bticino

Corso Git

Corso Git

bticino

- Git: vedremo come utilizzarlo da linea di comando
- Esistono molti ottimi client grafici, sia stand-alone che integrati nelle maggiori piattaforme di sviluppo (Eclipse, IntelliJIdea, NetBeans...)
- Dopo aver compreso l'utilizzo dei comandi principali, sarà più semplice utilizzare un client grafico di proprio gusto

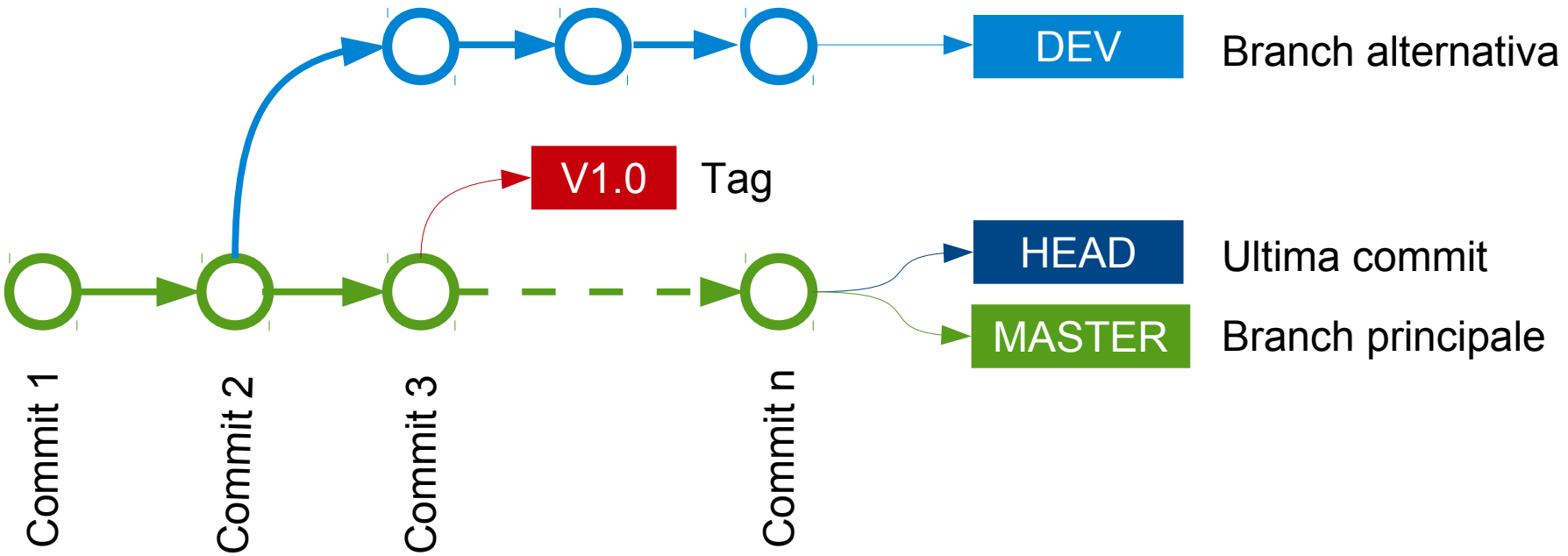
Corso Git

- Git: impostazioni preliminari
- Per spiegare tutte le possibilità di configurazione di Git occorrerebbe più tempo (e pazienza). Le essenziali, per iniziare, sono:
- git config --global user.name "Tuo Nome"
- git config --global user.email "tuo_email@bticino.it"

In questo modo Git conosce il nome di chi ha creato una commit

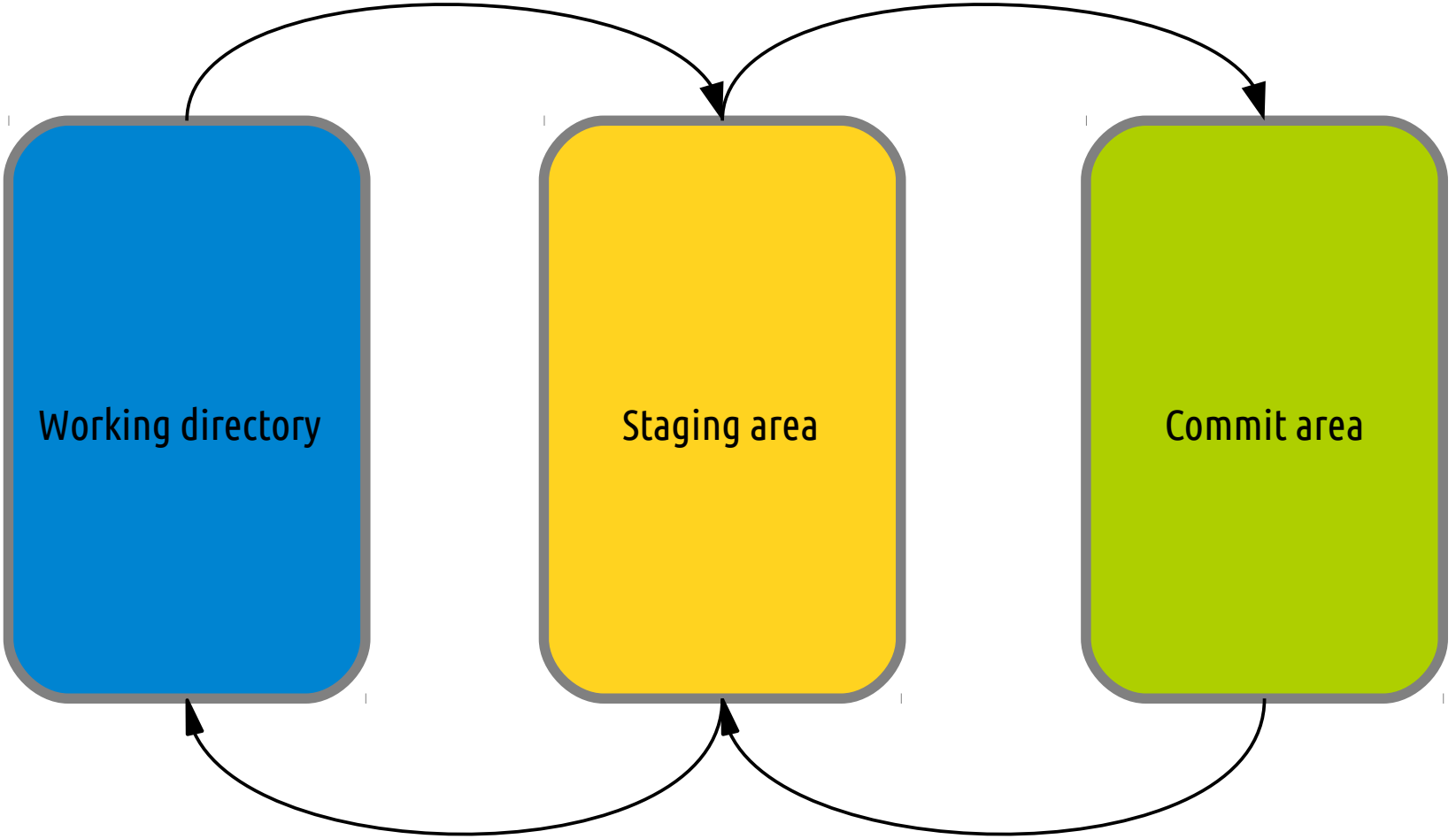
Corso Git

Terminologia



Corso Git

Aree di lavoro



Corso Git

bticino

SandBox

http://10.31.61.8/webTerminal

Corso Git

- Primo passo: creiamo un nuovo progetto
    - Creiamo una directory($ mkdir ciaomondo)
    - Posizioniamoci lì ($ cd ciaomondo)
    - Creiamo un file (anche vuoto), chiamandolo hello.txt ($touch hello.txt)
- Secondo passo: Creiamo il repository Git
    - $git init

```
$ git init
Initialized empty Git repository in /home/<user>/ciaomondo/.git/
```

Corso Git

bticino

- Terzo passo: aggiungiamo il file al repository
  - $git add hello.txt
  - $git commit -m "First Commit"

```
$ git add hello.txt

$ git commit -m "First Commit"
*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: empty ident name (for <tester@luigi-laptop.(none)>) not allowed
$
```

Corso Git

bticino

- Quarto passo: dopo esserci dichiarati, facciamo una commit e verifichiamo che cosa è successo
  - $git status

```
$ git commit -m “First commit”
[master 7c96e0f] First commit
 1 file changed, 1 insertion(+)
$ git status
# On branch master
nothing to commit, working directory clean
```

Il comando risponde dicendoci che non ci sono modifiche pendenti. Ciò significa che il repository ha tutto lo stato corrente della directory di lavoro. Non ci sono modifiche in sospeso da registrare.

Noi useremo il comando git status per continuare a verificare lo stato di allineamento tra il repository e la directory di lavoro.

Corso Git

bticino

- Modifichiamo il file hello.txt: **$<edit> hello.txt**

```
Ciao Mondo!
~
~
~
:wq
```

Verifichiamo lo stato del repository

$git status (il file è cambiato ma non è in stage)

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Corso Git

bticino

Staging: sottoponiamo le modifiche al controllo di Git, in modo da prepare una commit

$git add hello.txt

$git status

```
$ git add hello.txt
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    modified:   hello.txt
#
```

Corso Git

bticino

Staging e commit

• Modello Git: Git consente di organizzare le modifiche in modo che riflettano esattamente il lavoro svolto.

Ad esempio: ci sono tre file modificati (a.txt, b.txt e c.txt). Tutte le modifiche saranno messe in scena (on staging), ma **si desidera che i cambiamenti in a.txt e b.txt appartengano ad un singolo commit**. Le modifiche a c.txt non sono logicamente correlate alle prime, e dovrebbero essere in un commit separato . Si potrebbe fare quanto segue:

- $ git add a.txt b.txt
- $ git commit -m "Changes for a and b"

- $ git add c.txt
- $ git commit -m "Changes for c"

Corso Git

bticino

Commit

Fino ad ora abbiamo eseguito una commit passando il commento come parametro: git commit -m "Changes for a and b"

- Git consente di configurare un editor di testo agganciato alla fase di commit, che verrà aperto automaticamente per scrivere/modificare il commento.

- Per scegliere l'editor preferito, si usa il comando (per vi):
  - $ git git config –global core.editor vi

- Quindi: una commit senza parametro -m attiva un editor che permette di scrivere il commento in modo più agevole

- NB: è sempre possibile lasciare un commento vuoto (-m “”) ma non è una Best Practice...

Corso Git

bticino

Cambiamenti e non files

Git registra come entità elementari i singoli cambiamenti , e non i files modificati

- Per esempio, modifichiamo due volte il file hello.txt
  - $ <edit> hello.txt, e aggiungiamo un commento: “Creato da <user>”
  - $ git add hello.txt
- Ora facciamo una seconda modifica allo stesso file
  - $ <edit> hello.txt, e modifichiamo il commento: “Created by $user”
  - $ git status

```
$ git status
On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#     modified:   hello.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#     modified:   hello.txt
```

Corso Git

bticino

Cambiamenti e non files

- Ora facciamo una commit
  - $ git commit -m "Added a default comment"
  - $ git status

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

- Ora mettiamo di nuovo hello.txt on stage, ed eseguiamo una commit
  - $ git add hello.txt
  - $ git commit -m "Modified default comment"
  - $ git status

```
# On branch master
nothing to commit, working directory clean
```

Corso Git

bticino

Storia

Si possono visualizzare tutti i commit effettuati nel progetto

- `$ git log`

```
$ git log
commit bcc814ee0e61b46ee7be6a7c675d3777f3bb219d
Author: Luigi Talamona <luigi@talamona.org>
Date:   Fri Aug 30 16:39:37 2013 +0200

    Modified default comment

commit 385d2d8a4551d1b083d0f2aab0486c9942836953
Author: Luigi Talamona <luigi@talamona.org>
Date:   Fri Aug 30 16:28:21 2013 +0200

    Added a default comment

commit 10e8fc4f85dee3f17a9b9bb2b8826ac4a915a97f
Author: Luigi Talamona <luigi@talamona.org>
Date:   Fri Aug 30 15:30:01 2013 +0200

    First Commit
```

