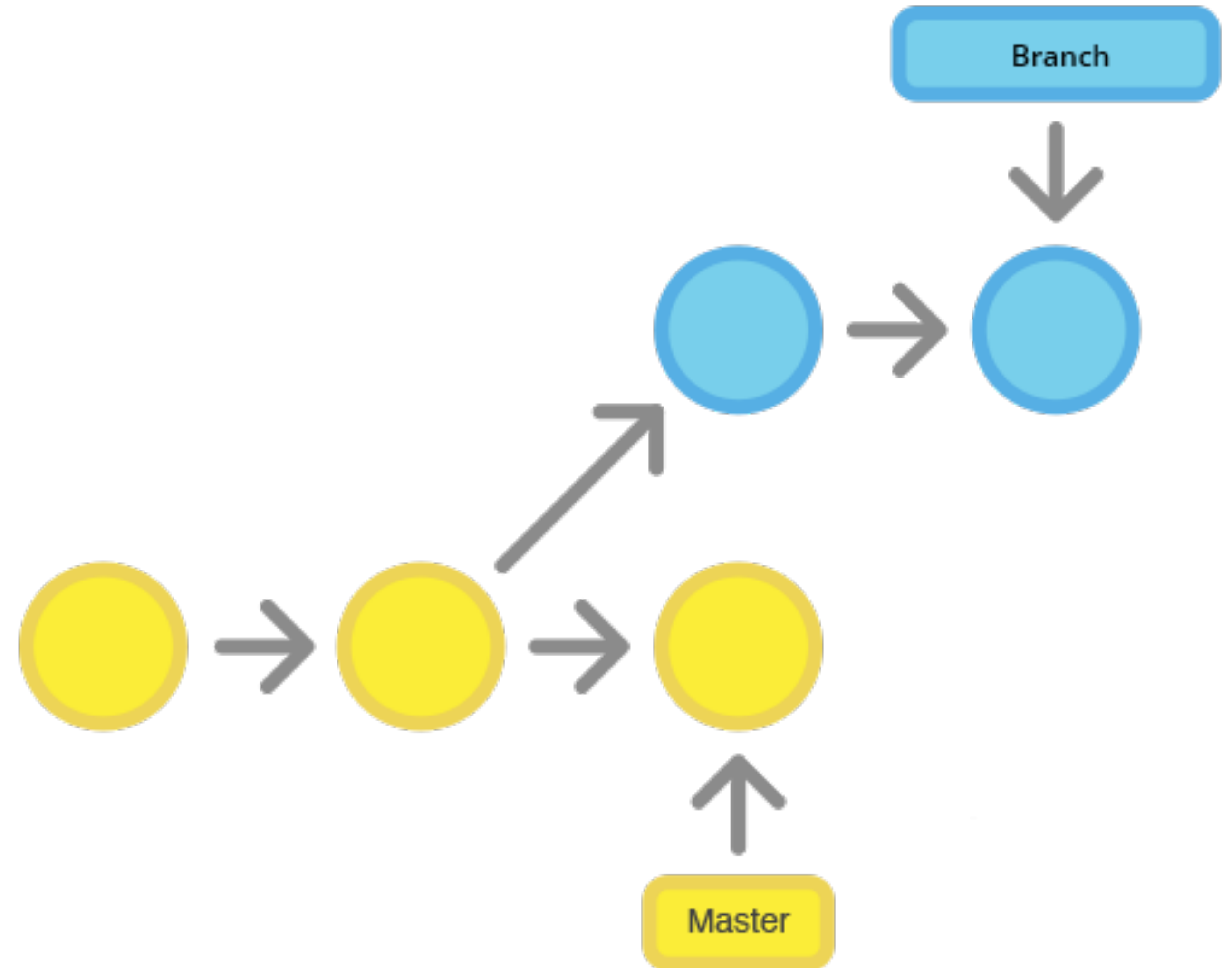
creiamo un nuovo branch con il seguente cmd

git checkout -b nameOfTheBranch

effettuiamo una modifica a piacere

git commit -am "msg di modifica"
git push -u origin nameOfTheBranch

Da questo momento tutte le modifiche che farete verranno pusshate sul nuovo **branch**!



