Version Control System

Un VCS e' un sistema software che permette di tenere traccia degli sviluppi di un progetto software.

Fondamentalmente i VCS sono nati per aiutare un team a sviluppare il proprio progetto software, tenedo traccia dei progressi fatti nella stesura del codice.

Quindi, primi aspetti, tra i benefici di un VCS …

- collaborazione (condivisione del codice)
- change tracking
    - history, commenti sui cambi, comparazione di version del codice
    - ripristino

(SCM - Source Code Management; RCS - Revision Control System).

Ogni membro del team vede il progetto, o il modulo in sviluppo, secondo la usuale organizzazione gerarchica … file e folder. Ognuno fa modifiche; ognuno lavora su una parte, permettendo agli altri di lavorare su altre parti, o magari le stesse; il VCS aiuta a risolvere eventuali modifiche conflittuali; il codice puo' essere sperimentato, e una successiva versione puo' essere stabilita.

- Experiment on the changes
- branching (gestire differenti linee di soluzione per il codice, poi convergenti)
- backup

Version Control System

Un progetto software e` organizzato come parte di un file system.

Per piccoli progetti personali il programmatore

- tiene traccia delle aggiunte di file, correzioni, nuovi gruppi di file
- sperimenta il Sistema nel suo sviluppo parziale
- sposta/copia file e cartelle per fare version diverse

- non prevede collaborazione, o almeno non si aspetta che la collaborazione sia semplice

Un VCS serve a gestire questo processo

- in modo piu` razionale
- permettendo la collaborazione

```
Projects
|
|----Customer A
     |---- Project 1
     |     |---- BuildControl       (CI/nAnt/Make
     |     |---- Documentation      (In XML forma
     |     |---- Source
     |     |     |---- Component X
     |     |     |---- Component X.tests
     |     |     |---- Component Y
     |     |     |---- Component Y.tests
     |     |---- Testing
     |     Project 1.sln      (Has folders as per
     |---- Project 2
     |---- Shared/Common components
|----Customer B
|----Customer C
|----<Our Company>
     |---- Internal Project A
```

https://softwareengineering.stackexchange.com/questions/14582/how-do-you-organize-your-projects-folders

Un concetto fondamentale e` quello di REPOSITORY, dove il progetto e' memorizzato nel file system, con la sua history, le versioni progressive, i commenti al codice e alla sua evoluzione …

Version Control System

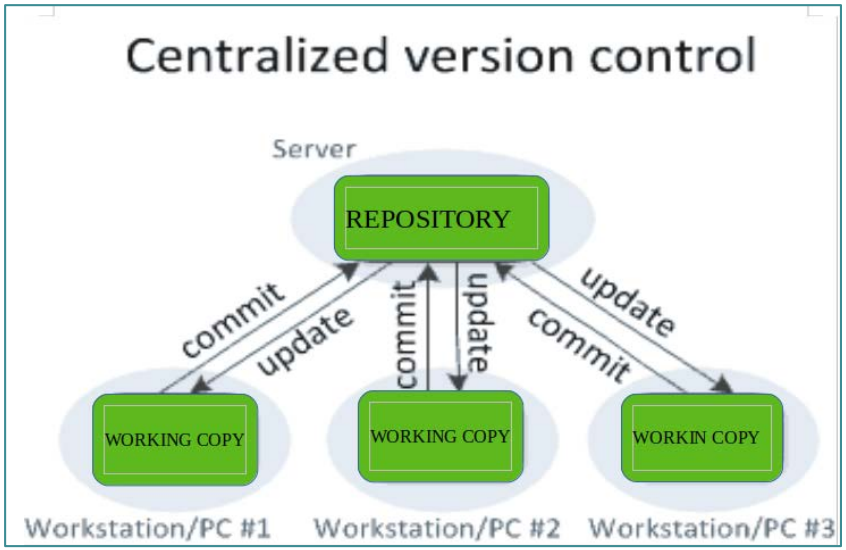
Un progetto software e' organizzato come parte di un file system.