Corso Git

bticino

Ritornare sui propri passi

E' facile ritornare sui propri passi, usando il codice hash della commit

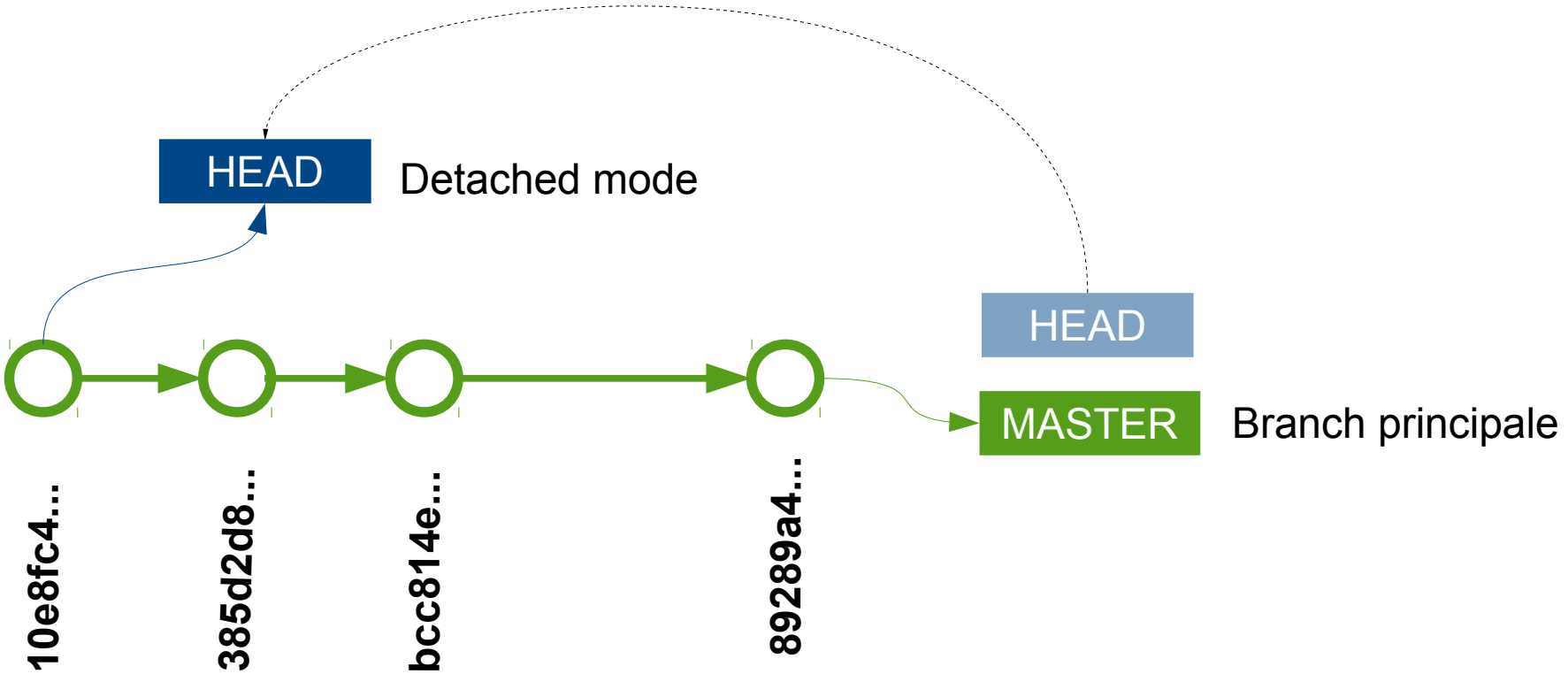
- $ git log, come prima, e scegliamo la prima commit
  10E8fc4...

- $ git **checkout** 10e8fc4

```
Note: checking out '10e8fc4f85dee3f17a9b9bb2b8826ac4a915a97f'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 10e8fc4... First Commit
```

Corso Git

Ritornare sui propri passi



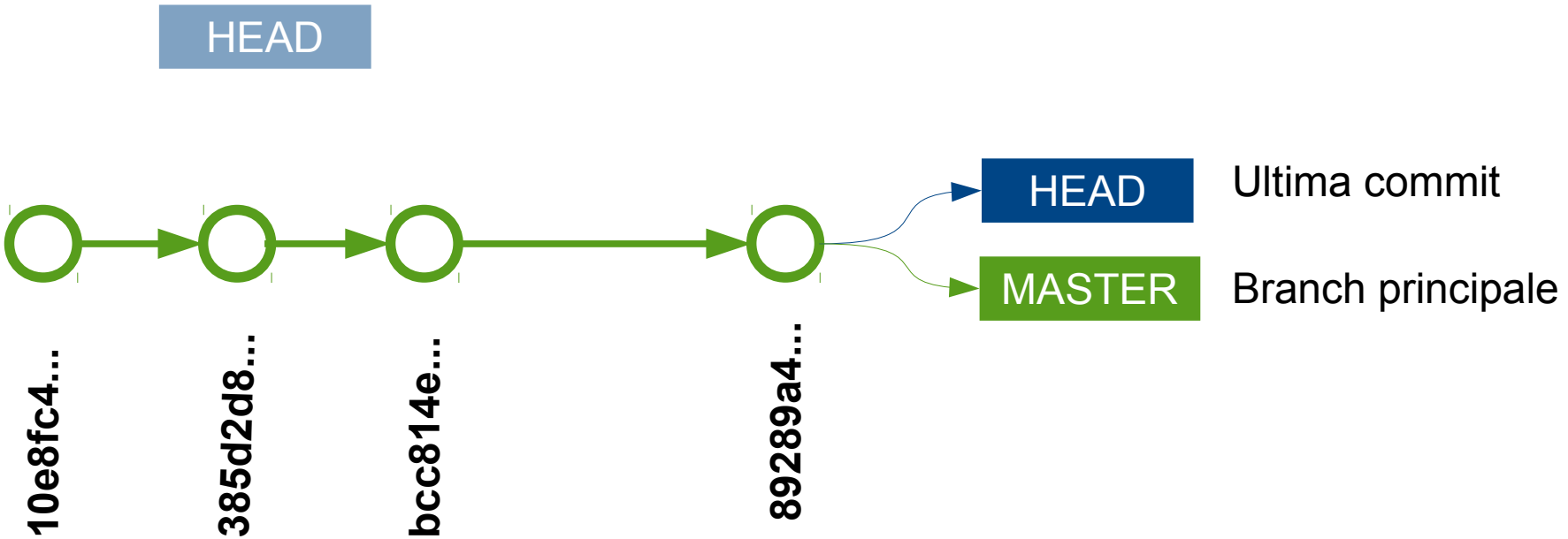
hello.txt?

Corso Git

Ritornare sui propri passi

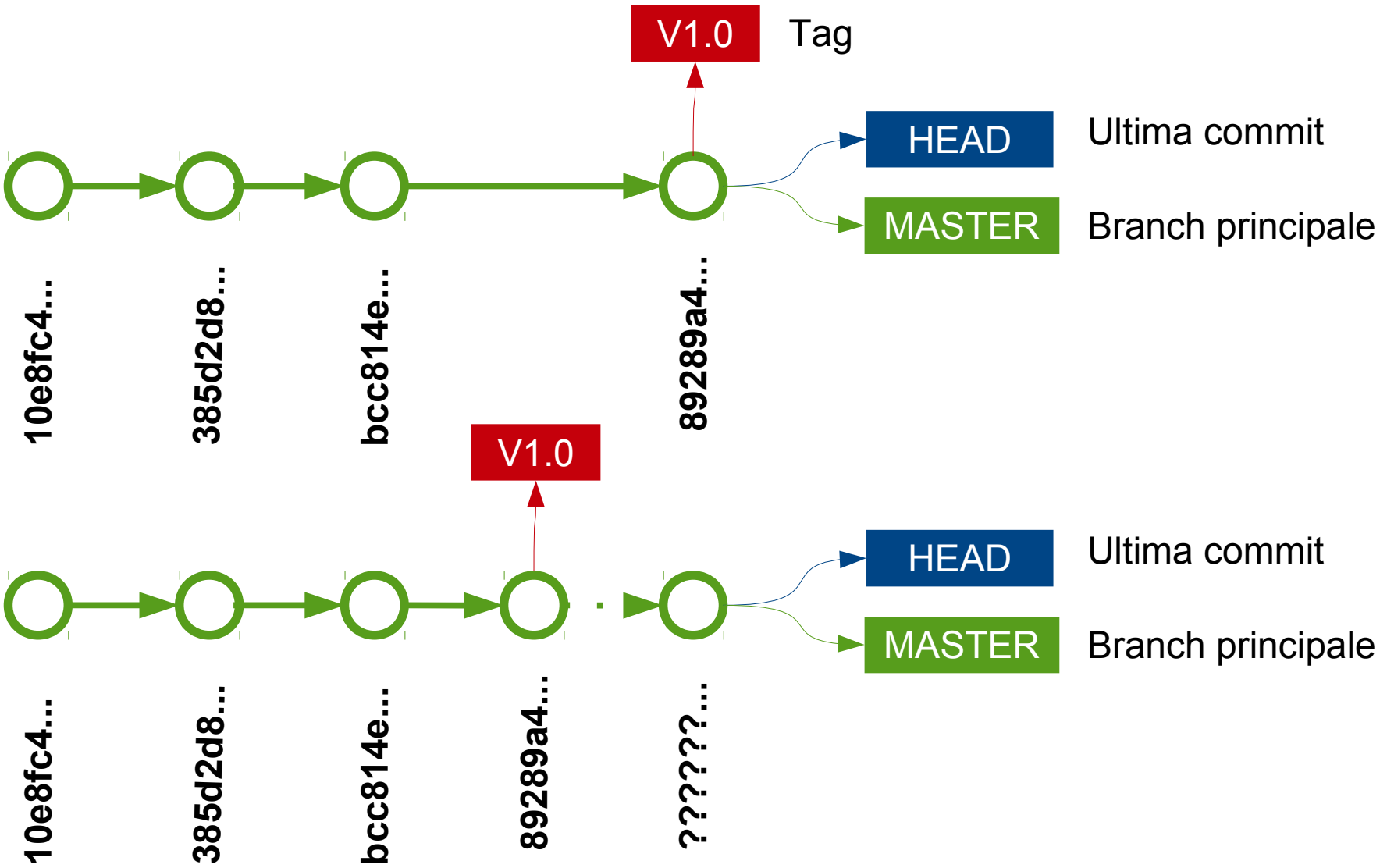
Ora, per tornare allo “stato dell'arte”

```
$ git checkout master
Previous HEAD position was 10e8fc4... First Commit
Switched to branch 'master'
$ cat hello.txt
# “Created by $USER”
Ciao Mondo
$
```



Corso Git

Etichettare i passi salienti (Tagging)



Corso Git

bticino

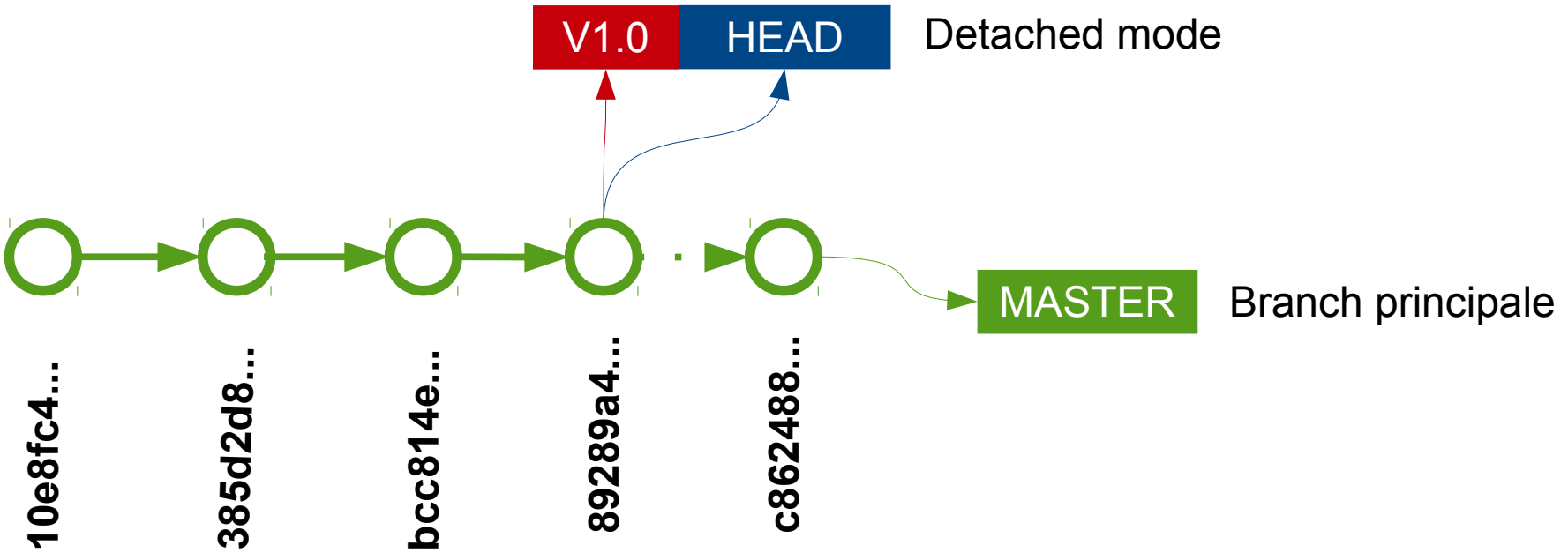
Etichettare i passi salienti (Tagging)

```
$ git tag V1.0 -m “Version 1 milestone”
$ git tag
V1.0
$ git checkout V1.0
Note: checking out 'V1.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 89289a4... User variable changed to uppercase (bug fixing)
```

