DelphiDay italian conference

PIACENZA EDITION

2019 - XVIII Edizione

git

Build software better, together

wintech italia

speaker: Maurizio Del Magno

embarcadero reseller partner

Maurizio Del Magno

Developer

levante software

i-ORM github.com/mauriziodm/iORM

DJSON github.com/mauriziodm/DJSON

mauriziodm@levantesw.it

mauriziodelmagno@gmail.com

facebook.com/maurizio.delmagno

iORM + DJSON (group)

Membro fondatore

eInvoice4D

https://github.com/delphiforce/eInvoice4D

DelphiDay italian conference PIACENZA EDITION

2019 - XVIII Edizione

git

Build software better, together

wintech italia

speaker: Maurizio Del Magno

embarcadero reseller partner

Git

…perchè questo nome, cosa significa?

‘Idiota’ Git Hub?

“Sono maledettamente egoista ed egocentrico, chiamo tutti i miei progetti in modo che rimandino direttamente e me: prima ‘Linux’, ora ‘Git’ ” *(cit. Linus Torvalds)*

“ ‘Git’ può significare qualsiasi cosa, a seconda del tuo stato d’animo, fai la tua scelta dal dizionario dei gerghi inglese” *(cit. Linus Torvalds)*

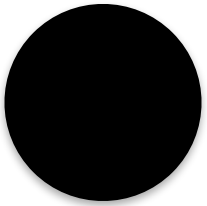
- *Idiota / stupido*
- *Persona polemica, irascibile, che vuole avere sempre ragione*
- *Testa di porco*
- *Semplice*
- *…*

- *“Global Information Tracker”* *(se sei di buon umore e funziona davvero per te, gli angeli cantano e la luce improvvisamente riempie la stanza - cit. Linus Torvalds)*
- *“Maledetto autocarro carico di m…a”* *(quando si rompe - cit. Linus Torvalds)*

Git

…a cosa serve?

- **…a tenere traccia delle modifiche ai sorgenti** (e ai files in generale)
- **…se qualcosa va storto ci permette di tornare senza problemi a versioni precedenti del codice**
- **…a collaborare con altri**

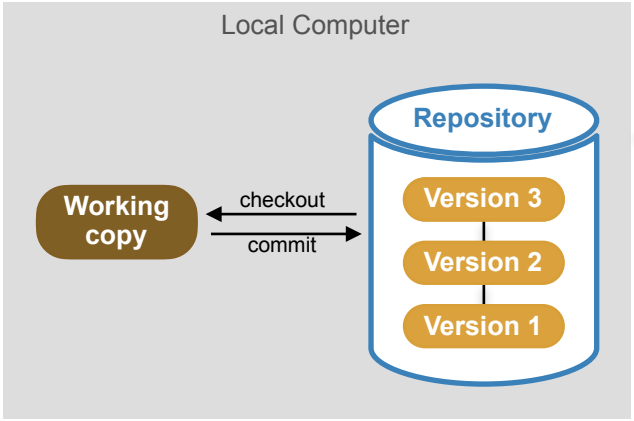


VCS *(Version Control System)*

LVCS

Local **V**ersion **C**ontrol **S**ystem

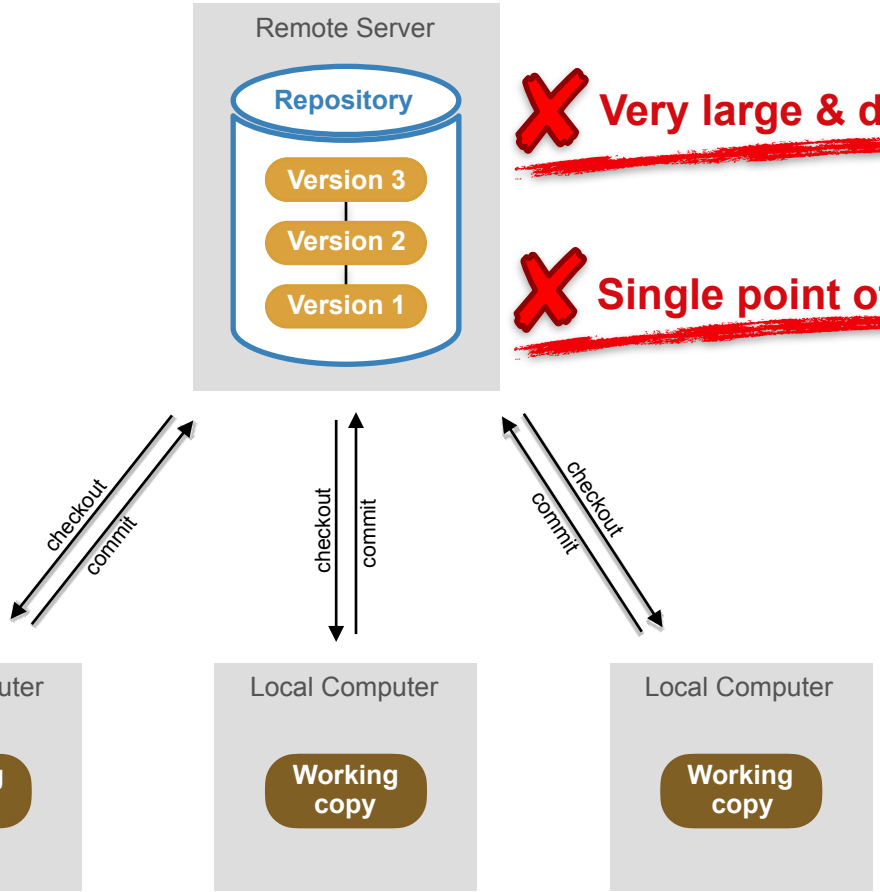
- RCS.



CVCS

Centralized **V**ersion **C**ontrol **S**ystem

- CVS
- Subversion
- Perforce.



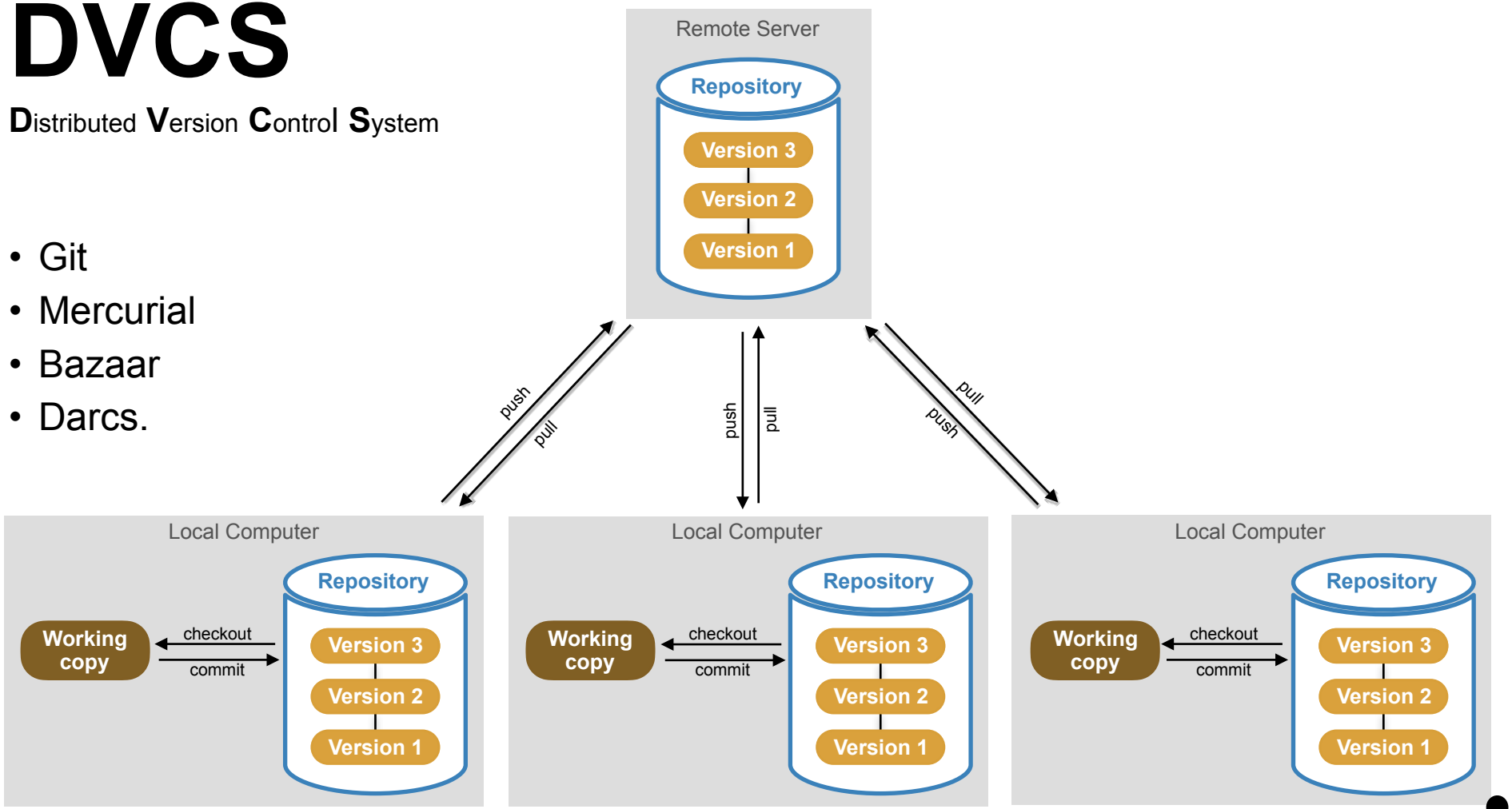
Very large & distributed teams?

