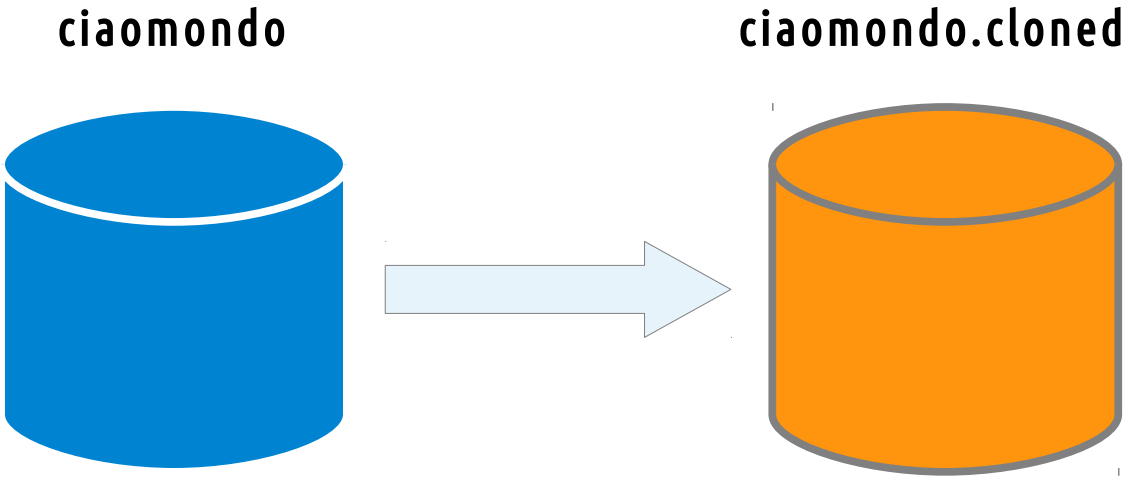
Corso Git

Lavorare con più repository: **clone**




```
$ pwd
$HOME/ciaomondo
$ cd ..
$ git clone ciaomondo ciaomondo.cloned
$ ls
ciaomondo
ciaomondo.cloned
```

Corso Git

bticino

Lavorare con più repository: **origin**

```
$ git remote
```

```
$ cd ciaomondo.cloned
$ git remote
origin
$ git remote show origin
* remote origin
  Fetch URL: $HOME/ciaomondo
  Push  URL: $HOME/ciaomondo
  HEAD branch: master
  Remote branches:
    DEVEL  tracked
    master tracked
    testA  tracked
    testB  tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

Corso Git

bticino

Lavorare con più repository: **remote branches**

```
$ git branch
$ git branch -a
```

```
$ git branch
* master
$ git branch -a
* master
  remotes/origin/DEVEL
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

Nel repository clonato, il branch di default è quello corrente nel repository sorgente

Corso Git

bticino

Lavorare con più repository: branch remoto

Quando si crea il clone di un repository, viene copiato solo il branch corrente. Nel repository i branches remoti vengono mostrati ma non possono essere utilizzati, a meno che non lo si decida esplicitamente