Corso Git

Etichettare i passi salienti (Tagging)

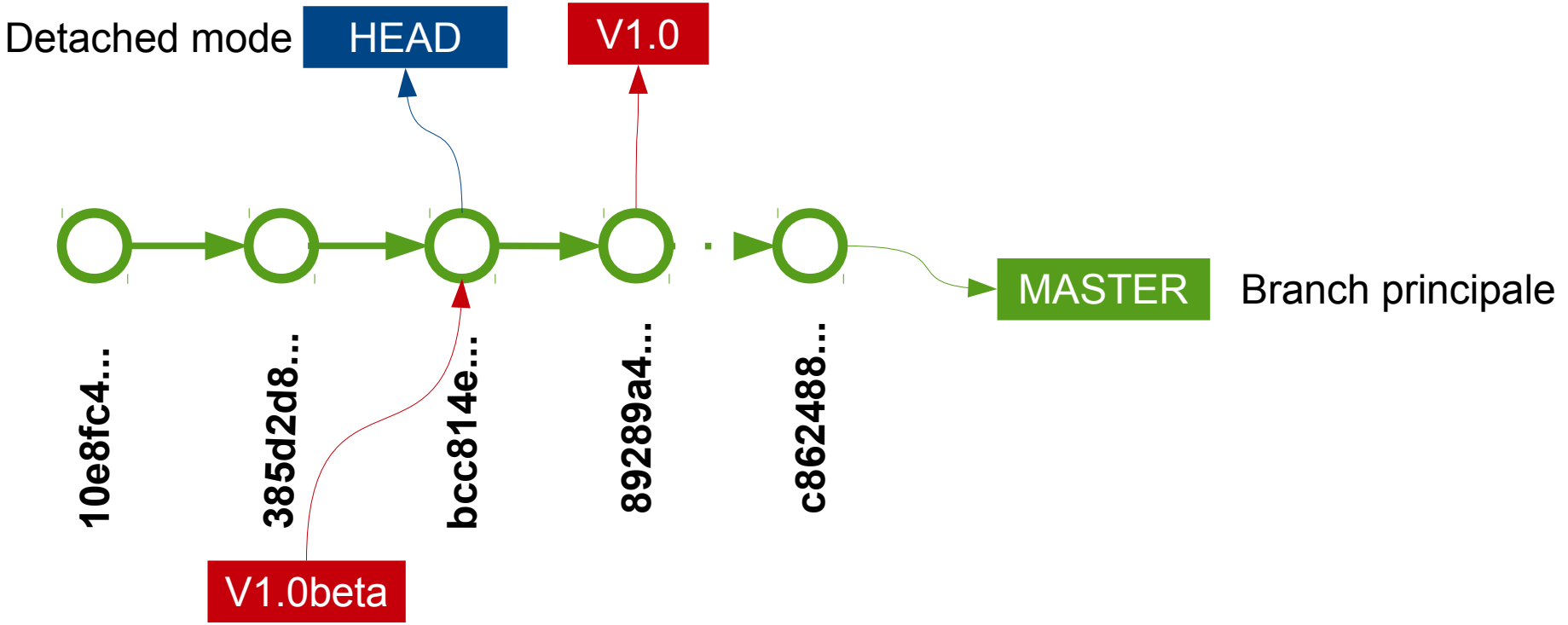
$ git checkout V1.0



Corso Git

Etichettare i passi salienti (Tagging)

$ git checkout V1.0^ (cerca il primo predecessore della commit indicata)



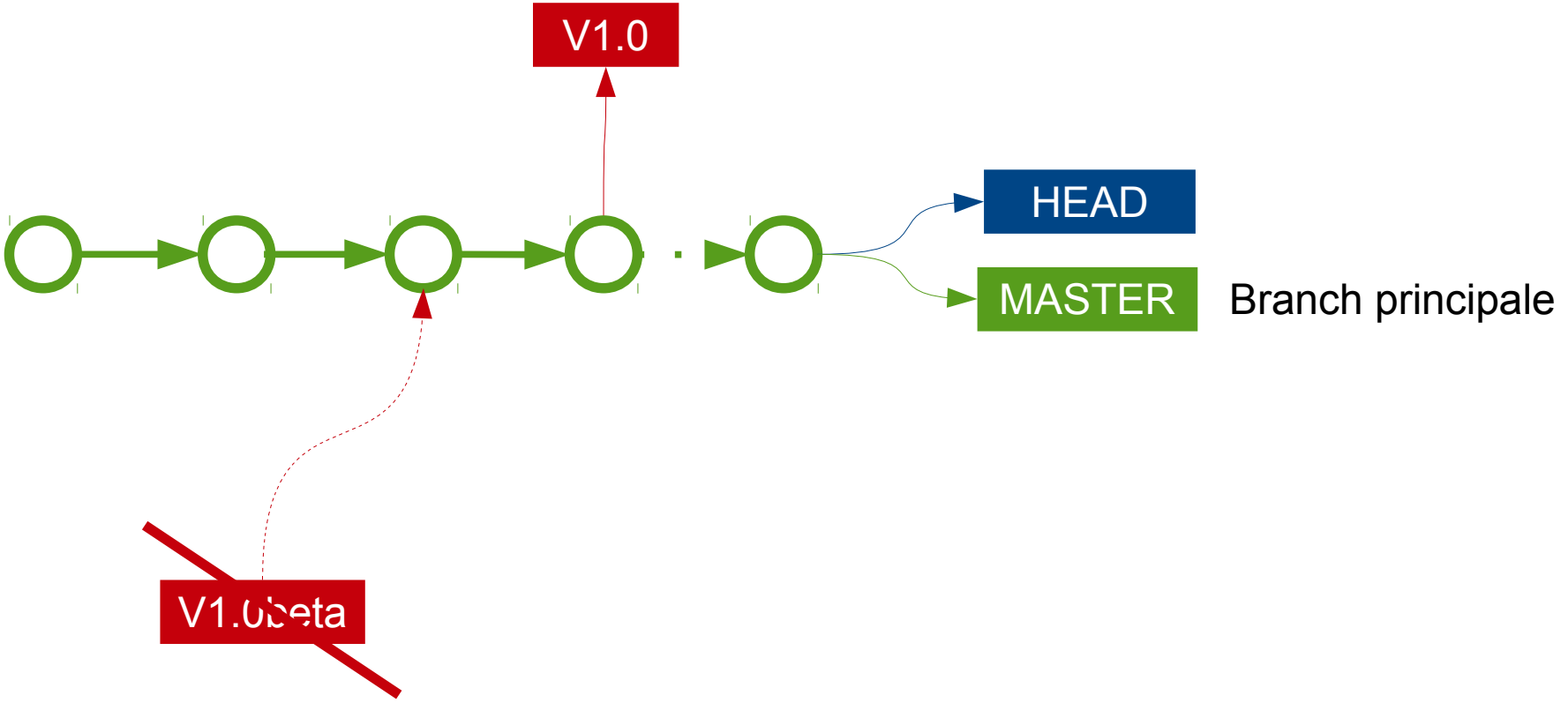
$ git tag V1.0beta
$ git tag

```
$ git tag
V1.0
V1.0beta
$
```

Corso Git

Cancellare una tag

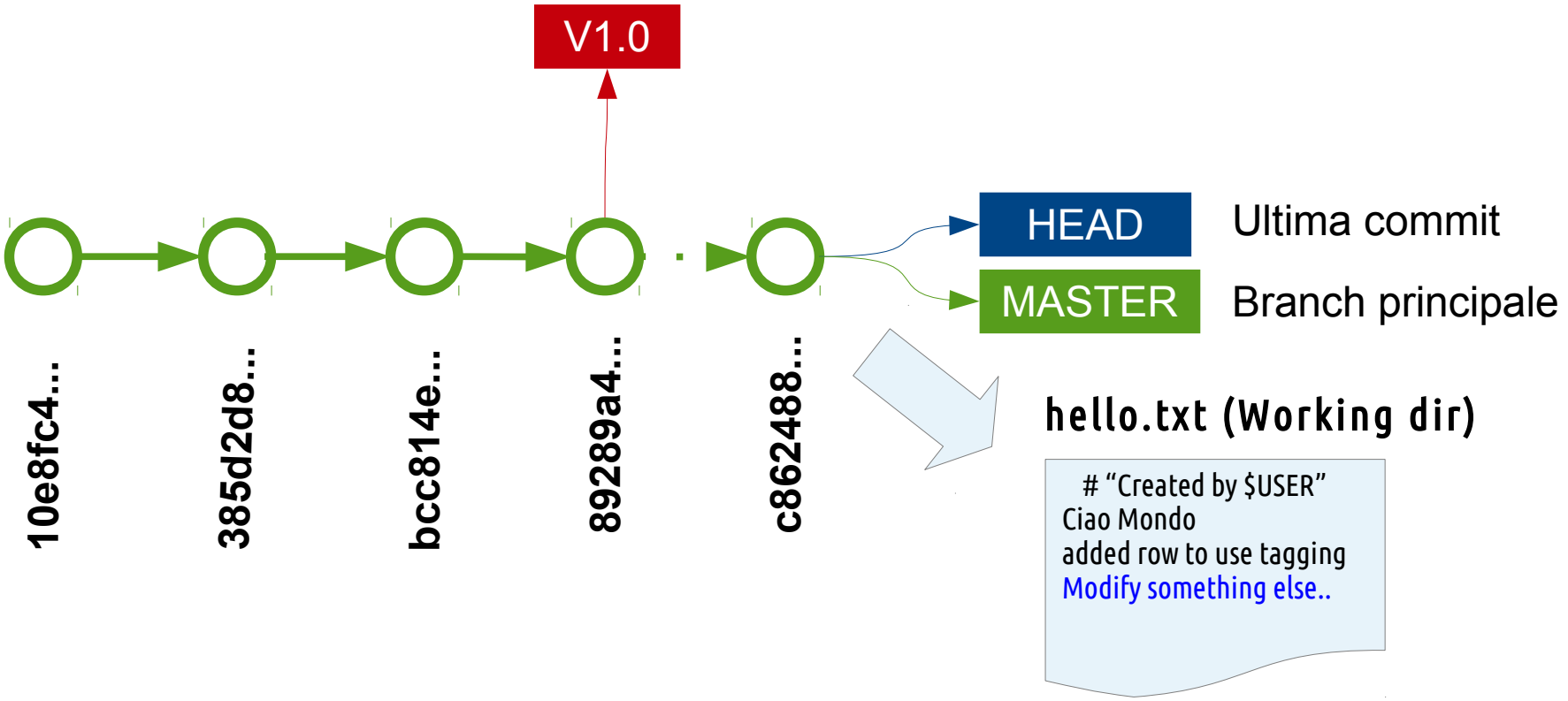
```
$ git tag –d V1.0beta
```




```
$ git tag -d V1.0beta
Deleted tag 'V1.0beta' (was ......)
$
```