Single point of failure

DVCS

Distributed **V**ersion **C**ontrol **S**ystem

- Git
- Mercurial
- Bazaar
- Darcs.



Git

…i perchè di Linus Torvalds

- **Serviva un sostituto di BitKeeper** *(che non era più gratuito)*

- **Nessun altro sistema gratuito soddisfaceva le necessità di Torvalds**
  1. Sistema distribuito *(come BitKeeper)*
  2. Salvaguardia dalla corruzione dei dati *(accidentale o intenzionale)*
  3. Altissime prestazioni
  4. Forte supporto allo sviluppo non lineare *(migliaia di branch paralleli)*
  5. CVS come esempio di cosa “non fare” *(nel dubbio fai il contrario)*

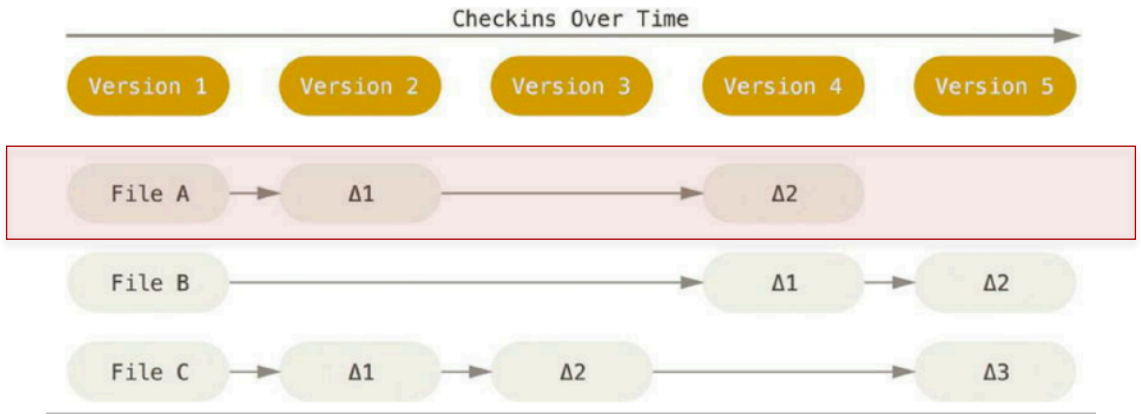
Git Basics:

Dimenticate quello che sapete degli altri VCS!

Git Basics:

Snapshots, Not Differences

Gli altri VCS memorizzano liste di differenze per singolo file



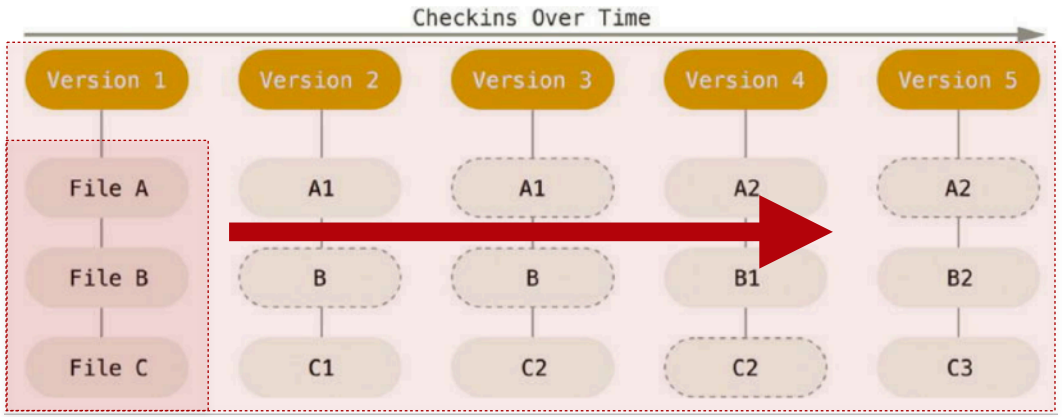
Git Basics:

Snapshots, Not Differences

Git memorizza snapshots del filesystem

Commit = foto/snapshot dello stato del filesystem in un dato istante

Repository = Stream di snapshots



Git Basics:

Snapshots, Not Differences

Più simile a un mini-filesystem…

… con tools aggiuntivi particolarmente potenti.

(Linus Torvalds)

Git Basics:

Nearly Every Operation Is Local

- **Quasi ogni operazione è locale** *(non ha bisogno di accedere a info remote)*
  - Scorrere la storia del progetto
  - Ripristinare una versione precedente (checkout)
  - Commit
  - Branch, Merge
- **Lavorare offline** *(es: develop, commit, branch, merge, stash, patch… upload differito)*
  - Aereo
  - Treno
  - Off-VPN
  - In caso di guasti o comunque senza connessione per qualunque motivo

Git Has Integrity

- **Tutto è check-summed prima di essere archiviato**
  - Strettamente integrato in Git al più basso livello
  - Impossibili variazioni, corruzioni, perdite di informazioni che Git non sia in grado di rilevare
  - Il tipo di check-sum usato è SHA-1 *(40 car. es: "24b9da6552252987aa493b52f8696cd6d3b00373")*
  - E' usato anche come nome per i commit *(No filenames, bastano i primi 7 caratteri es: "24b9da6")*
- **Non cancella nulla, aggiunge solo**
- **E' veramente difficile convincere Git a fare qualcosa che causi perdita di informazioni o che non sia annullabile**

Git

…caratteristiche

Distribuito

Sicuro

Veloce

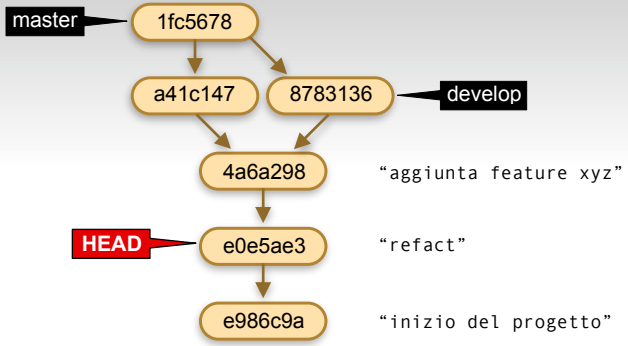
Forte supporto allo sviluppo non lineare

(migliaia di branches paralleli)

Repository

2 tipi di oggetti

- **Commit**
  - Rappresenta lo stato di un progetto in un dato istante *(foto/snapshot)*
  - Nome *(SHA-1)*
  - Descrizione
  - Riferimento a genitore
  - Un oggetto commit può avere più di un genitore.