Esercitazione 2 - branching

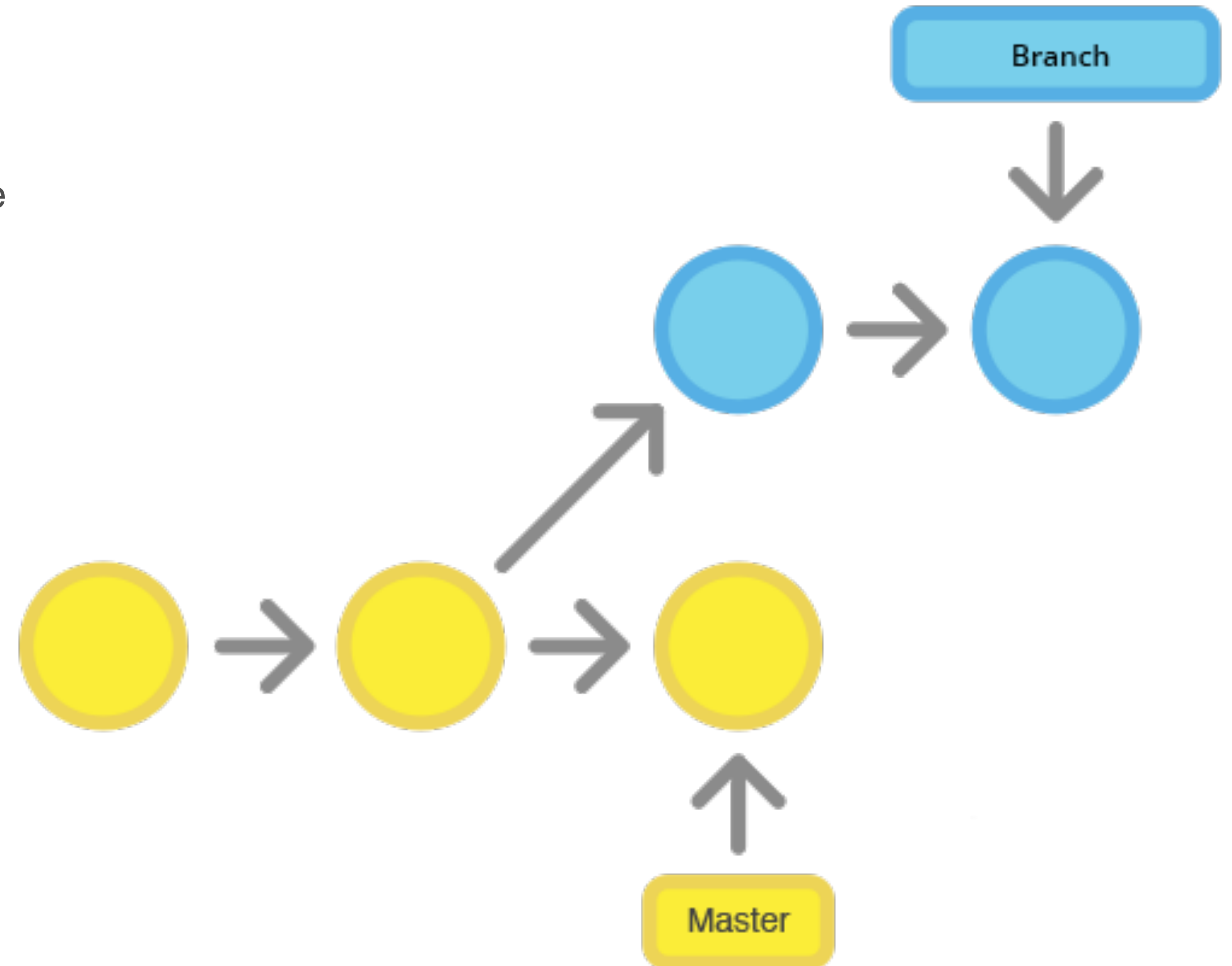
Provate ad effettuare altre modifiche ed aggiungere nuovi file e a pusshare il tutto sulla repository!