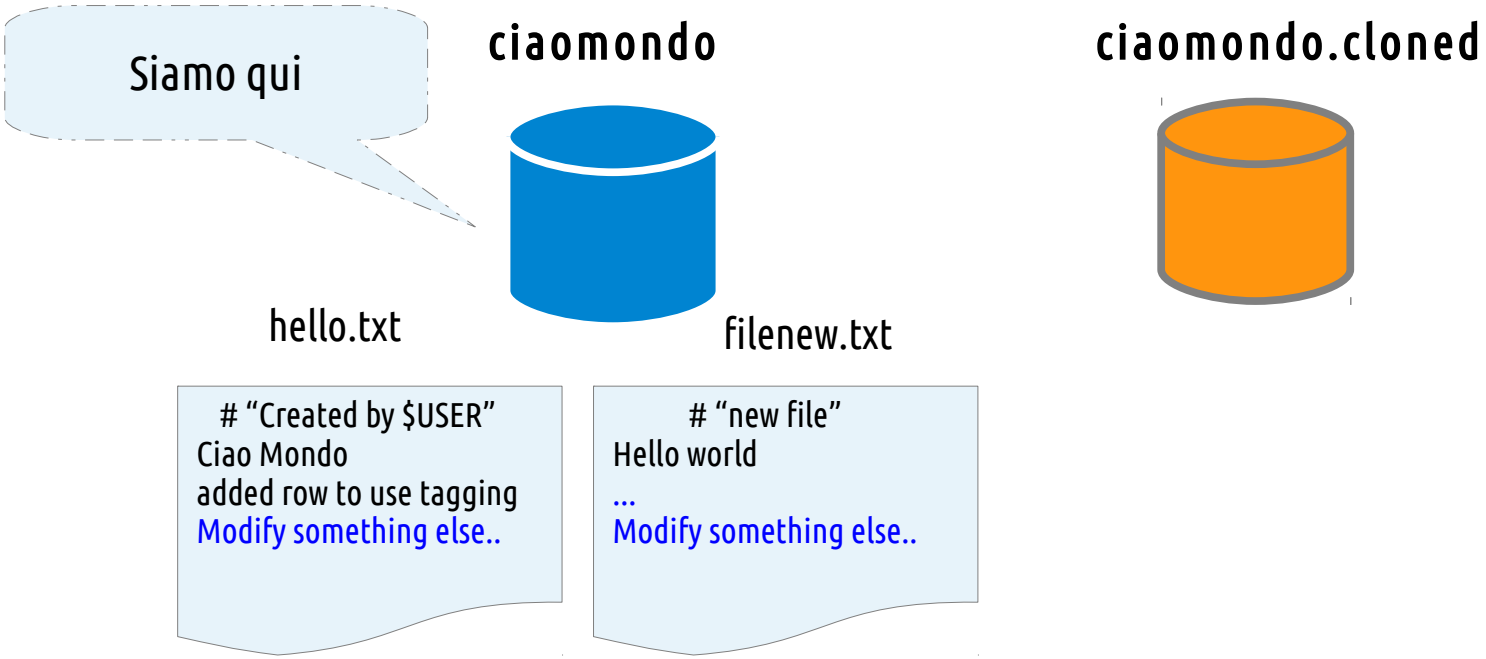
```
$ git branch
* master
$ git branch -a
* master
  remotes/origin/DEVEL
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

git branch --track devel remotes/origin/DEVEL

```
$ git branch –track devel remotes/origin/DEVEL
Branch devel set up to track remote branch DEVEL from origin.
$ git branch -a
* master
devel
  remotes/origin/DEVEL
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

Corso Git

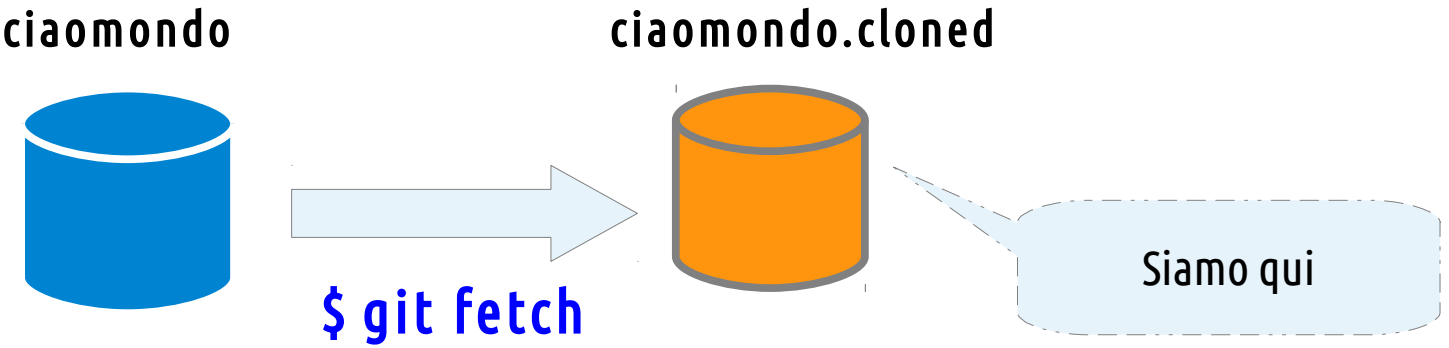
Lavorare con più repository: **fetch e pull**



Nel repository origine modifichiamo il file hello.txt, che esisteva già prima della copia, e ne aggiungiamo uno nuovo (filenew.txt)

Corso Git

Lavorare con più repository: **fetch e pull**



Se verifichiamo il corrispondente del file modificato nel repository originale, noteremo che **non è cambiato. Anche il nuovo file non compare. Fetch** infatti fa il download delle commit da origin, le memorizza temporaneamente (cartella .git) ma senza fare un merge successivo.