- **Intestazioni (Heads)**
  - Puntatore a un oggetto commit
  - Nome / Titolo *(alla creazione del repository viene creata una intestazione '**master**' per default)*
  - Possono esserci molteplici intestazioni
  - **HEAD:** speciale intestazione che punta all'oggetto commit su cui stiamo lavorando. *(working directory) (checkout) (attenzione al maiuscolo)*

Repository

2 “azioni” principali + 1

Branch *(o ramo, braccio)*

- Una linea indipendente di sviluppo
- Ha un nome
- C’è sempre un ramo principale “master”
- Sperimentare/testare nuove feature o bug-fix senza introdurre anomalie nel ramo principale
- Lavorare in team

Merge

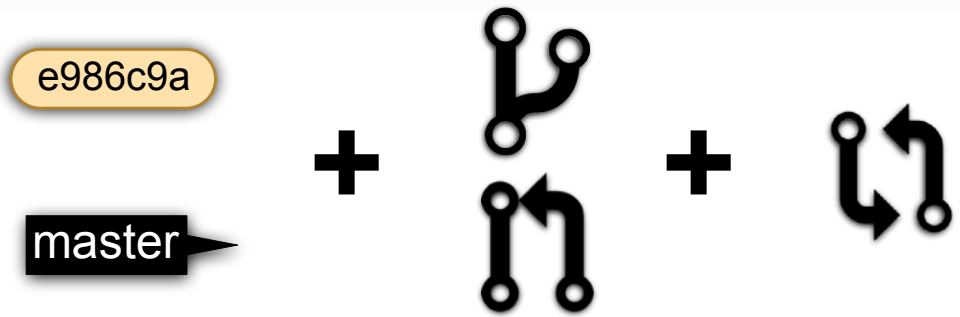
- Fusione tra due rami
- 3 tipi di merge: fast forward, clean, conflict

Push/Pull

- **Push:** scrivere sul repository remoto (origin) le modifiche fatte nel nostro repository locale *(upload)*
- **Pull:** leggere dal repository remoto (origin) le modifiche fatte da altri *(download)*

Git Basics:

2 + 2 + 1 = git



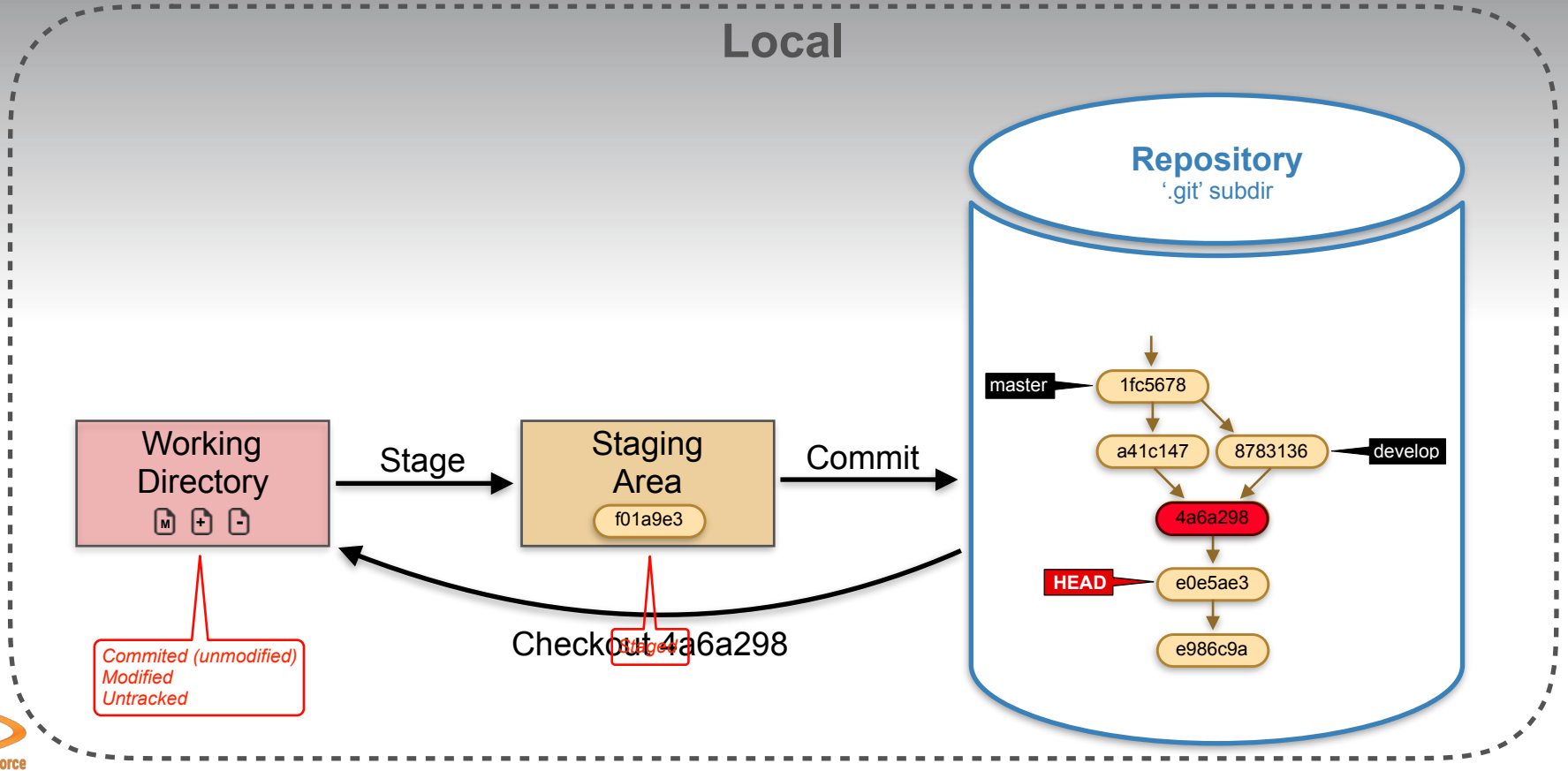
Files: 3 Stati Possibili

1. **Committed:** già salvato in modo sicuro nel repository locale
2. **Modified:** modificato e non ancora committato (modifiche pendenti)
3. **Staged:** file modificato marcato per essere incluso nel prossimo commit/snapshot

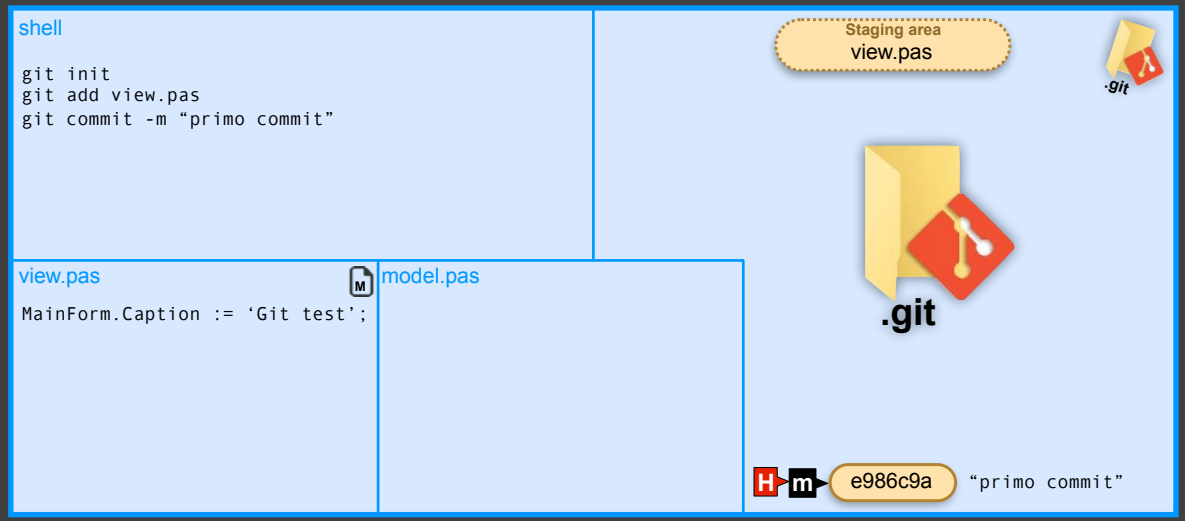
Untracked: file non tracciato da Git

Git Basics:

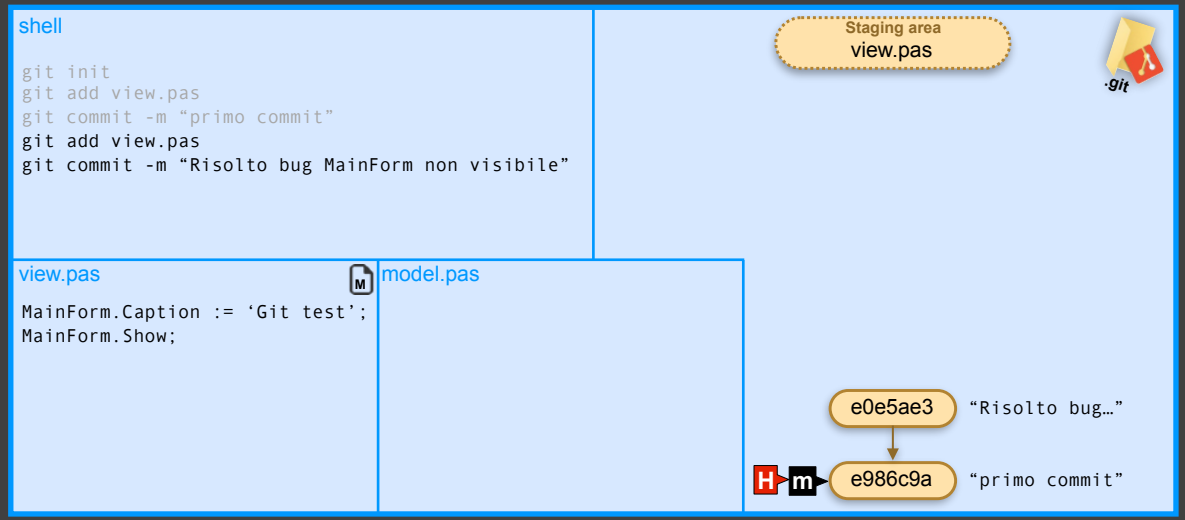
Repository: 3 Sezioni



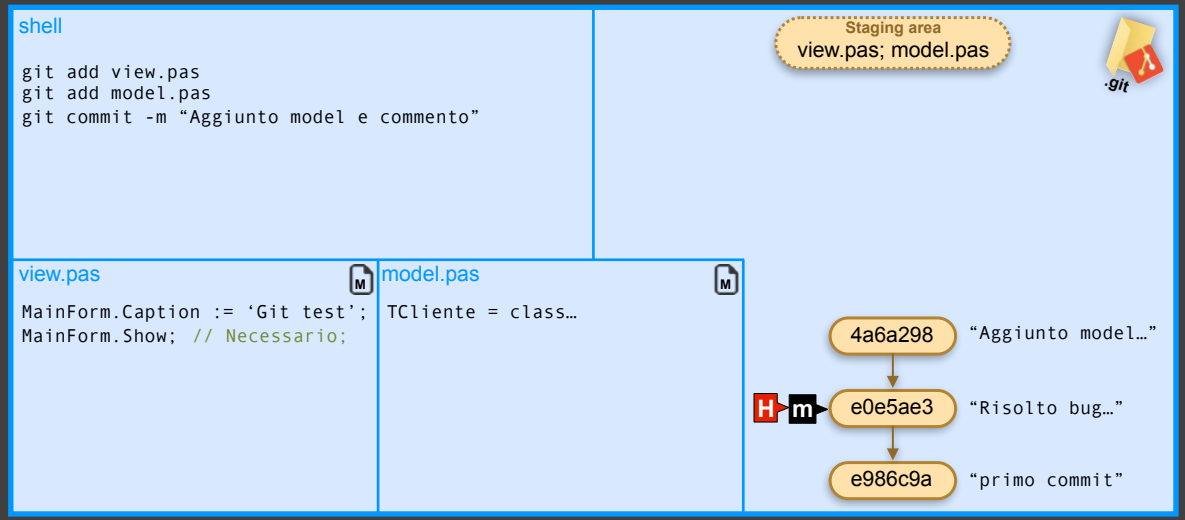
1) Inizializzazione e primo commit



2) Modifichiamo il file



3) Un altro commit?



4) Come era prima?

shell

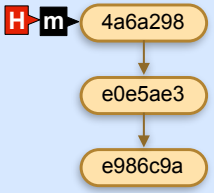
```
git checkout e0e5ae3
```

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show; // Necessario;
```

model.pas

```
TCliente = class…
```



5) Ci chiedono una nuova feature (ma senza toccare il master)

shell

```
git checkout e0e5ae3
git branch feature1
git checkout feature1
git add .
git commit “Nuova classe cliente…”
```

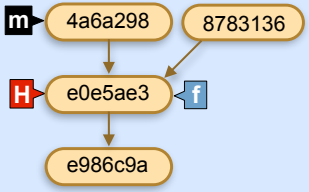
Staging area
view.pas; model.pas

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show;
LCliente := TNewCliente.Create;
```

model.pas

```
TNewCliente = class…
```



6) Continuiamo a lavorare sulla nuova feature

shell

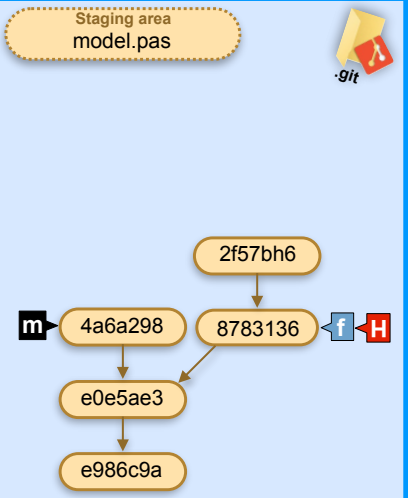
```
git checkout e0e5ae3
git branch feature1
git checkout feature1
git add .
git commit “Nuova classe cliente…”
git commit -a -m “aggiunto proprietà Age”
```

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show;
LCliente := TNewCliente.Create;
```

model.pas

```
TNewCliente = class…
property Age: integer read FAge;
```



7) Ora però dobbiamo tornare alla versione di produzione

shell

```
git checkout e0e5ae3
git branch feature1
git checkout feature1
git add .
git commit “Nuova classe cliente…”
git commit -a -m “aggiunto proprietà Age”
git checkout master
git commit -a -m “visualizzazione età cliente”
```

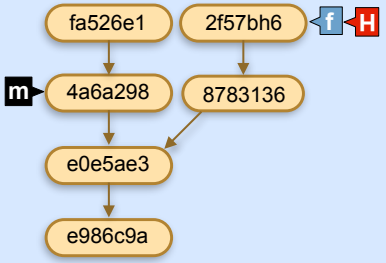
Staging area
view.pas

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show; // Necessario;
```

model.pas

```
property Age: integer read FAge;
```



8) Merge (no conflict)

shell

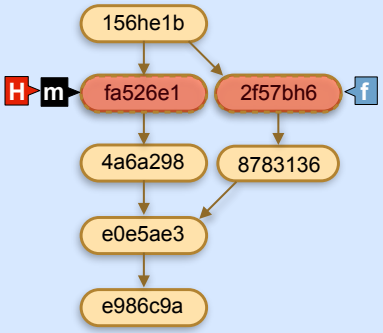
```
git merge feature1
```

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show;  // Necessario
LCliente := TNewCliente.Create;
Label1.Caption := LCliente.Age;
```

model.pas

```
TNewCliente = class
property Age: integer read FAge;
```



9) Merge (with conflict)

shell

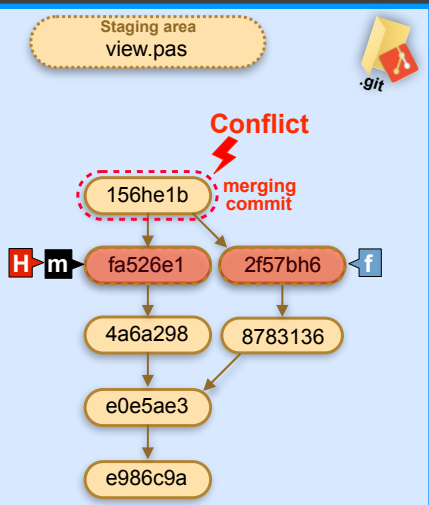
```
git merge feature1
Auto-merging model.pas
CONFLICT (content): Merge conflict in model.pas
Automatic merge failed; fix conflicts and then
commit the result.
git add model.pas
git commit “feature1 merged on master”
```

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show;  // Necessario
Label1.Caption := LCliente.Age;
```

model.pas

```
TCliente = class…
<<<<<<< HEAD
TCliente = c…
=======
TNewCliente…s…
property Age: integer read FAge;
>>>>>>> feature1
…
```



10) Eliminiamo il branch (oppure no?)

shell

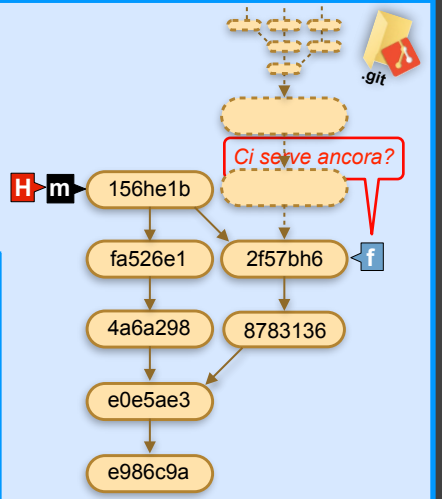
```
git merge feature1
Auto-merging model.pas
CONFLICT (content): Merge conflict in model.pas
Automatic merge failed; fix conflicts and then
commit the result.
git add model.pas
git commit “feature1 merged on master”
git branch -D feature1
```

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show;  // Necessario
LCliente := TNewCliente.Create;
Label1.Caption := LCliente.Age;
```

model.pas