Noterete direttamente dalla pagina della repository che le nuove modifiche e i file che avete aggiunto si trovano solamente esplorando il nuovo branch

Finito lo sviluppo sul nuovo branch sarà nostro interesse riportare il tutto sul merge principale, il **master**!

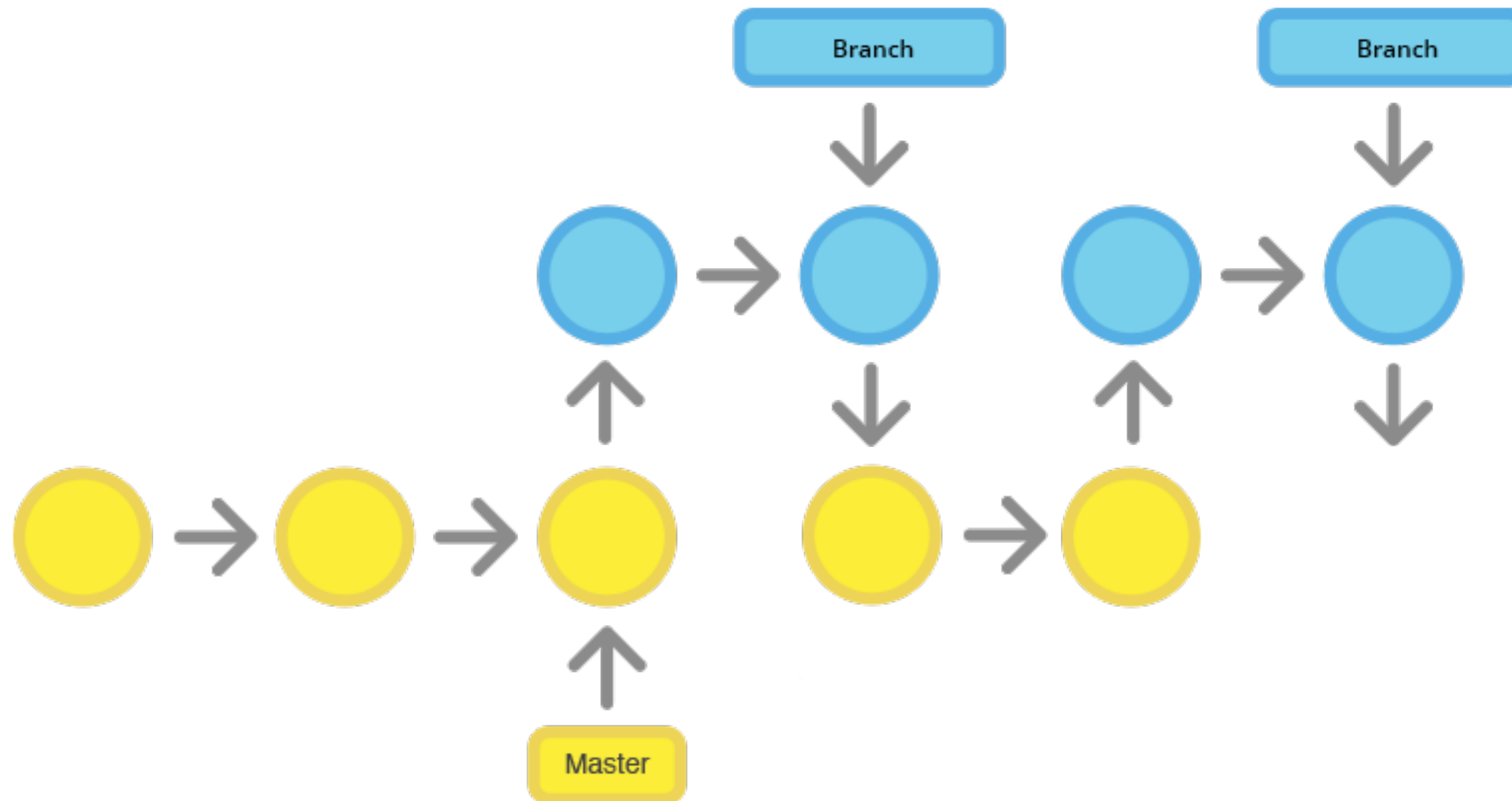
Per fare ciò, come abbiamo già fatto prima, spostiamoci sul branch **master**

```
git checkout master  
git merge nomeDelVostroBranch
```