Corso Git

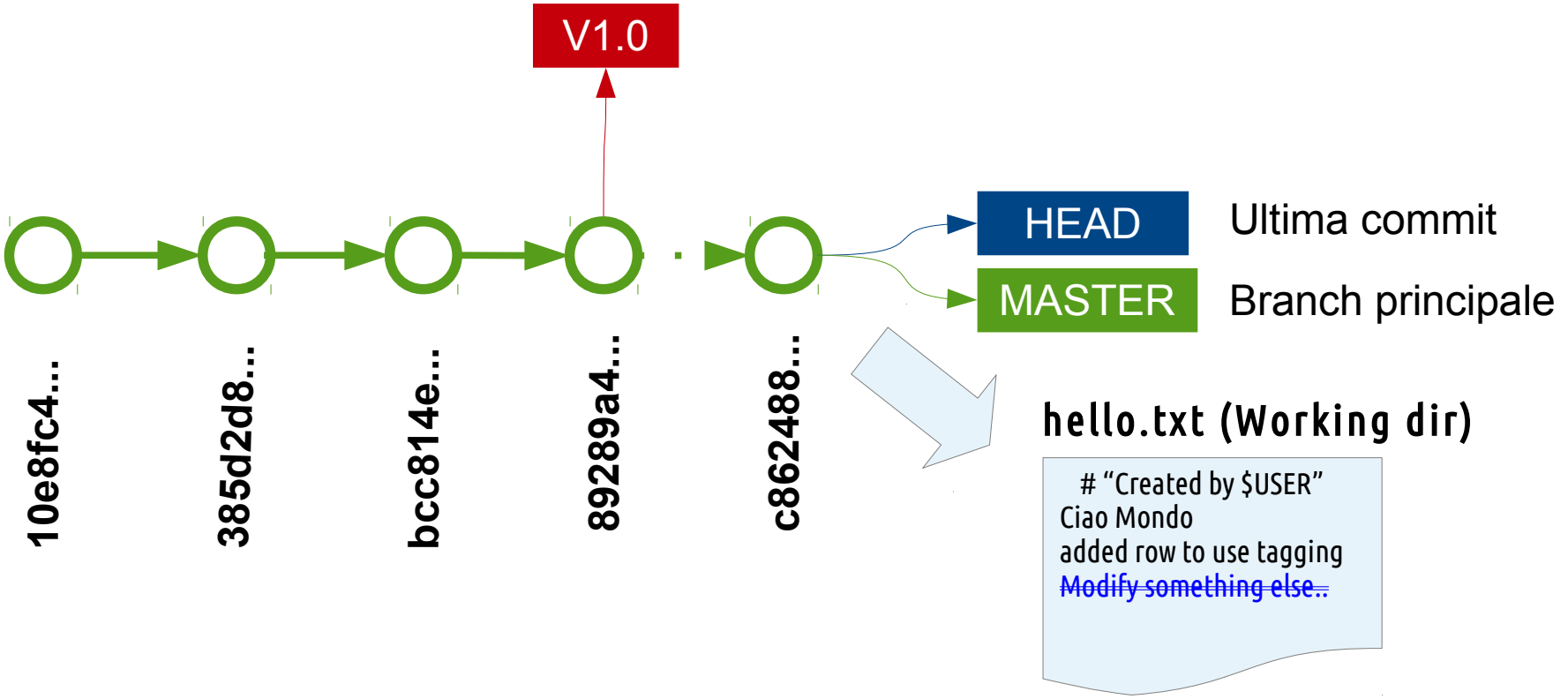
Annullare le modifiche (prima dello staging)



Corso Git

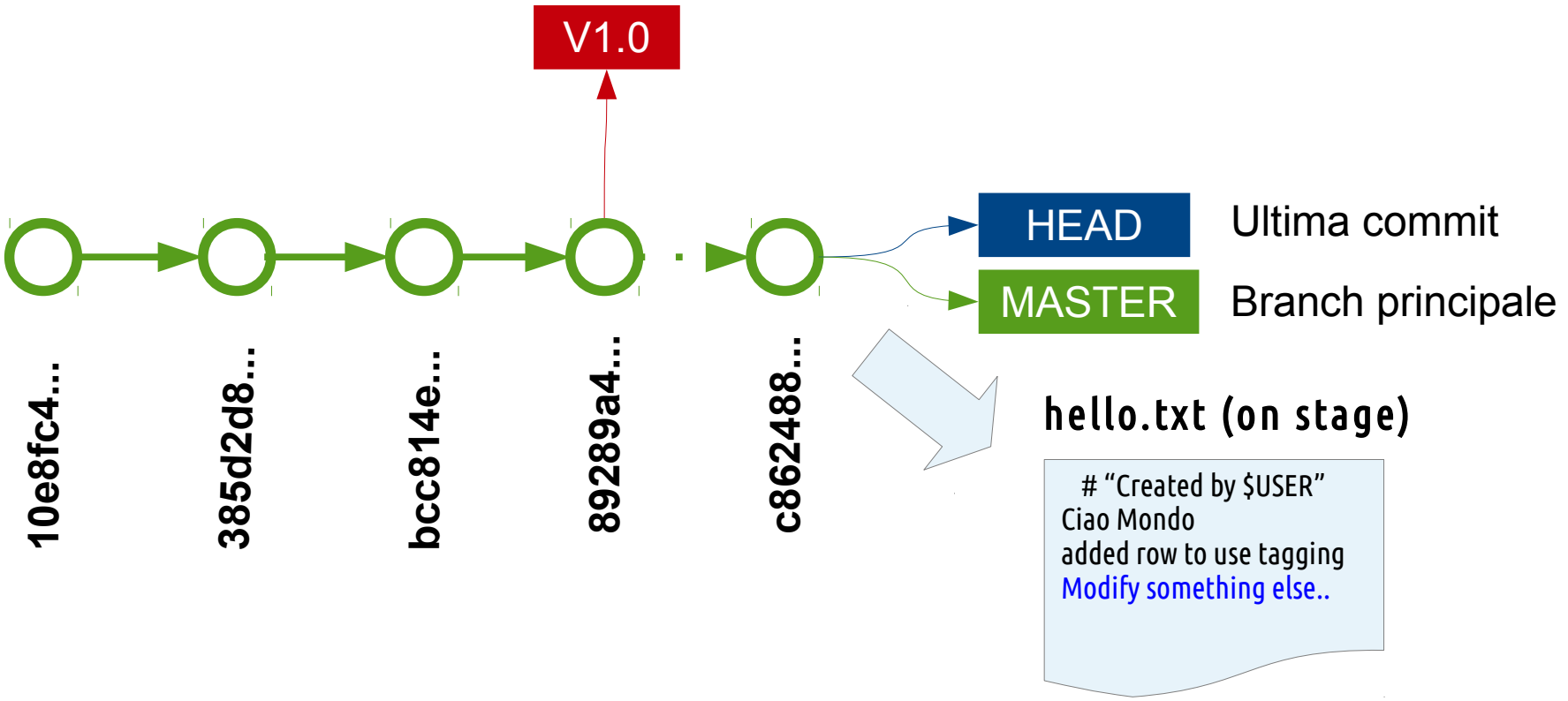
Annullare le modifiche (prima dello staging)

$ git checkout hello.txt



Corso Git

Annullare le modifiche (on stage)

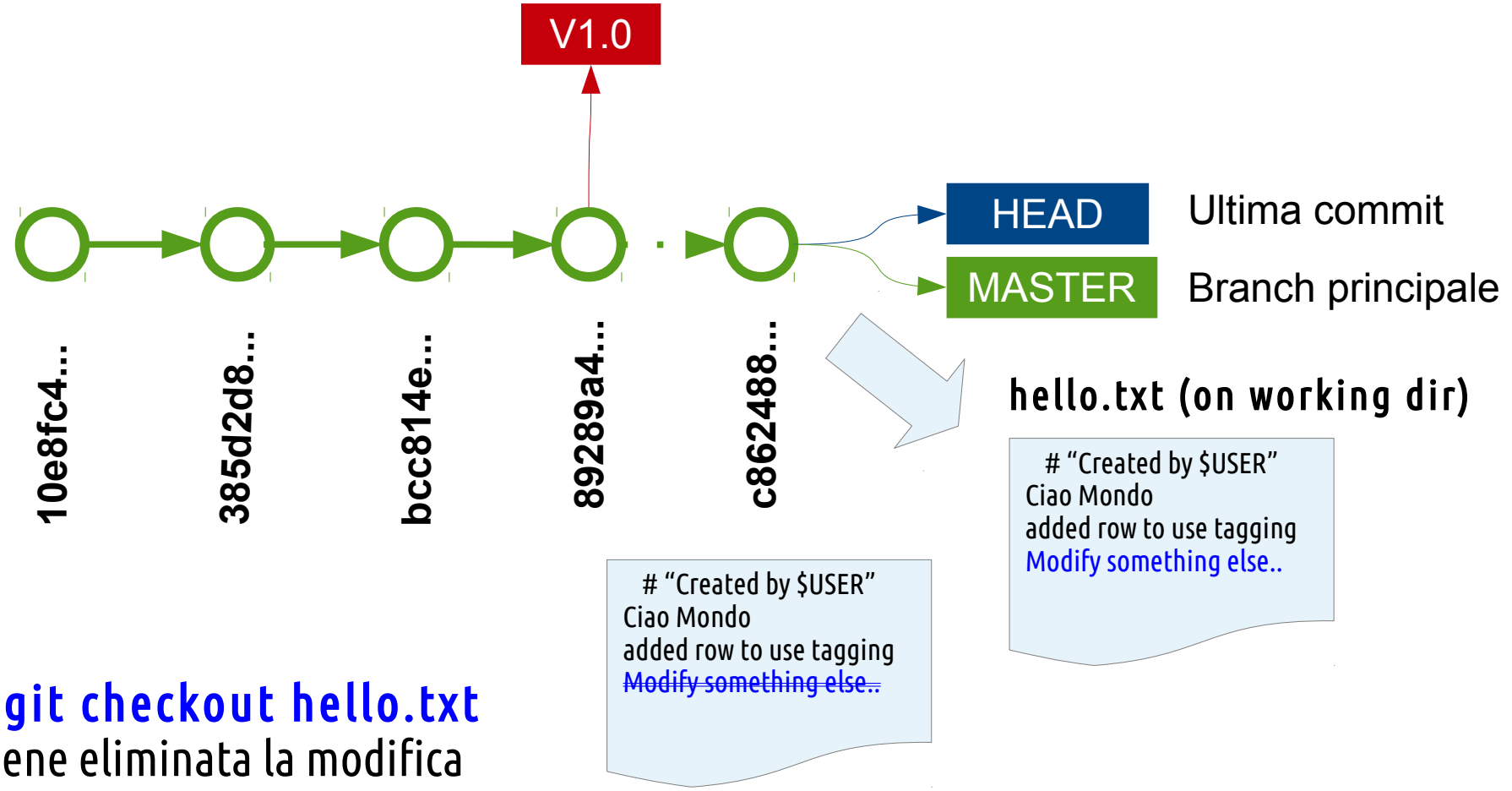


Corso Git

Annullare le modifiche (on stage)

$ git reset HEAD hello.txt

Il file viene tolto dalla staging area e lasciato nella working directory con la modifica



$ git checkout hello.txt

Viene eliminata la modifica

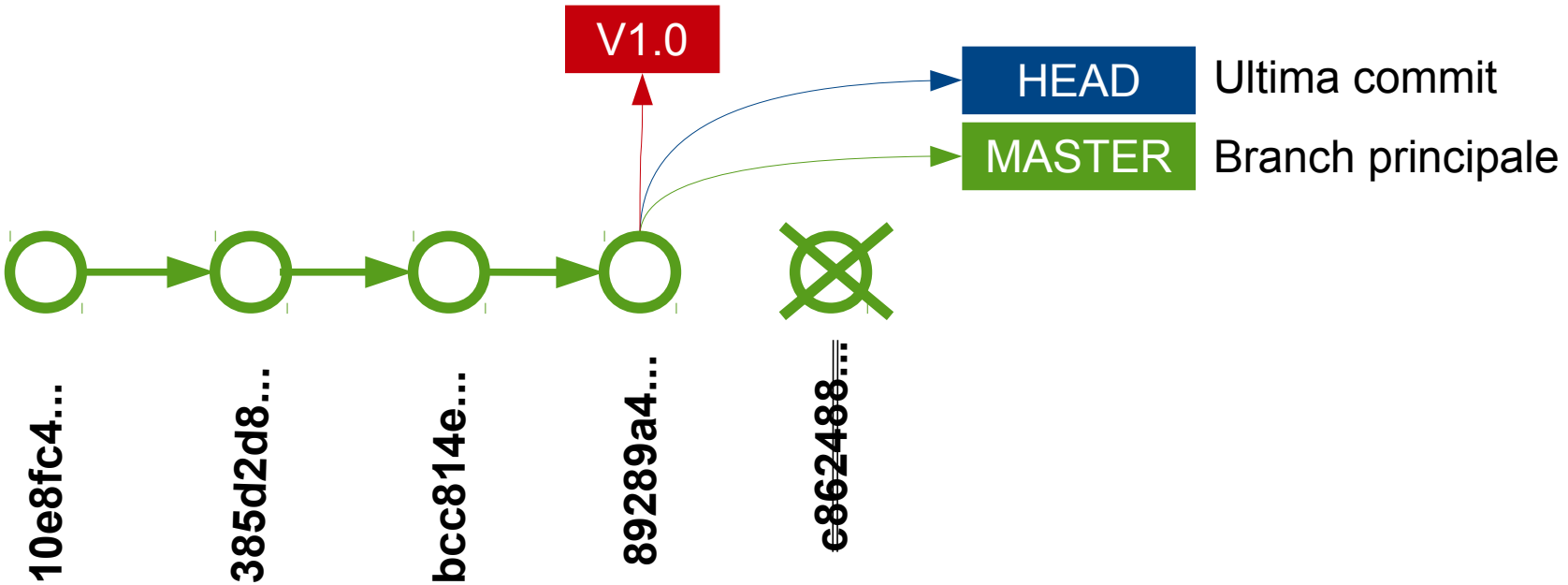
Corso Git

Annullare una commit

$ git revert HEAD

```
$ git revert HEAD
[master da1f328] Revert "Added new row to test GIT tagging"
 1 file changed, 1 deletion(-)
$ cat hello.txt
# “Created by $USER”
Ciao Mondo
$
```