```
TNewCliente = class…
property Age: integer read FAge;
```



11) Se branch il viene eliminato prima del merge

reflog command
git branch NewBranchName SHA1
git checkout -b NewBranchName SHA1

shell

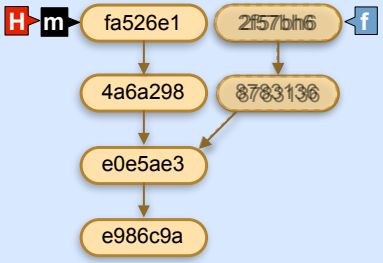
```
git branch -D feature1
git reflog

fa526e1 HEAD@{0}: checkout: moving from branch feature1 to master
2f57bh6 HEAD@{1}: commit: visualizzazione età cli…
8783136 HEAD@{2}: commit: aggiunto proprietà Age
E0e5ae3 HEAD@{4}: checkout: moving from branch master to feature1
…
git branch oldfeature1 HEAD@{1}
```

view.pas

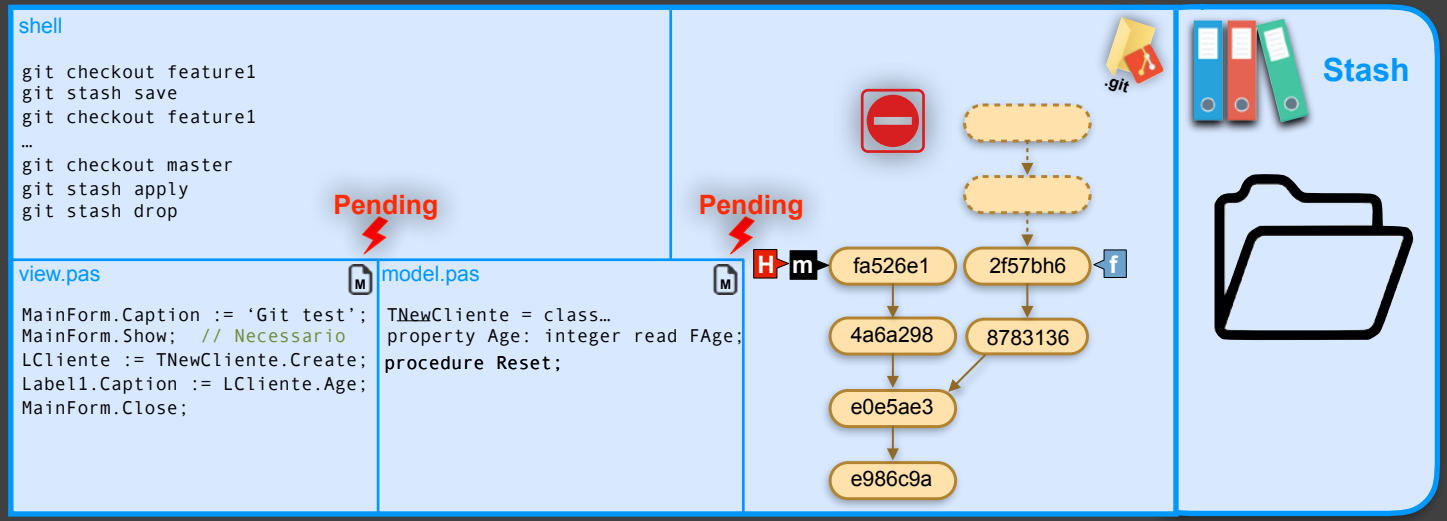
```
MainForm.Caption := 'Git test';
MainForm.Show;  // Necessario
LCliente := TNewCliente.Create;
Label1.Caption := LCliente.Age;
```

model.pas

```
TNewCliente = class…
property Age: integer read FAge;
```



12) Stash


shell
git checkout feature1
git stash save
git checkout feature1
…
git checkout master
git stash apply
git stash drop
Pending
Pending
view.pas
MainForm.Caption := 'Git test';
MainForm.Show; // Necessario
LCliente := TNewCliente.Create;
Label1.Caption := LCliente.Age;
MainForm.Close;
model.pas
TNewCliente = class…
property Age: integer read FAge;
procedure Reset;
fa526e1
2f57bh6
4a6a298
8783136
e0e5ae3
e986c9a
Stash

13) Rebase (no conflict)
Fast forward merge alla fine

shell

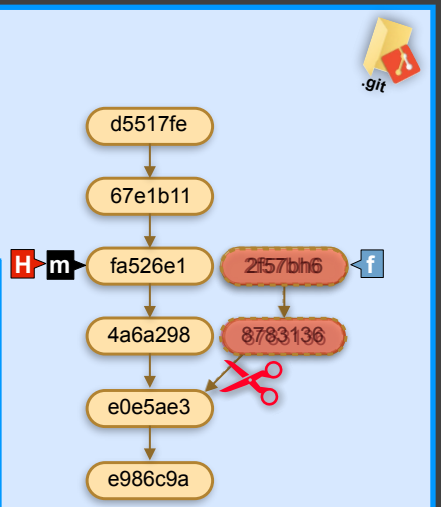
```
git checkout feature1
git rebase master
First, rewinding head to replay your work on top of it...
Applying: commit
Applying: commit
git checkout master
git merge feature1
git branch -D feature1
```

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show;  // Necessario
LCliente := TNewCliente.Create;
Label1.Caption := LCliente.Age;
```

model.pas

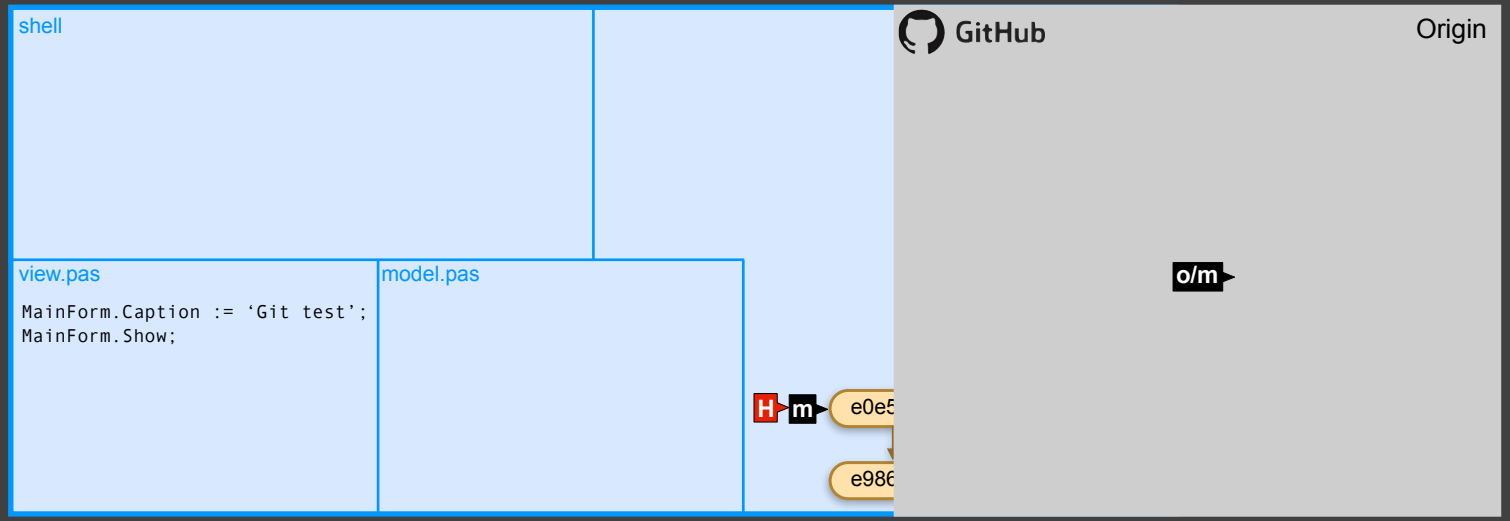
```
TNewCliente = class…
property Age: integer read FAge;
```



Non fare rebase su rami già inviati a un repository remoto

14) Cominciamo a collaborare

Creazione repository remoto su GitHub



15) creare un repository remoto (Origin)

git remote -v
colleghiamo il repository remoto (origin) al nostro repository locale
colleghiamo il nostro branch master locale con il master remoto
cacciamo il nostro primo Push