Corso Git

bticino

Lavorare con più repository: **fetch e pull**

```
$ git pull
Updating c6f442e..1c79f82
Fast-forward
 file1.txt  | 1 +
 file_c.txt | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 file_c.txt
```

Dopo aver invocato il comando pull, il file appare modificato.

Quindi pull = **fetch + merge**

Corso Git

bticino

Lavorare con più repository: **bare**

I repository **bare** (nudo) sono repository che servono solo per la condivisione, e **non hanno la working directory**. I repository creati con questa opzione si denotano convenzionalmente con la desinenza **.git**

```
git clone --bare ciaomondo ciaomondo.git
```

```
$ cd $HOME/temp
$ git clone --bare ciaomondo ciaomondo.git
Cloning into bare repository 'ciaomondo.git'...
done.
$
```

Da notare il fatto che un repository clonato con questa opzione **ha tutti i branches** del repository padre

```
$ cd ciaomondo.git
$ git branch
  DEVEL
* master
$
```

Corso Git

Repository remoti

Tutto quello che abbiamo visto finora ci fa pensare che un repository remoto (in un altro network) non sia altro che un repository **bare**, al quale abbiamo accesso e che possiamo clonare per iniziare le nostre attività.

Ci sono diversi repository managers per Git, in Cloud o installabili in un local network. Un esempio del primo tipo è:

https://github.com/Bticino-Varese

del secondo, invece:

http://10.31.21.14:8080/ (GitBlit server)

Corso Git

Repository remoti

In ogni caso clonare un repository **bare** per ottenere un repository di lavoro si fa sempre allo stesso modo

```
$ cd $HOME
$ git clone https://github.com/Bticino-Varese/sandbox1.git
Cloning into 'sandbox1'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 0), reused 6 (delta 0)
Unpacking objects: 100% (9/9), done.
$
```

Corso Git

Repository remoti

Si lavora sul progetto in **locale**, come visto nei passi precedenti

```
$ cd $HOME/sandbox1
$ touch test.txt
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    test.txt
nothing added to commit but untracked files present (use "git add" to track)
$ git add test.txt
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    new file:   test.txt
#
$ git commit -m “Initial release”
$
```

Corso Git

Repository remoti: push

Git ci comunica che il delta commit vale 1; non siamo più allineati con il repository **bare** remoto, dal quale siamo partiti, e ci suggerisce cosa fare per riottenere il riallineamento.

```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
$
```

Quindi se seguiamo il consiglio dell'helper in linea otteniamo:

```
$ git push
Counting objects: 6, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 327 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/Bticino-Varese/sandbox1.git
  226c560..199dfc6  master -> master
```

Corso Git

Repository remoti: conflitti

Immaginiamo ora che un secondo sviluppatore abbia creato un altro clone dallo stesso repository remoto, ed abbia fatto delle commit in locale mentre noi facevamo la push vista prima. Cosa succede se vuole a propria volta fare una push?

```
$ git push
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/Bticino-Varese/sandbox1.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first merge the remote changes (e.g.,
hint: 'git pull') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

L'helper ci aiuta, suggerendoci cosa avremmo dovuto fare prima di iniziare a lavorare (fetch), e cosa dobbiamo fare ora (pull).

Corso Git

Repository remoti

Fetch: download delle differenze tra locale e remoto, ma senza operare un **merge.**

```
$ git fetch
 remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (3/3), done.
$
```

Pull: download delle differenze tra locale e remoto, con successiva merge automatica (se non ci sono conflitti!!)

```
$ git pull
 Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
$
```

Corso Git

Repository remoti

Il file che genera il conflitto viene “decorato” automaticamente con le indicazioni necessarie a risolvere la disputa

```
$ vi test.txt
 riga 1
<<<<<<< HEAD
riga 2 da user2
=======
riga2 by user1
>>>>>>> 4147529a1246ce87e7578b9fd5ed922de5bf3c91
...
```

Risolto il conflitto, facciamo la commit riparatrice ed infine salviamo il tutto nel repository remoto

```
$ git add test.txt
$ git commit -m “Conflict resolved”
$ git push
(...)
To https://github.com/Bticino-Varese/sandbox1.git
  4147529..5f38757  master -> master
$
```

Corso Git

Tags remote

Le tags che utilizziamo nel repository locale non vengono automaticamente copiate nel repository remoto