Viene creata una **commit di 'rollback' che annulla gli effetti della commit annullata, che comunque resta**



Corso Git

bticino

Eliminare una commit da un branch

Mi accorgo di aver fatto una commit che, per vari motivi, non dovrebbe comparire in assoluto, nemmeno come Revert (per esempio contiene dati sensibili...). In questi casi il comando da utilizzare è

$ git **reset <tag, hash>**

Che succede?

a) Riscrive il branch fino alla commit specificata

b) Reset della staging area, per renderla consistente con la commit specificata (opzione)

c) Reset della working directory, per lo stesso motivo (opzione)

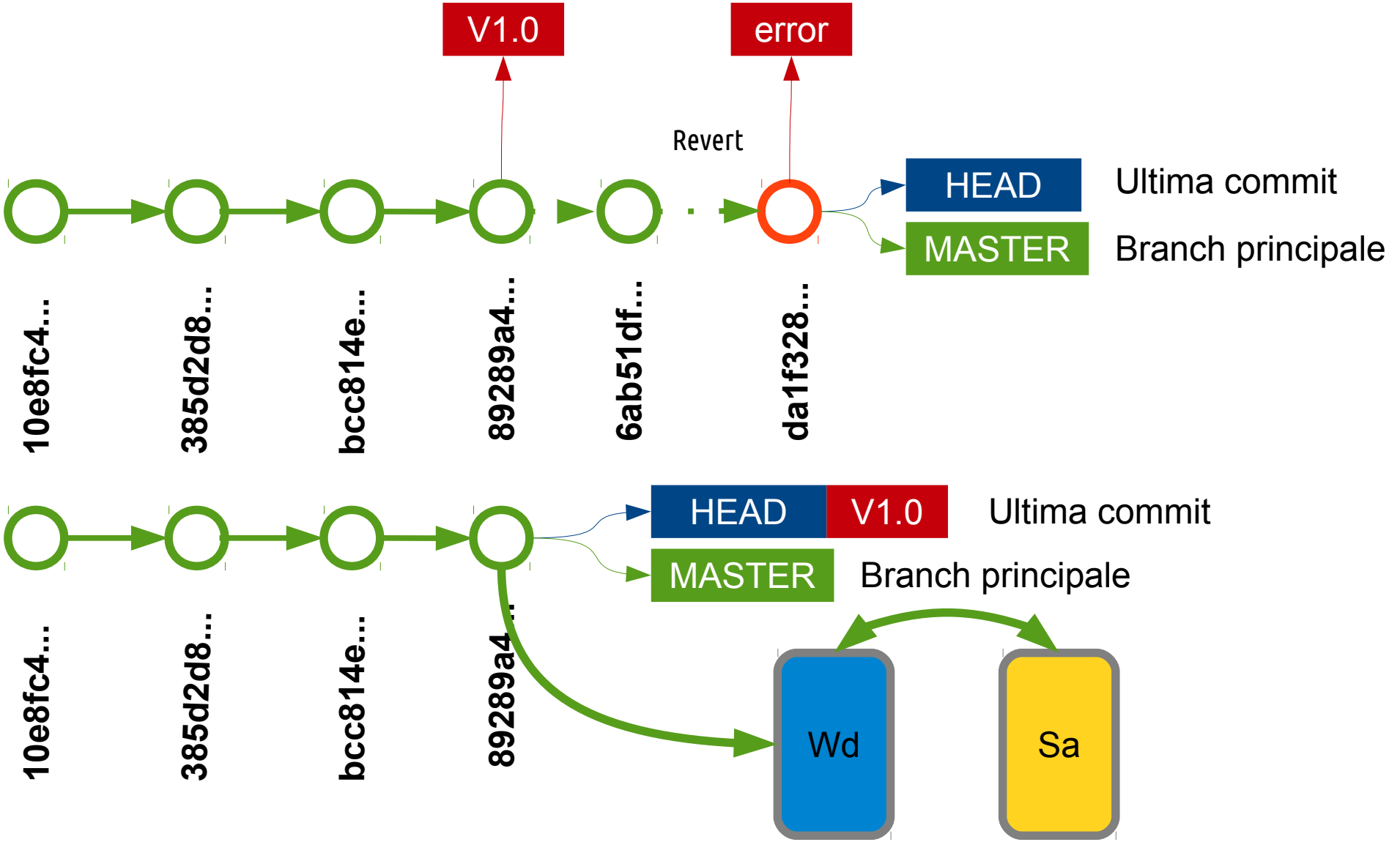
I punti b,c diventano attivi se utilizziamo

$ git **reset - -hard <tag, hash>**

Corso Git

Eliminare una commit da un branch

$ git reset - - hard V1.0



Corso Git

bticino

Eliminare una commit da un branch

```
$ git reset - - hard V1.0
HEAD is now at 89289a4 User variable changed to uppercase (bug fixing)
$ git hist
* 89289a4 2013-08-30 | User variable changed to uppercase (bug fixing) (HEAD, tag: V1.0, master) ...
* bcc814e 2013-08-30 | Modified default comment [Luigi Talamona]
* 385d2d8 2013-08-30 | Added a default comment [Luigi Talamona]
* 10e8fc4 2013-08-30 | First Commit [Luigi Talamona]
```

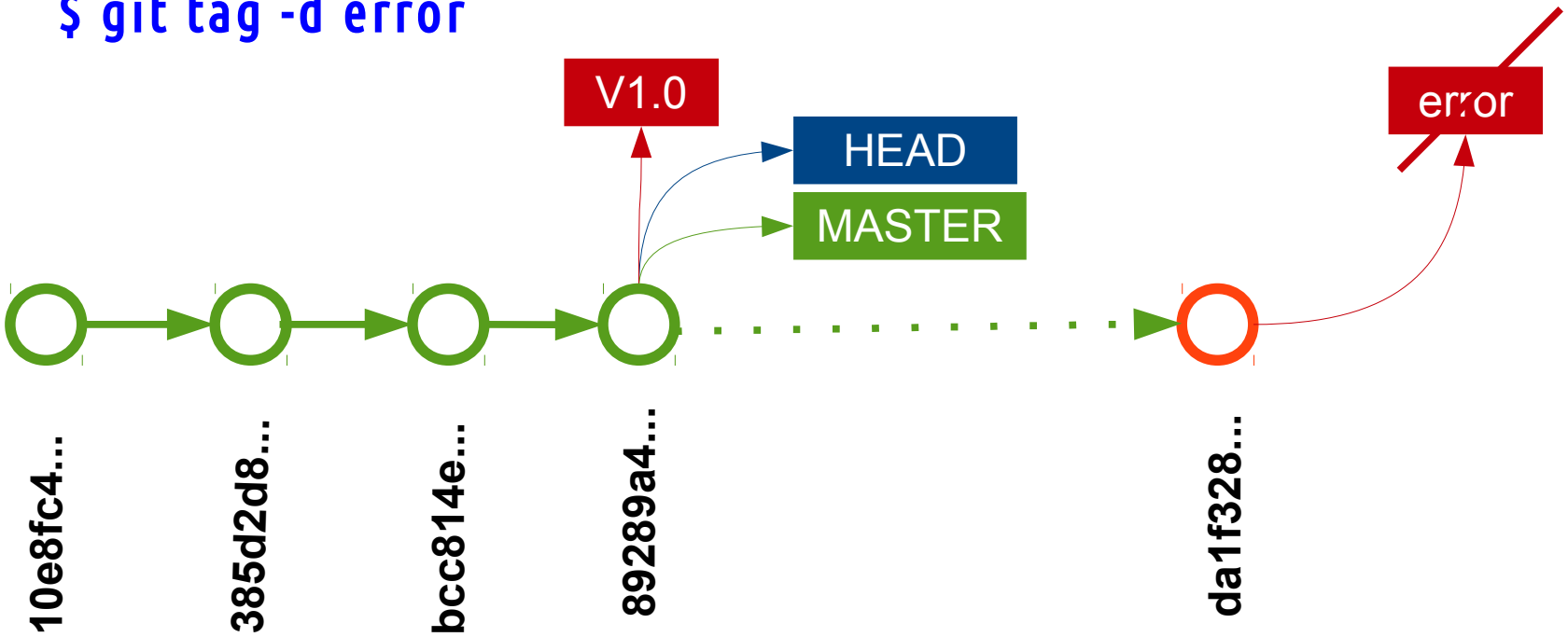
IMPORTANTE: la commit eliminata ma etichettata non viene persa!

```
$ git hist - - all
* da1f328 2013-08-30 | Revert "Added new row to test GIT tagging" (tag: error) [Luigi Talamona]
* 89289a4 2013-08-30 | User variable changed to uppercase (bug fixing) (HEAD, tag: V1.0, master) ...
* bcc814e 2013-08-30 | Modified default comment [Luigi Talamona]
* 385d2d8 2013-08-30 | Added a default comment [Luigi Talamona]
* 10e8fc4 2013-08-30 | First Commit [Luigi Talamona]
```

Corso Git

Se rimuovo la tag, perdo la commit per sempre

$ git tag -d error




```
$ git tag -d error
Deleted tag 'error' (was da1f328)
git hist - - all
* 89289a4 2013-08-30 | User variable changed to uppercase (bug fixing) (HEAD, tag: V1.0, master) ...
* bcc814e 2013-08-30 | Modified default comment [Luigi Talamona]
* 385d2d8 2013-08-30 | Added a default comment [Luigi Talamona]
* 10e8fc4 2013-08-30 | First Commit [Luigi Talamona]
```

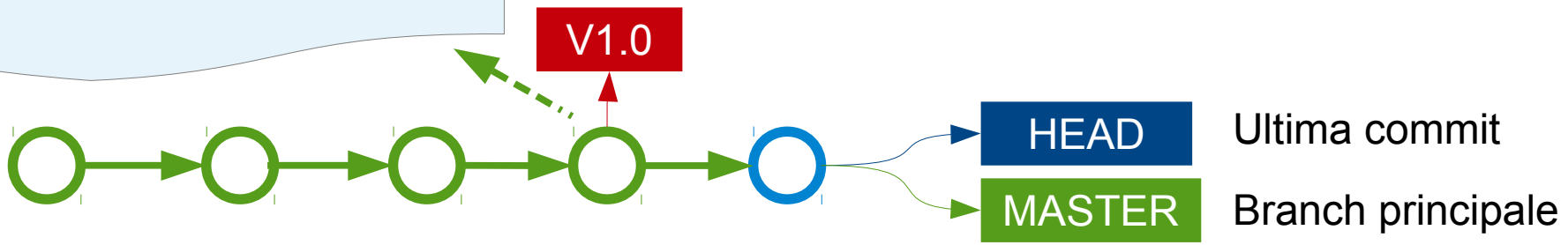
Corso Git

Correggere l'ultima commit

Modifico un file, lo metto on stage (**add**), e poi nella commit area (**commit**)

```
# "Created by $USER" - role developer
Ciao Mondo
added row to use tagging
```

hello.txt




```
$ git hist - - all
* c46d428 2013-09-01 | Added role to comment line (HEAD, master) [Luigi Talamona]
* 89289a4 2013-08-30 | User variable changed to uppercase (bug fixing) (HEAD, tag: V1.0, master) ...
* bcc814e 2013-08-30 | Modified default comment [Luigi Talamona]
* 385d2d8 2013-08-30 | Added a default comment [Luigi Talamona]
* 10e8fc4 2013-08-30 | First Commit [Luigi Talamona]
```