Esercitazione 2 - git flow

Git flow non è altro che una **metodologia di branching**



Esercitazione 2

Utilizzare il
SISTEMA DI VERSIONING
(2)
ARCHIVED

Esercitazione 2

Lavorare in
TEAM

Esercitazione 2 - lavorare in team



Esercitazione 2 - lavorare in team

Uno dei due componenti del team, vada sulla propria repository remota (github) e dalle impostazioni della stessa aggiunga il vicino di banco come collaboratore dandogli privilegi sia di lettura che scrittura!

Il fortunato ospitato dovrà farsi una copia locale della repository, quindi navigherà via linea di comando fino alla sua cartella di workspace e eseguirà il seguente comando

git clone *git@github.com:compagnoUsername/nameOfTheOnlineRepo.git*

Lanciate nella vostra workspace un **ll** e noterete di avere la repository del vostro vicino sul vostro computer, esploratela!

Una volta esplorata, provate a modificare, editare o eliminare un file, committate la modifica, e pusshate

Esercitazione 2 - lavorare in team

Ora il secondo membro del team per vedersi le modifiche del compagno in locale, dovrà “scaricarsi” le modifiche, più tecnicamente fare un **pull**

Quindi dentro la repository eseguire un

git pull

Esplorate la repository e vi troverete le modifiche che il vostro compagno di team ha **commitato** e **pushato**

Quando si lavora in team, non è necessario aspettarsi l'un l'altro, fortunatamente (a contrario di hg) **git** è molto intelligente e ogni volta che farete un commit e cercherete di fare un push vi verrà automaticamente negato a meno che voi non abbiate la versione della repository aggiornata

in caso sia aggiornata non vi impedirà il push

in caso non sia aggiornata vi impedirà il push e vi chiederà di effettuare un **pull**

Esercitazione 2 - lavorare in team

**Cosa succede se i due compagni di merende
hanno modificato lo stesso identico file??**

Anche in questo caso spesso **git** ci viene incontro risolvendo da solo i conflitti ed effettuando un **merge** automatico.

In caso contrario vi chiederà di risolvere manualmente il conflitto, comminare e solo allora vi permetterà di effettuare un **push**

Mettetevi in linea col vostro compagno di squadra e provate a modificare lo stesso file, chi finirà prima potrà **pusshare**, il secondo dovrà **pullare**.

In questo modo verificheremo se git riuscirà a risolvere il conflitto in automatico o meno!

In caso negativo lo faremo noi a mano eliminando i pezzi segnalatici da **git** che riteniamo non ci servano

Esercitazione 2 - lavorare in team

Git tipicamente ci segnala i conflitti e ce li evidenzia nel nostro codice inserendo tipicamente della punteggiatura

```
1 <<<<<<< HEAD
2 culooooooooooooooooooooo
3 =====
4 fuuuuck
5 >>>>>>> 0f04caa
```

Esercitazione 2 - lavorare in team

<<<<

HEAD: è il contenuto della versione della repository remota

====

Vostro contenuto in locale

====

Molto semplicemente lasciate solo la parte di vostro interesse, **commitate e pusshate!**

Esercitazione 2

LAVORO IN TEAM
ARCHIVED