```
$ git tag TAG_REMOTA
$ git tag
...
TAG_REMOTA
$
```

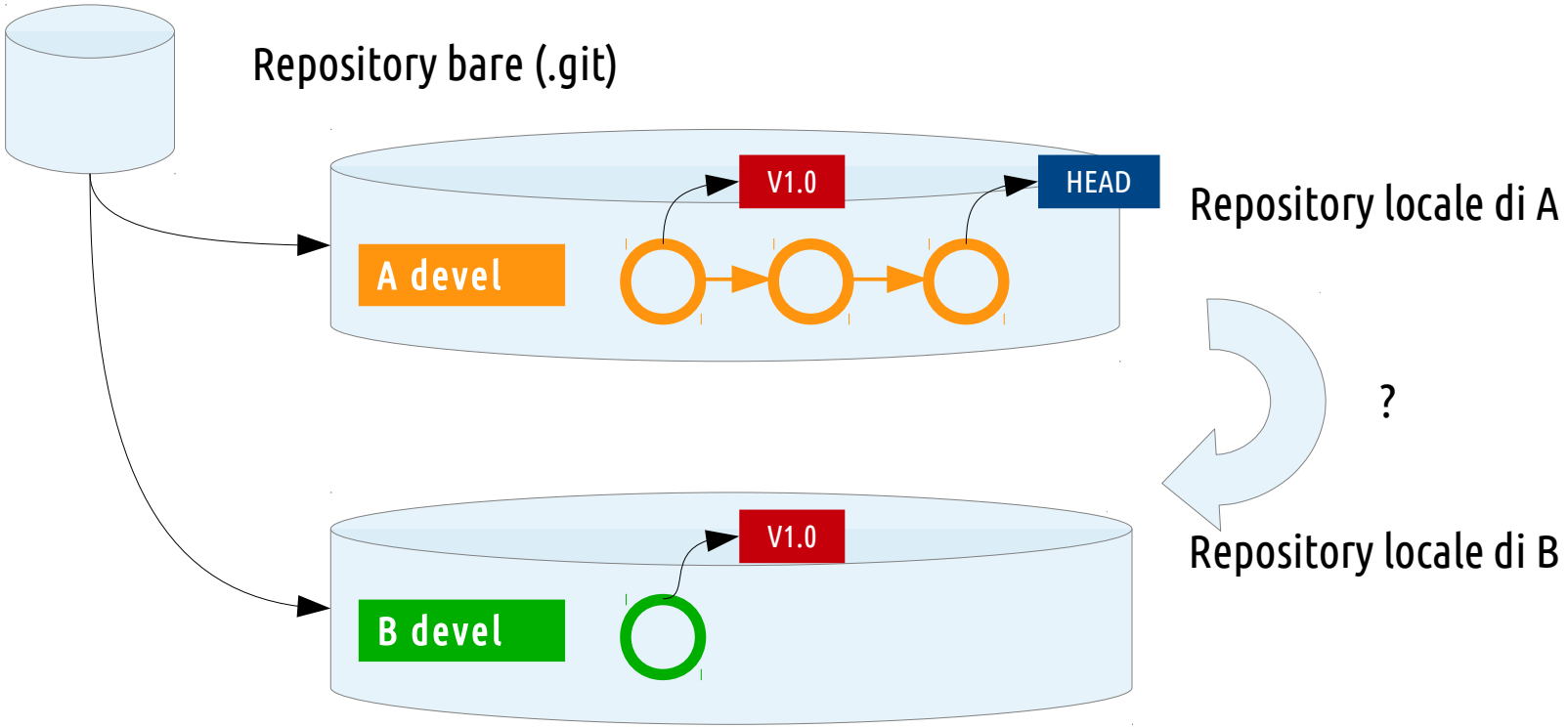
Per memorizzarle anche nel repository remoto, è necessario dichiararlo esplicitamente