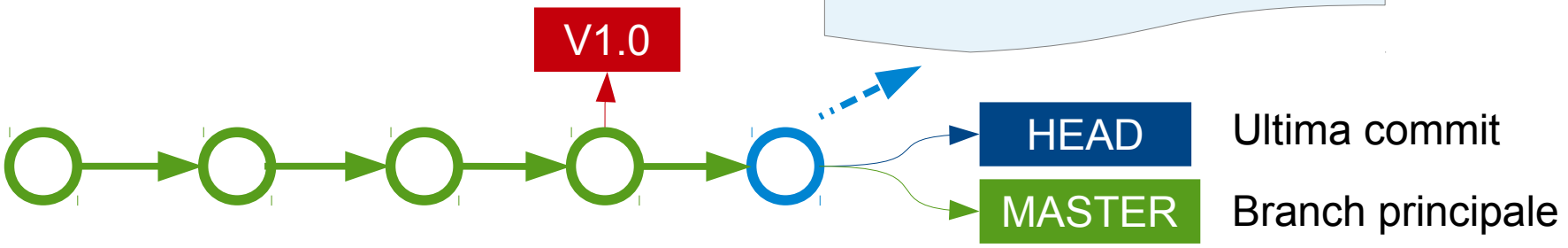
Corso Git

Correggere l'ultima commit

Ho sbagliato: devo correggere il testo che ho aggiunto

hello.txt

```
# “Created by $USER” - user role developer
Ciao Mondo
added row to use tagging
```



Non voglio aggiungere una nuova commit (anche se potrei), ma piuttosto correggere l'ultima...

Come fare?

```
$ git commit - -amend
```

Corso Git

bticino

Correggere l'ultima commit

```
$ git commit - - amend
Added user role to comment line

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD^1 <file>..." to unstage)
#
#       modified:   hello.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   hello.txt
$ git hist
* f11f541 2013-09-01 | Added user role to comment line (HEAD, master) [Luigi Talamona]
* 89289a4 2013-08-30 | User variable changed to uppercase (bug fixing) (tag: V1.0) [Luigi Talamona]
* bcc814e 2013-08-30 | Modified default comment [Luigi Talamona]
* 385d2d8 2013-08-30 | Added a default comment [Luigi Talamona]
* 10e8fc4 2013-08-30 | First Commit [Luigi Talamona]
```

Corso Git

bticino

Muovere files

Voglio muovere il file hello.txt in una nuova directory

```
$ mkdir newdir
$ git mv hello.txt newdir
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#     renamed:    hello.txt -> newdir/hello.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#     modified:   newdir/hello.txt
$ git add newdir
$ git commit -m “added a new dir”
```

Corso Git

bticino

Cancellare files

Voglio cancellare un file dal progetto ed eliminarlo dal controllo di versione

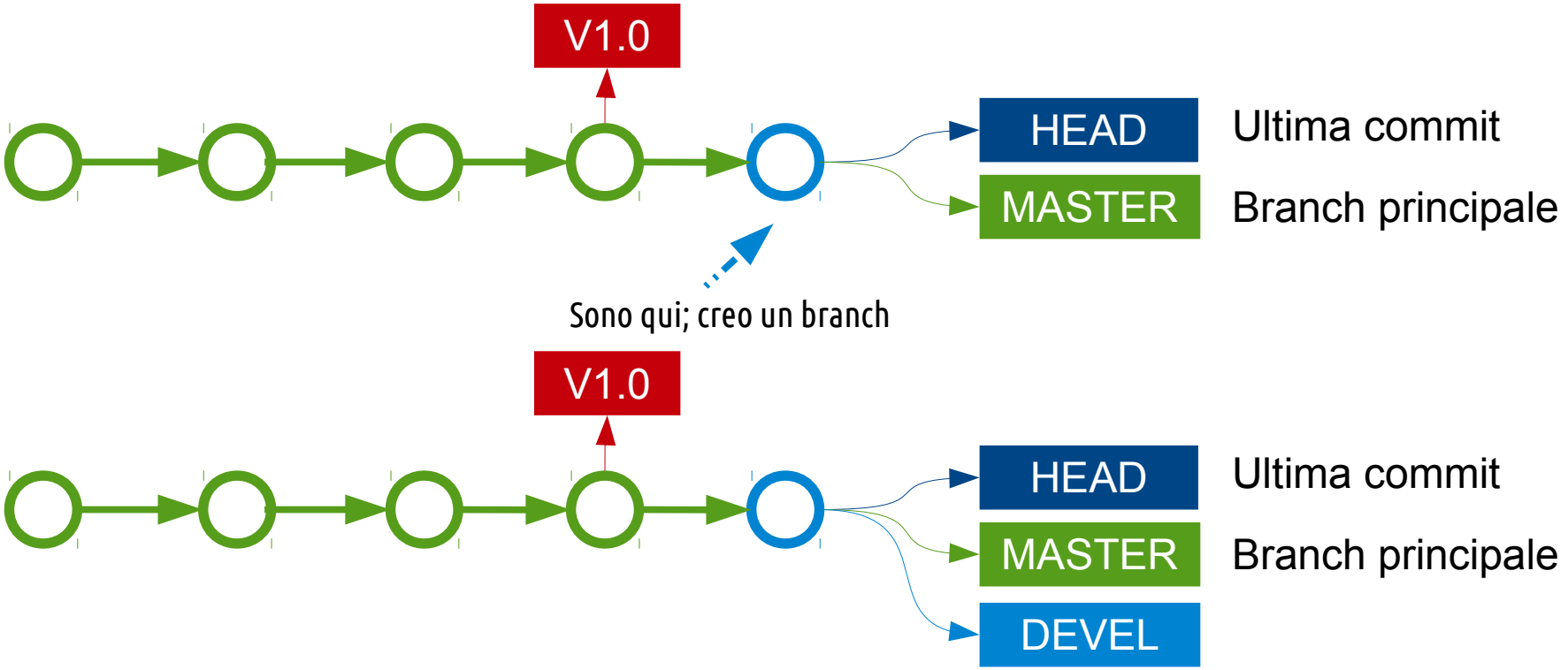
```
$ git rm file1.txt
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    deleted:    file1.txt
#
$ $ git commit -m “cancellato file1.txt”
$
```

Corso Git

Utilizzare i branches

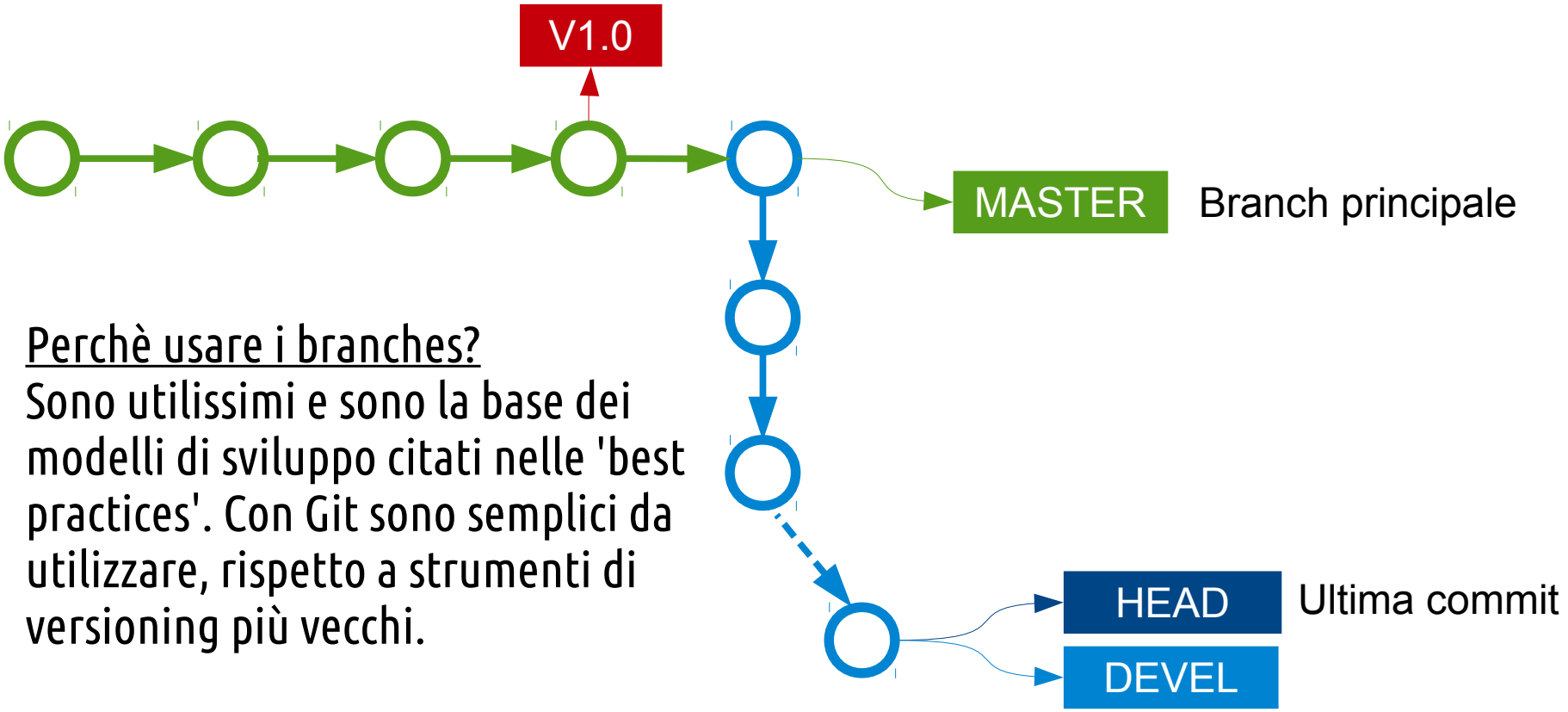
Cos'è un branch?

Un branch in Git è semplicemente un puntatore mobile ad una commit. Quando creo un nuovo branch, creo solo un puntatore ad una commit esistente. Se poi mi posiziono su quel puntatore, e lavorando creo nuove commit, creo un nuovo ramo di sviluppo.



Corso Git

Utilizzare i branches



Perchè usare i branches?
Sono utilissimi e sono la base dei modelli di sviluppo citati nelle 'best practices'. Con Git sono semplici da utilizzare, rispetto a strumenti di versioning più vecchi.