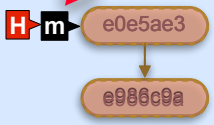
shell

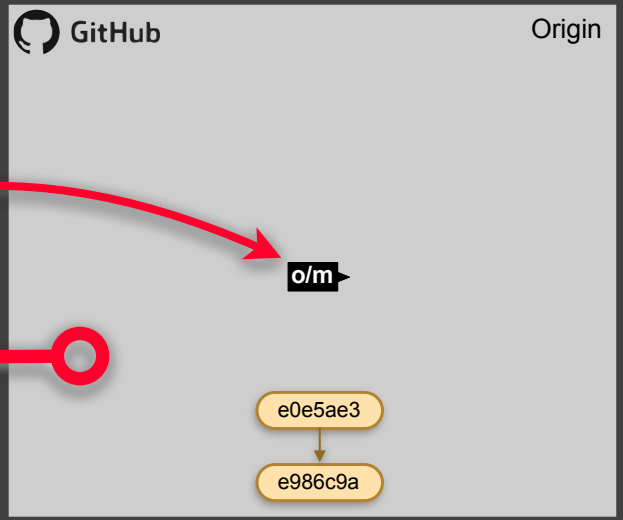
```
git remote add origin https://github.com/mauriziodm/delphiday2019.git
git push –set-upstream origin master
git push
```

view.pas

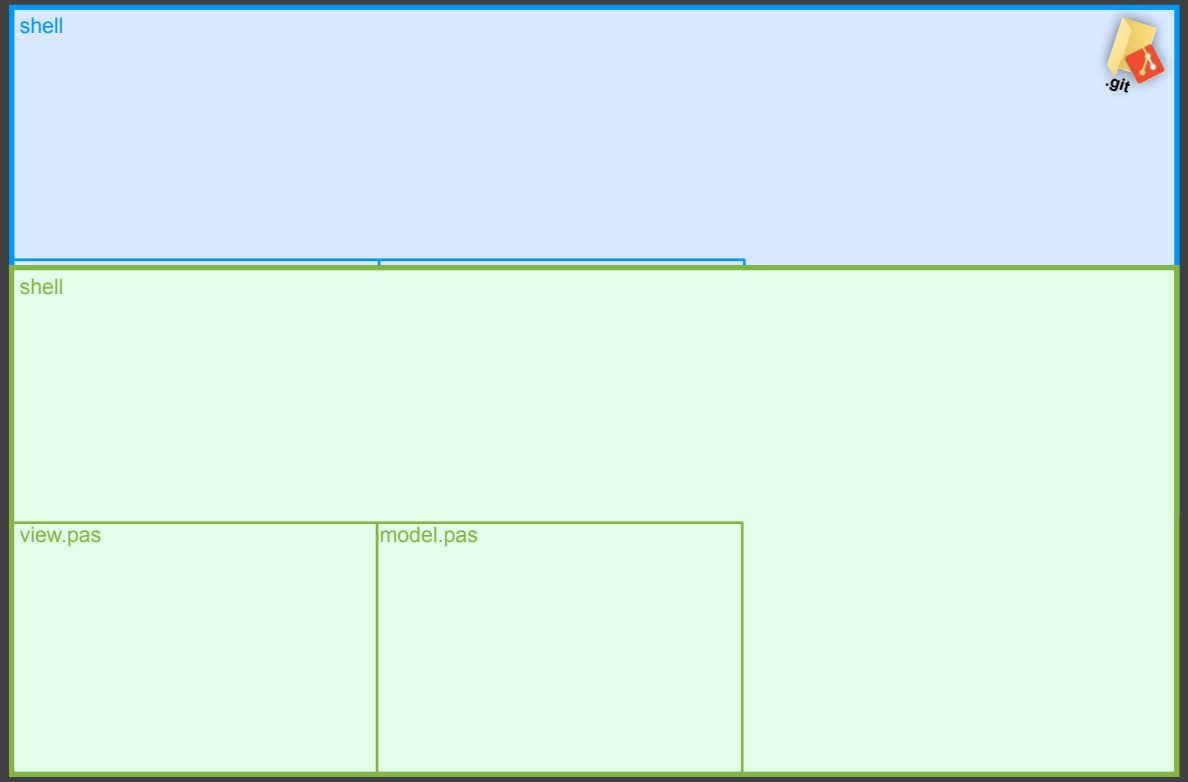
```
MainForm.Caption := 'Git test';
MainForm.Show;
```

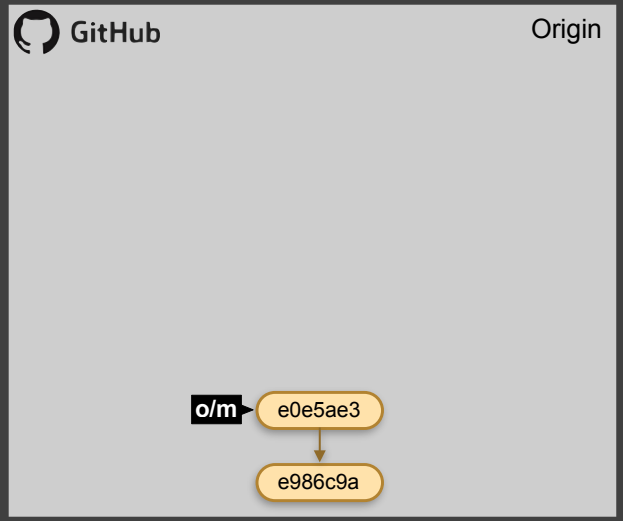
model.pas





16) Un nuovo contributor





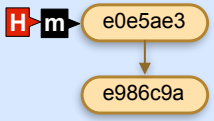
17) Un nuovo contributor
git clone

shell

view.pas

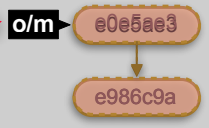
```
MainForm.Caption := 'Git test';
MainForm.Show
```

model.pas



GitHub

Origin



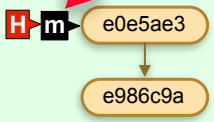
shell

```
git clone https://mauriziodm:password@github.com/mauriziodm/delphiday2019.git
```

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show;
```

model.pas



18) Un nuovo contributor
push/pull (same branch, no conflict)

shell

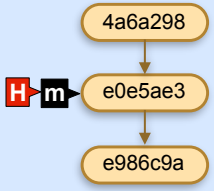
```
git fetch
git status
On branch master
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
git pull
```

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show
```

model.pas

```
TNewCliente = class…
```



shell

```
git clone https://mauriziodm:password@github.com/mauriziodm/delphiday2019.git
git commit -a -m "aggiunta classe TNewCliente"
git push
```

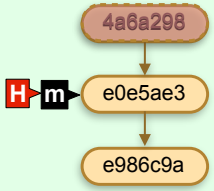
Staging area
model.pas

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show;
```

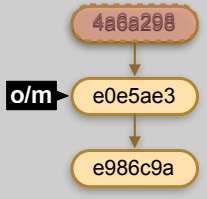
model.pas

```
TNewCliente = class…
```



GitHub

Origin



DelphiDay

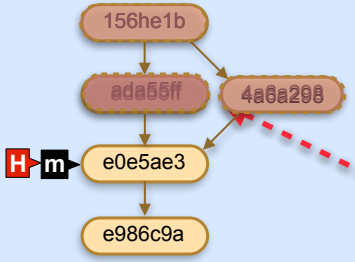
19) Un nuovo contributor
push/pull (same branch, divergent)

shell
```
git commit -a -m “aggiunta classe TOtherCliente”
git push
git pull
git push
```

Staging area
model.pas

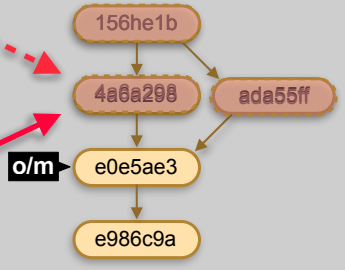
view.pas
```
MainForm.Caption := ‘Git test’;
MainForm.Show
```

model.pas
```
TOtherCliente = class…
```



GitHub
Origin

Divergent

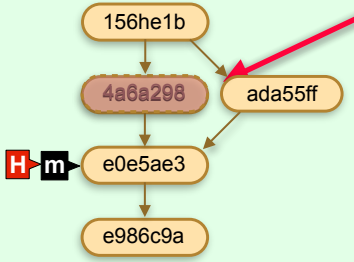


shell
```
git clone https://mauriziodm:password@github.com/mauriziodm/delphiday2019.git
git commit -a -m “aggiunta classe TNewCliente”
git push
git pull
```

Staging area
model.pas

view.pas
```
MainForm.Caption := ‘Git test’;
MainForm.Show;
```

model.pas
```
TNewCliente = class…
```



DelphiDay

20) Un nuovo contributor
push/pull (same branch, divergent+conflict)

shell