```
$ git push --tags
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/Bticino-Varese/sandbox1.git
 * [new tag]         TAG_REMOTA -> TAG_REMOTA
$
```

Corso Git

Patch

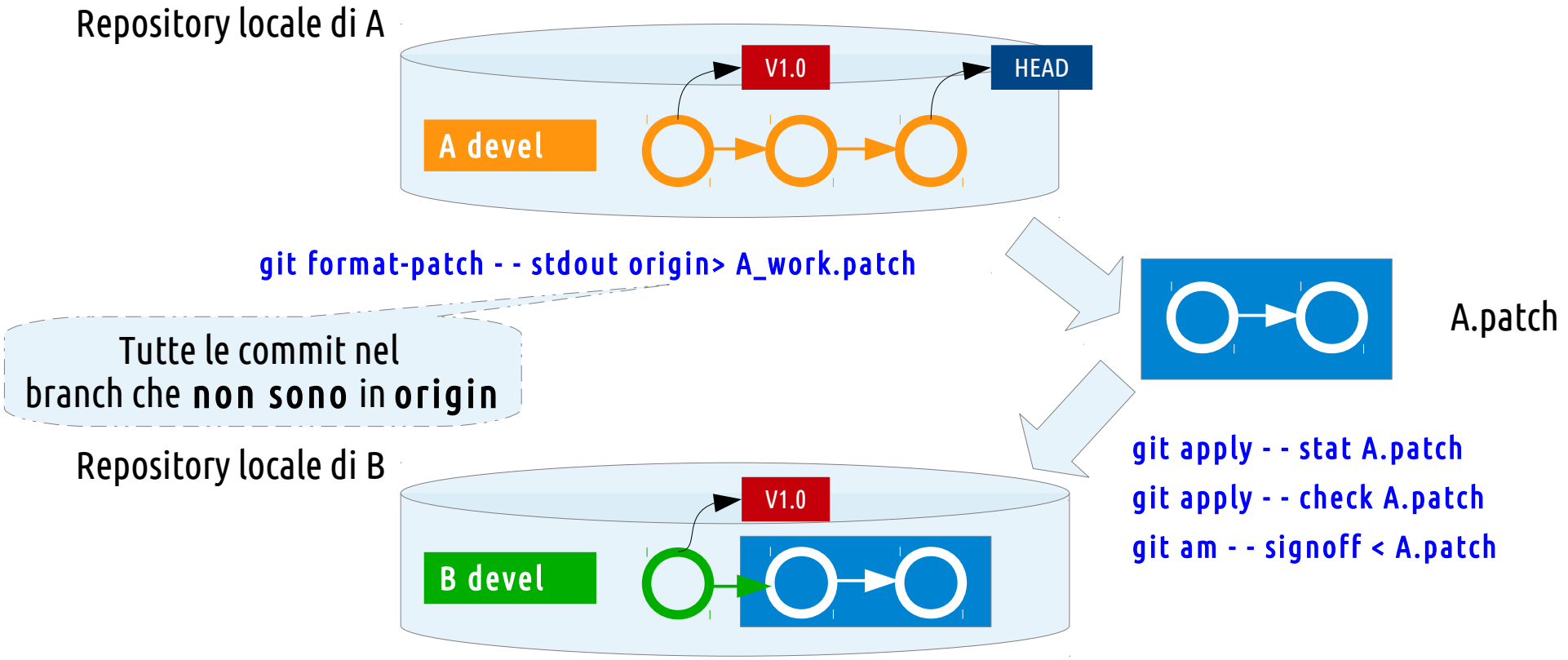
Supponiamo che lo sviluppatore **A** cloni un repository remoto ed inizi a lavorare. Improvvisamente, per qualche motivo, deve lasciare l'attività in corso e passarla allo sviluppatore **B**...



Corso Git

Patch

Le modifiche fatte da A sono ancora instabili, ed è meglio che non vengano salvate nel repository bare. Con Git si può creare una patch, che è un file di testo che elenca le modifiche fatte da A, da un certo punto in poi. B prende questo file e lo applica al proprio repository locale



Corso Git

Patch

```
$ git tag
...
V1.0
... <A> fa delle modifiche, le aggiunge all'area di stage e fa una commit
$ git commit -m “Prima commit”
... <A> fa delle altre modifiche, le aggiunge all'area di stage e fa una commit
$ git commit -m “Seconda commit”
... <A> prepara la patch da passare al collega
$ git format-patch –stdout origin > A_work.txt
```

```
... <B> ottiene la patch (allegato mail, usbstick, ftp, scp...) e ne verifica il contenuto
$ git apply - -stat A_work.txt
...
... <B> verifica che la patch non contenga errori
$ git apply - - check A_work.txt

... <B> applica la patch al proprio repository locale
$ git am - -signoff < A_work.txt
```