Corso Git

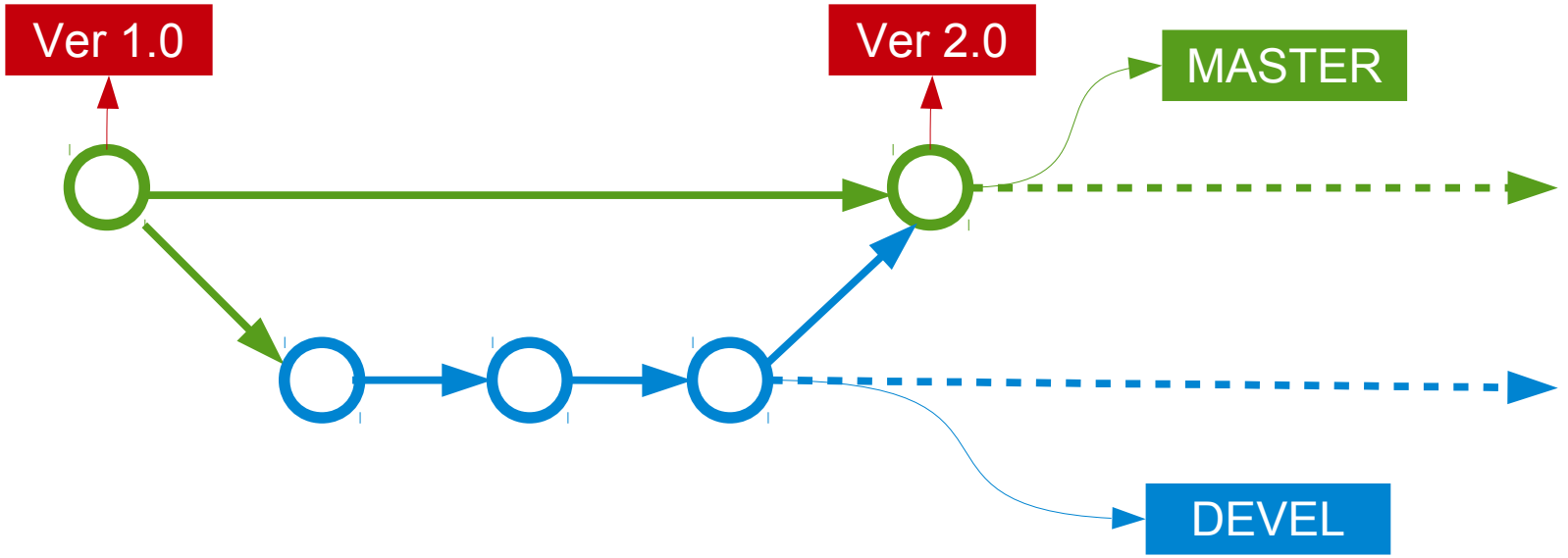
Utilizzare i branches

Flusso normale di sviluppo

Un progetto software in fase di sviluppo mantiene sempre almeno due 'cantieri' aperti:

PRODUZIONE il ramo che genera le release da distribuire (master)

DEVEL Il ramo che contiene i nuovi sviluppi

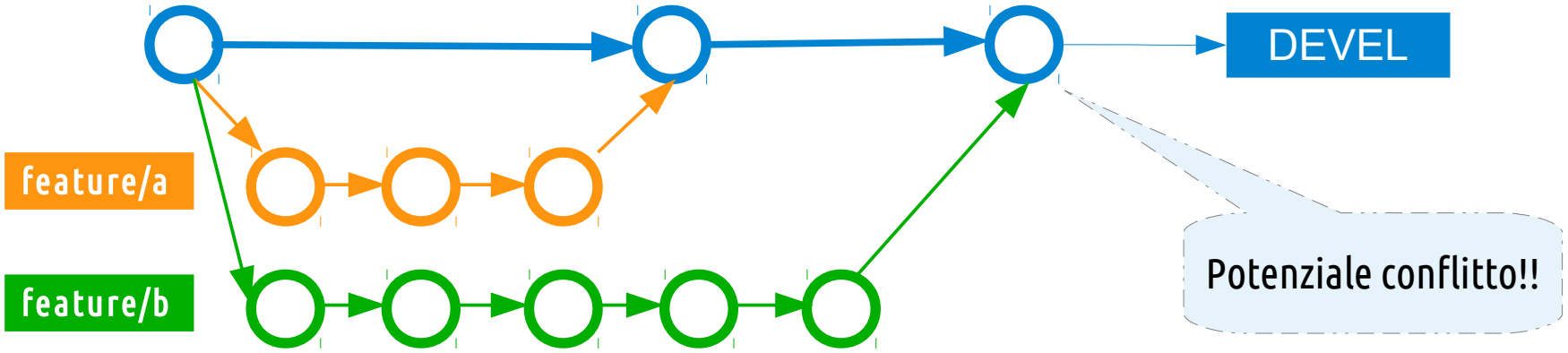


Corso Git

Utilizzare i branches

Feature branch

- Branch di sviluppo di una nuova funzionalità
- Ramifica a partire da **devel** e si innesta su **devel**
- a fine sviluppo viene cancellato

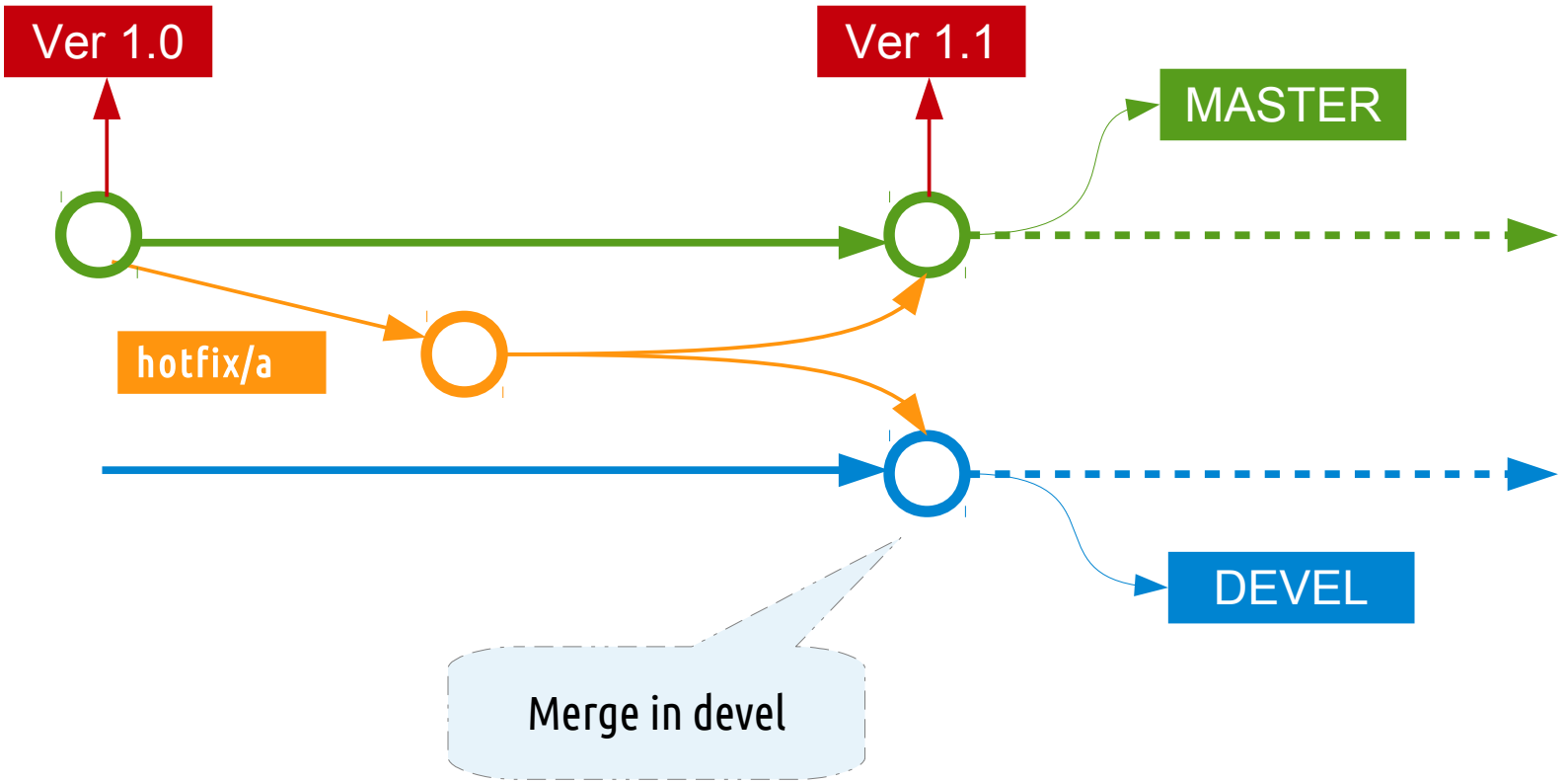


Corso Git

Utilizzare i branches

Hotfix branch

- branch di fix per un bug in produzione
- ramifica da master (produzione) e si innesta su master
- a fine sviluppo viene cancellato (può essere necessario un merge su devel)

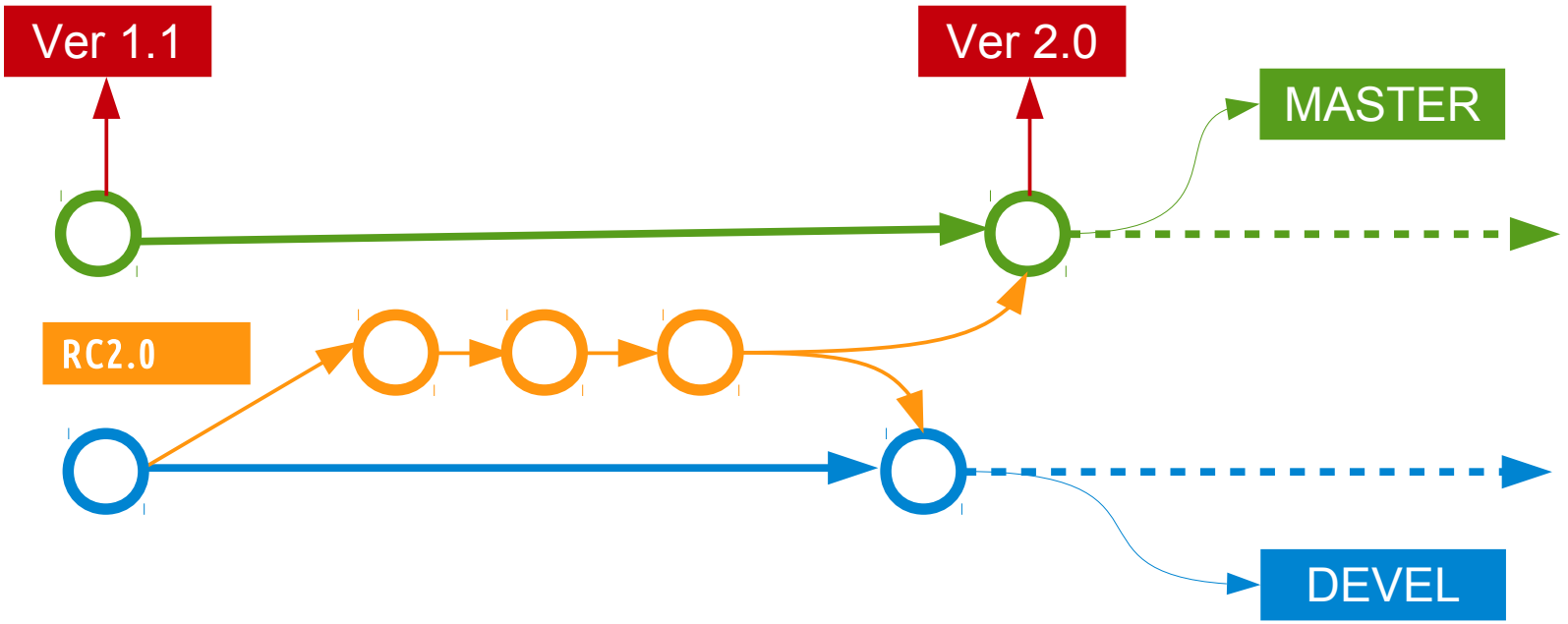


Corso Git

Utilizzare i branches

Release branch

- candidata in test per la prossima versione di produzione
- ramifica da devel e si innesta su master
- viene generalmente “taggata” con un numero di versione



Corso Git

Creare un branch

Supponiamo di partire dal branch MASTER, ultima commit (HEAD)

```
$ git branch DEVEL
$ git checkout DEVEL
```

Più brevemente:

```
$ git checkout -b DEVEL
```

```
$ git checkout -b DEVEL
Switched to a new branch 'DEVEL'
$ git branch
* DEVEL
  master
$ git hist
* b94d813 2013-09-01 | added a new dir (HEAD, master, DEVEL) [Luigi Talamona]
* f11f541 2013-09-01 | Added user role to comment line [Luigi Talamona]
* 89289a4 2013-08-30 | User variable changed to uppercase (bug fixing) (tag: V1.0)..
* bcc814e 2013-08-30 | Modified default comment [Luigi Talamona]
* 385d2d8 2013-08-30 | Added a default comment [Luigi Talamona]
* 10e8fc4 2013-08-30 | First Commit [Luigi Talamona]
```