```
git commit -a -m “aggiunta classe TOtherCliente”
git push
git pull
git add model.pas
git commit -m “fatto merge TNewCliente”
git push
```

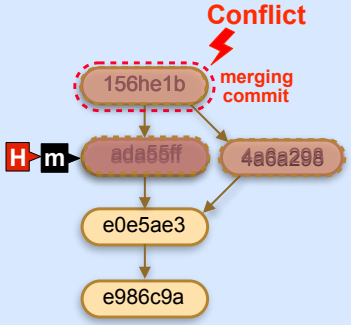
Staging area
model.pas

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show
```

model.pas

```
TOtherCliente = class…
<<<<<<< HEAD
TOtherCliente = class…
=======
TNewCliente = class…
>>>>>>> origin/master
…
```



shell

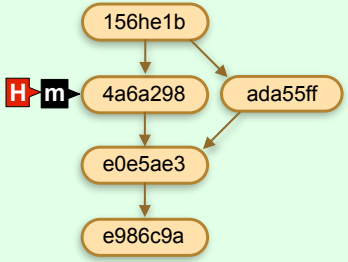
```
git clone https://mauriziodm:password@github.com/mauriziodm/delphiday2019.git
git commit -a -m “aggiunta classe TNewCliente”
git push
git pull
```

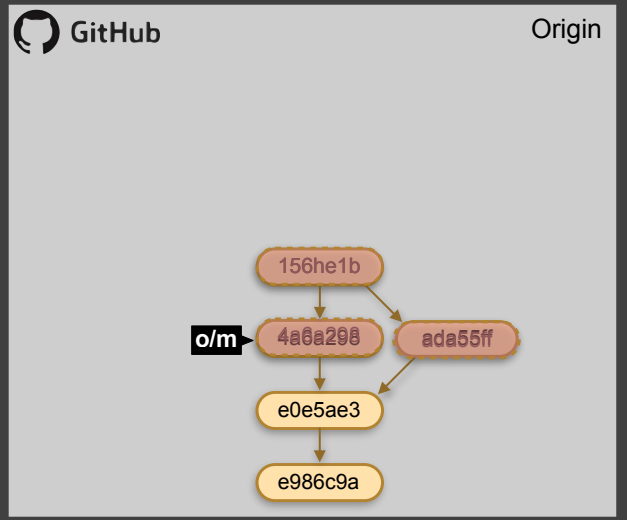
view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show;
```

model.pas

```
TNewCliente = class…
```





21) Un nuovo contributor
branch separati

shell

```
git commit -a -m “aggiunto classe TOtherCliente”
git push
git fetch
git pull
```

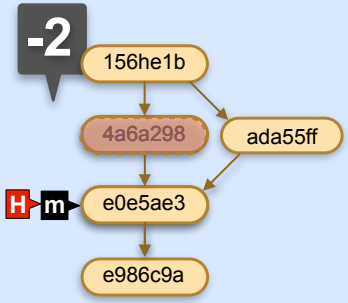
Staging area
model.pas

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show;
```

model.pas

```
TOtherCliente = class…
```



shell

```
git branch newfeature
git checkout newfeature
git commit -a -m “nuova classe TNewCliente”
git push
git checkout master
git fetch
git pull
git merge newfeature
git push
```

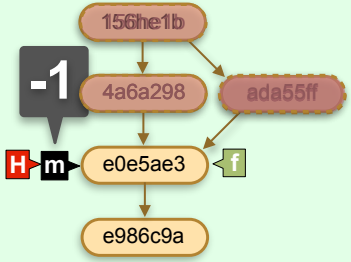
Staging area
model.pas

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show;
```

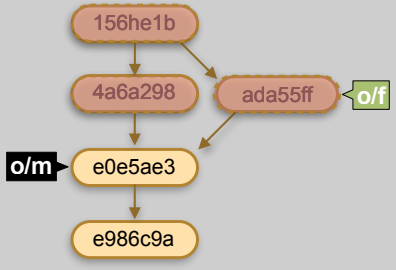
model.pas

```
TNewCliente = class…
```



GitHub

Origin



DelphiDay

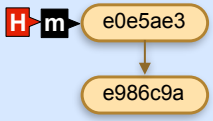
22) Se il contributor non ha i diritti?
fork

shell

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show
```

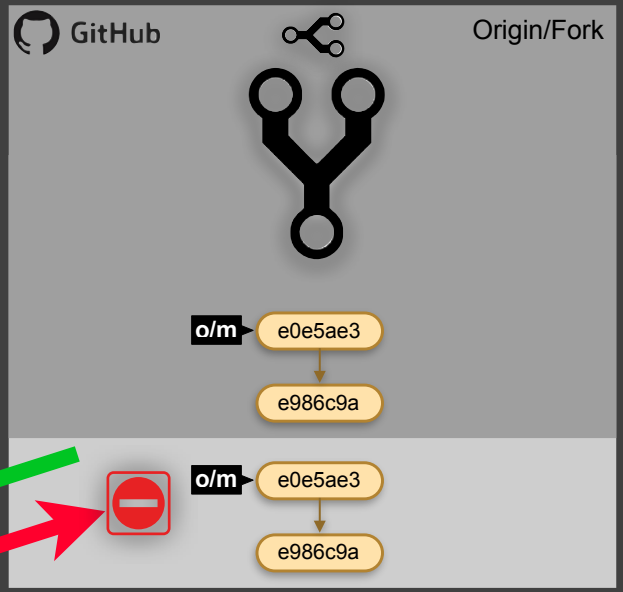
model.pas



shell

view.pas

model.pas



23) Se il contributor non ha I diritti?

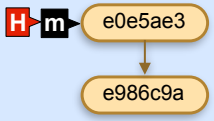
Fork: clone locale

shell

view.pas

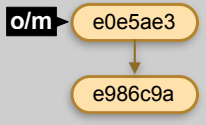
```
MainForm.Caption := 'Git test';
MainForm.Show
```

model.pas



GitHub

Origin



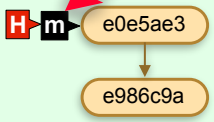
shell

```
git clone https://rinobosco70:BoscoRino70!!@github.com/mauriziodm/delphiday2019.git
```

view.pas

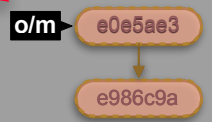
```
MainForm.Caption := 'Git test';
MainForm.Show;
```

model.pas



GitHub

Origin/Fork



24) Se il contributor non ha I diritti?

New Branch (ora possiamo)
Push

shell

```
git commit -a -m “aggiunto classe TOtherCliente”
git push
```

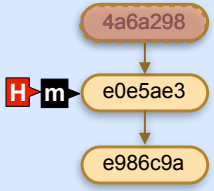
Staging area
model.pas

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show
```

model.pas

```
TOtherCliente = class…
```



shell

```
git branch newfeature
git checkout newfeature
git commit -a -m “nuova classe TNewCliente”
git push
```

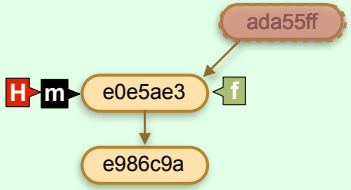
Staging area
model.pas

view.pas

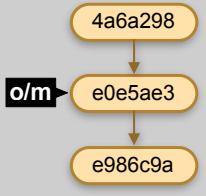
```
MainForm.Caption := ‘Git test’;
MainForm.Show;
```

model.pas

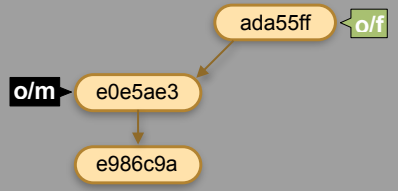
```
TNewCliente = class…
```



GitHub — Origin



GitHub — Origin/Fork



25) Se il contributor non ha I diritti?

Pull request (su GitHub)

shell

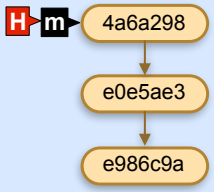
```
git commit -a -m “aggiunto classe TOtherCliente”
git push
```

view.pas