Corso Git

Navighiamo tra i branches

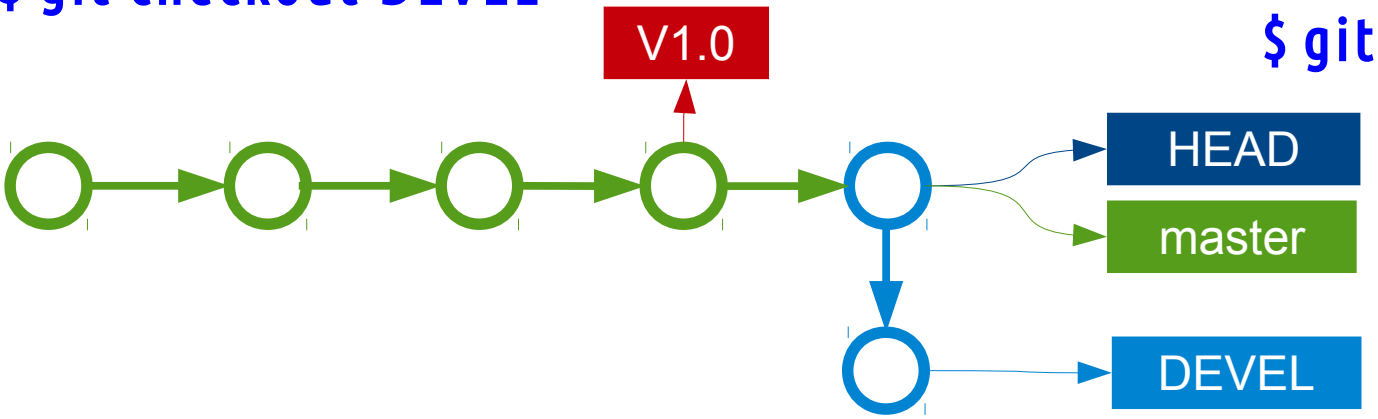
Siamo nel branch DEVEL e vogliamo passare al branch master:

$ git checkout master

Di nuovo da master a DEVEL

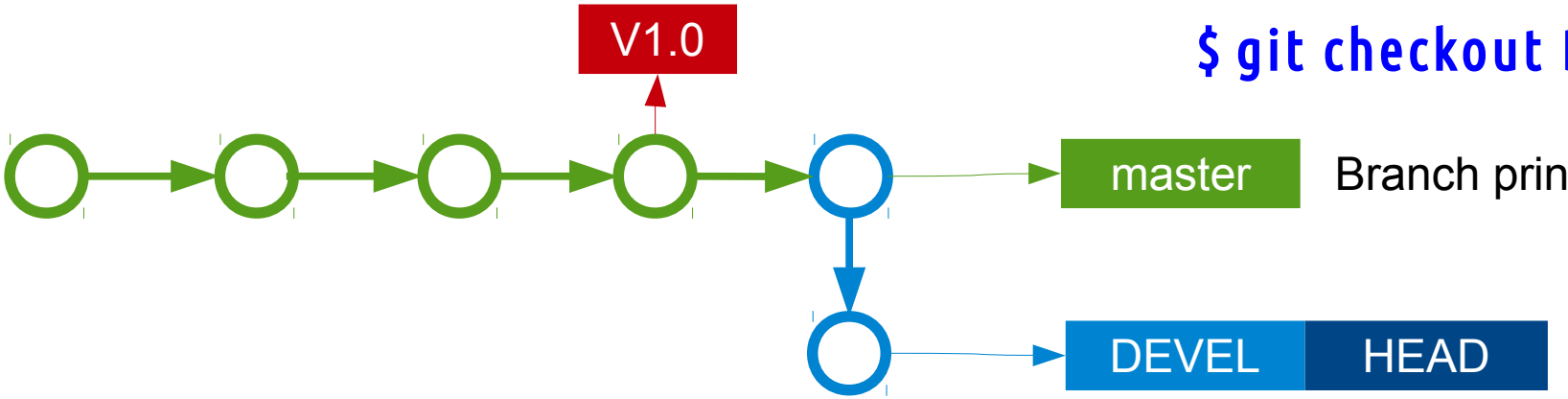
$ git checkout DEVEL

$ git checkout master



Branch principale

$ git checkout DEVEL



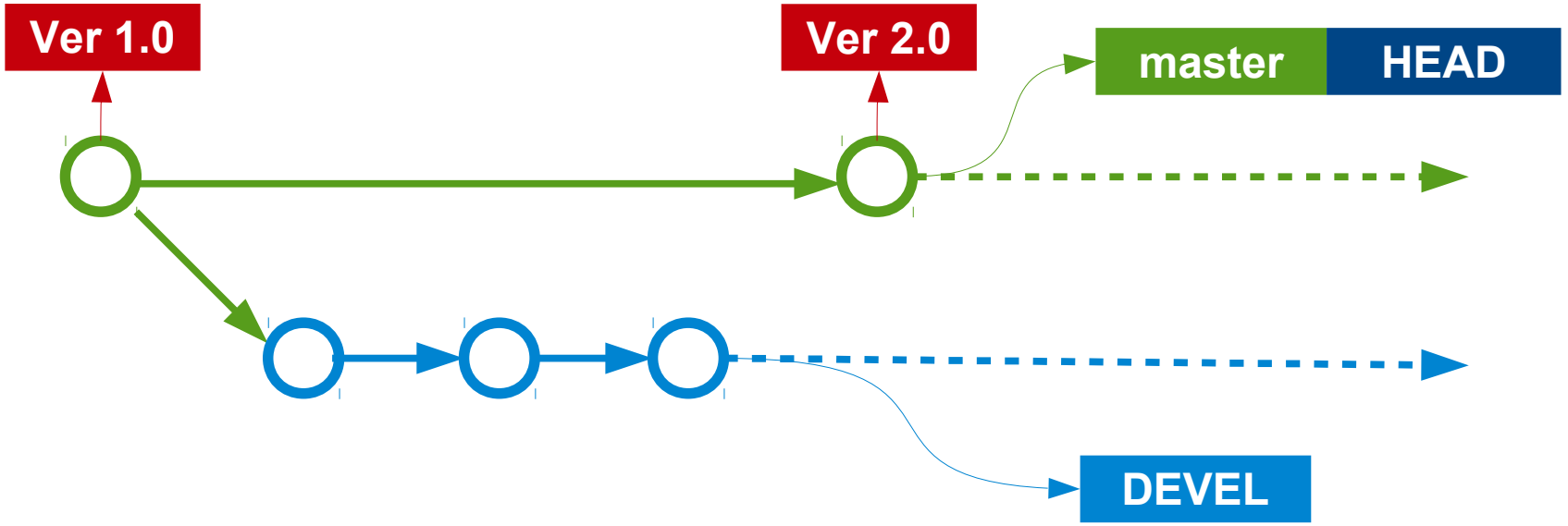
Branch principale

Corso Git

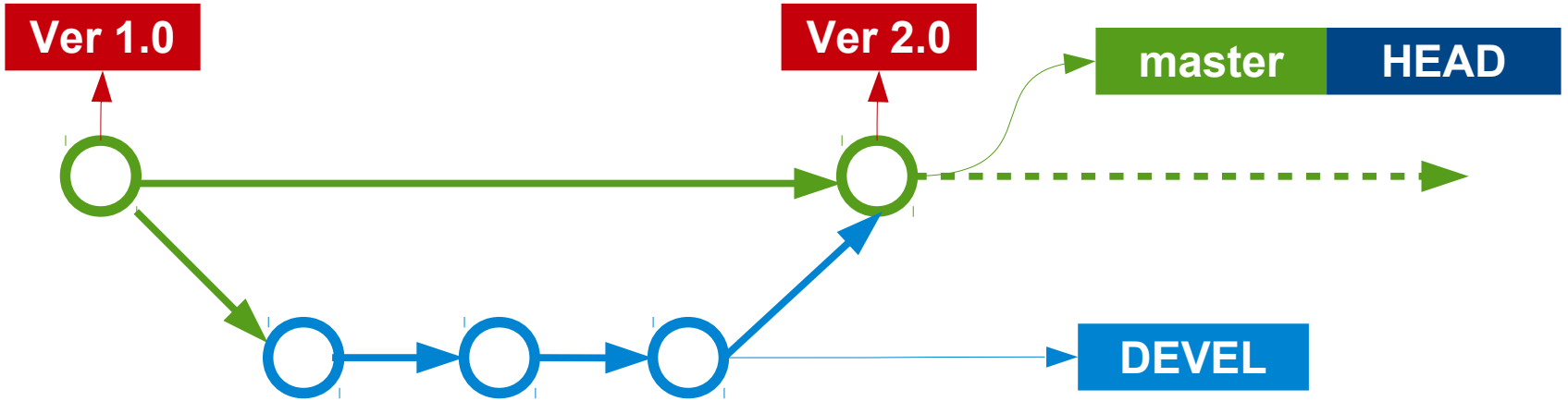
Merging

Siamo al punto di far passare il lavoro svolto da un ramo ad un altro

$ git checkout master



$ git merge DEVEL

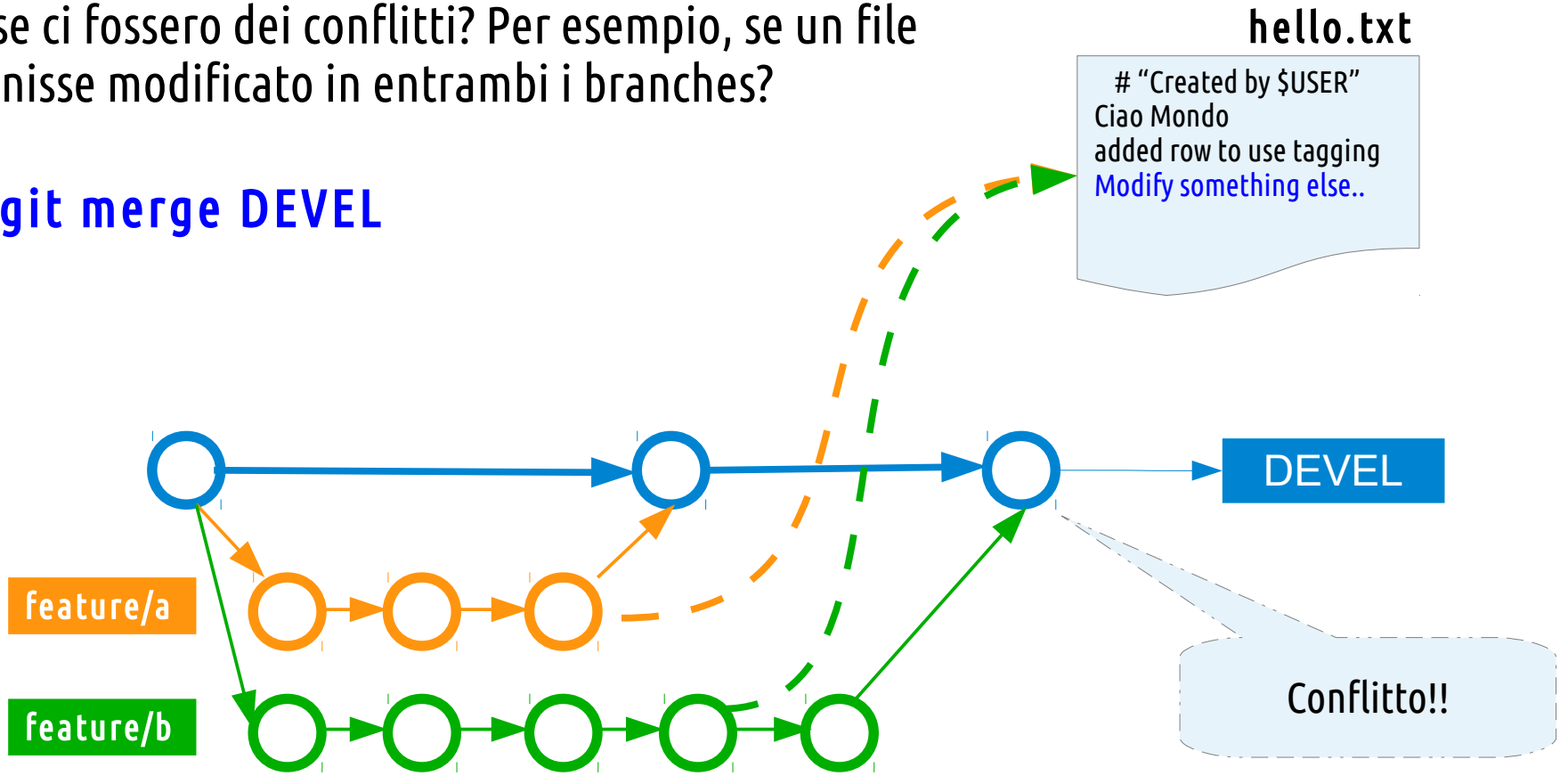


Corso Git

Merging

E se ci fossero dei conflitti? Per esempio, se un file venisse modificato in entrambi i branches?

$ git merge DEVEL



In questo caso dobbiamo risolvere manualmente il conflitto e poi fare una commit.
Questo succede con qualsiasi sistema di versioning.