```
MainForm.Caption := ‘Git test’;
MainForm.Show
```

model.pas

```
TOtherCliente = class…
```



shell

```
git branch newfeature
git checkout newfeature
git commit -a -m “nuova classe TNewCliente”
git push
git commit -a -m “aggiunto proprietà age”
git push
```

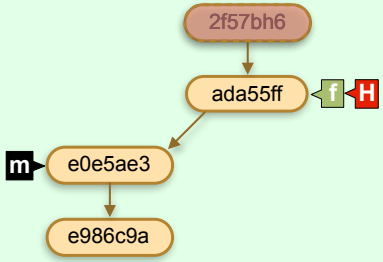
Staging area
model.pas

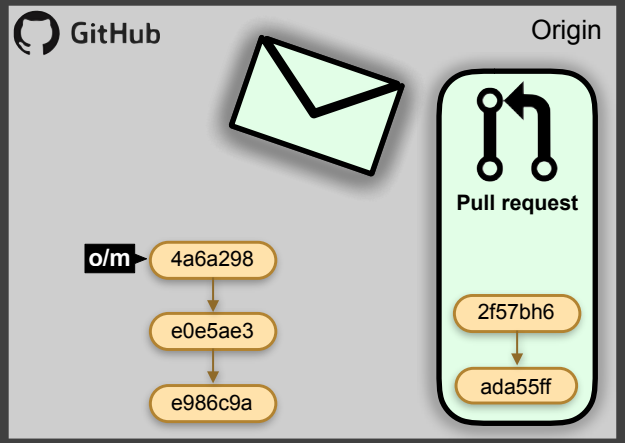
view.pas

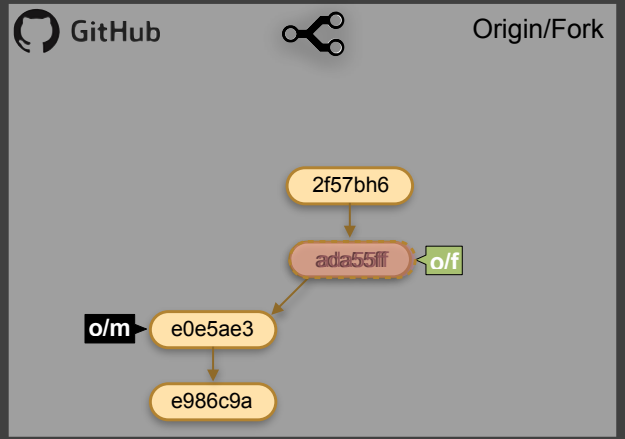
```
MainForm.Caption := ‘Git test’;
MainForm.Show;
```

model.pas

```
TNewCliente = class…
property Age: integer read FAge;
```







26) Se il contributor non ha I diritti?

Pull request: merge a cura del manteiner

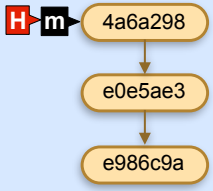
shell

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show
```

model.pas

```
TOtherCliente = class…
```



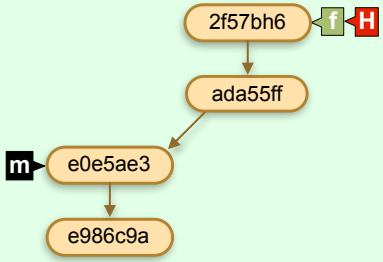
shell

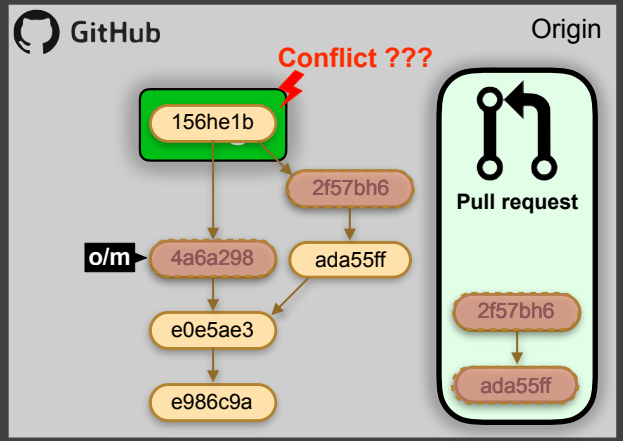
view.pas

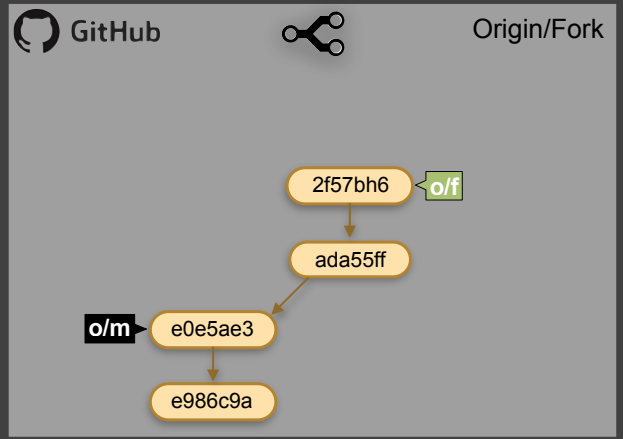
```
MainForm.Caption := 'Git test';
MainForm.Show;
```

model.pas

```
TNewCliente = class…
property Age: integer read FAge;
```







27) Se il contributor non ha I diritti?

Pull request: merge a cura del contributor

shell

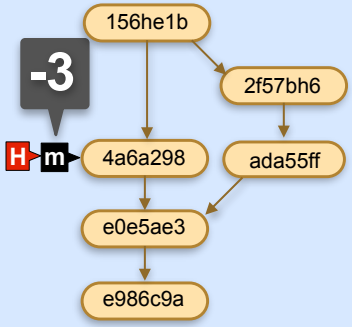
```
git fetch
git pull
```

view.pas

```
MainForm.Caption := 'Git test';
MainForm.Show
```

model.pas

```
TOtherCliente = class…
```



shell

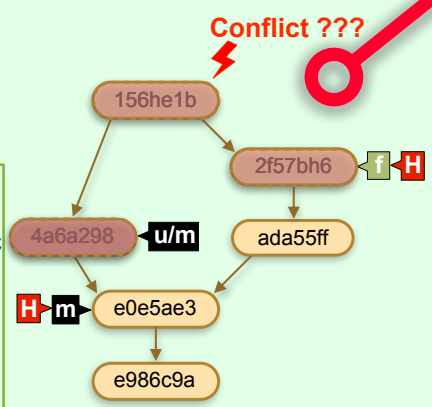
```
git remote add upstream https://github.com/mauriziodm/delphiday2019.git
git fetch upstream
git merge upstream/master
git push
```

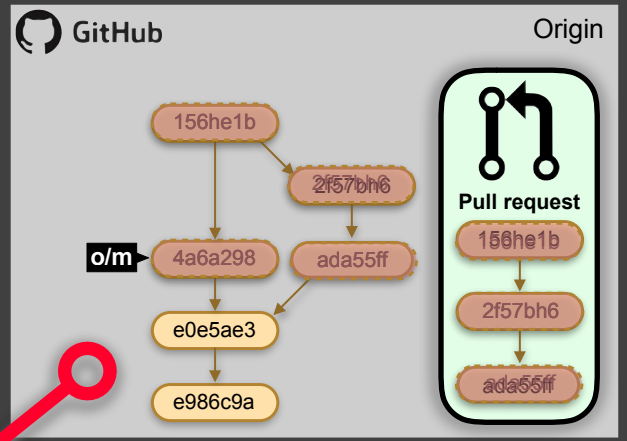
view.pas

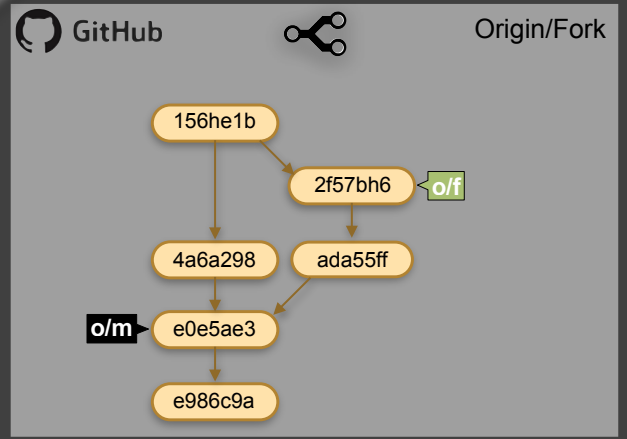
```
MainForm.Caption := 'Git test';
MainForm.Show;
```

model.pas

```
TNewCliente = class…
property Age: integer read FAge;
```







DOMANDE?

Maurizio Del Magno

Developer

levante software

i-ORM github.com/mauriziodm/iORM

DJSON github.com/mauriziodm/DJSON

mauriziodm@levantesw.it

mauriziodelmagno@gmail.com

facebook.com/maurizio.delmagno

iORM + DJSON (group)

Membro fondatore

eInvoice4D

https://github.com/delphiforce/eInvoice4D

DelphiDay italian conference

PIACENZA EDITION

2019 - XVIII Edizione

Grazie!!!

wintech italia

speaker: Maurizio Del Magno

embarcadero reseller partner