Due tipi fondamentali di VCS aiutano a gestirlo, quando un team deve collaborare allo sviluppo:

- Centralized VCS (CVCS, es. Subversion, CVS)]
  - il repository e` centralizzato
  - ogni membro del team ha la sua copia locale, ci lavora sopra, sperimenta le soluzioni, e poi fa commit, cioe` fa allineare il progetto ai cambiamenti che ha introdotto
  - gli altri membri del team fanno update, per avere in locale la versione aggiornata

- Distributed VCS (DVCS, es. Git, Mercurial)
- (local VCS, es. RCS)

```
Projects
|
|----Customer A
     |---- Project 1
     |     |---- BuildControl       (CI/nAnt/Make
     |     |---- Documentation      (In XML forma
     |     |---- Source
     |     |     |---- Component X
     |     |     |---- Component X.tests
     |     |     |---- Component Y
     |     |     |---- Component Y.tests
     |     |---- Testing
     |     Project 1.sln      (Has folders as per
     |---- Project 2
     |---- Shared/Common components
|----Customer B
|----Customer C
|----<Our Company>
     |---- Internal Project A
```



Version Control System

Un progetto software e` organizzato come parte di un file system.

Due tipi fondamentali di VCS aiutano a gestirlo, quando un team deve collaborare allo sviluppo:

- Centralized VCS (CVCS, ex. Subversion, CVS)]
- Distributed VCS (DVCS, ex. Git, Mercurial)

I sistemi distribuiti introducono repository locale (copie, sincronizzabili, di quello centrale) e la differenza tra

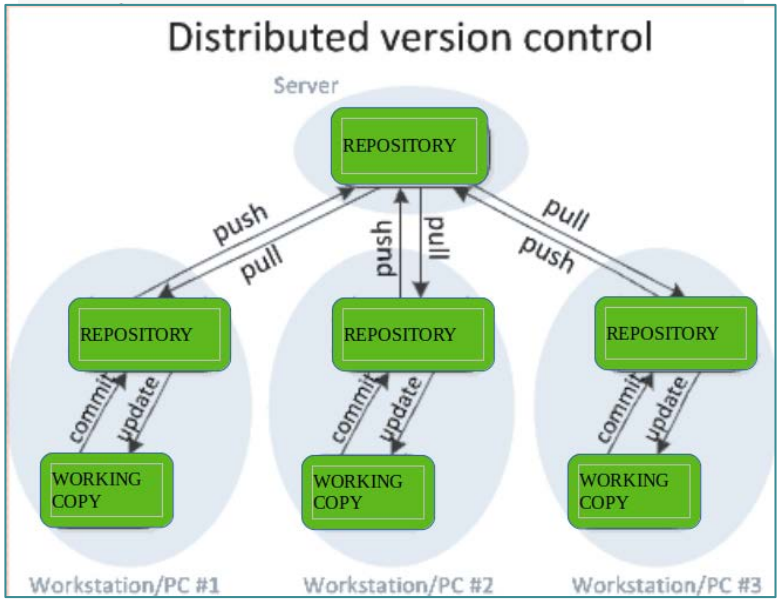
- fase di lavoro nella work area (file system)

- fase di staging (i file sono tracciati e sperimentabili, pronti per il commit), e

- fase di allineamento del repository locale con i cambi sperimentati (commit)

Questo permette di sperimentare la parte di sistema su cui si e' lavorato, prima di allineare anche il repository centrale (push)

```
Projects
|
|----Customer A
     |---- Project 1
     |     |---- BuildControl       (CI/nAnt/Make
     |     |---- Documentation      (In XML forma
     |     |---- Source
     |     |     |---- Component X
     |     |     |---- Component X.tests
     |     |     |---- Component Y
     |     |     |---- Component Y.tests
     |     |---- Testing
     |     Project 1.sln      (Has folders as per
     |---- Project 2
     |---- Shared/Common components
|----Customer B
|----Customer C
|----<Our Company>
```



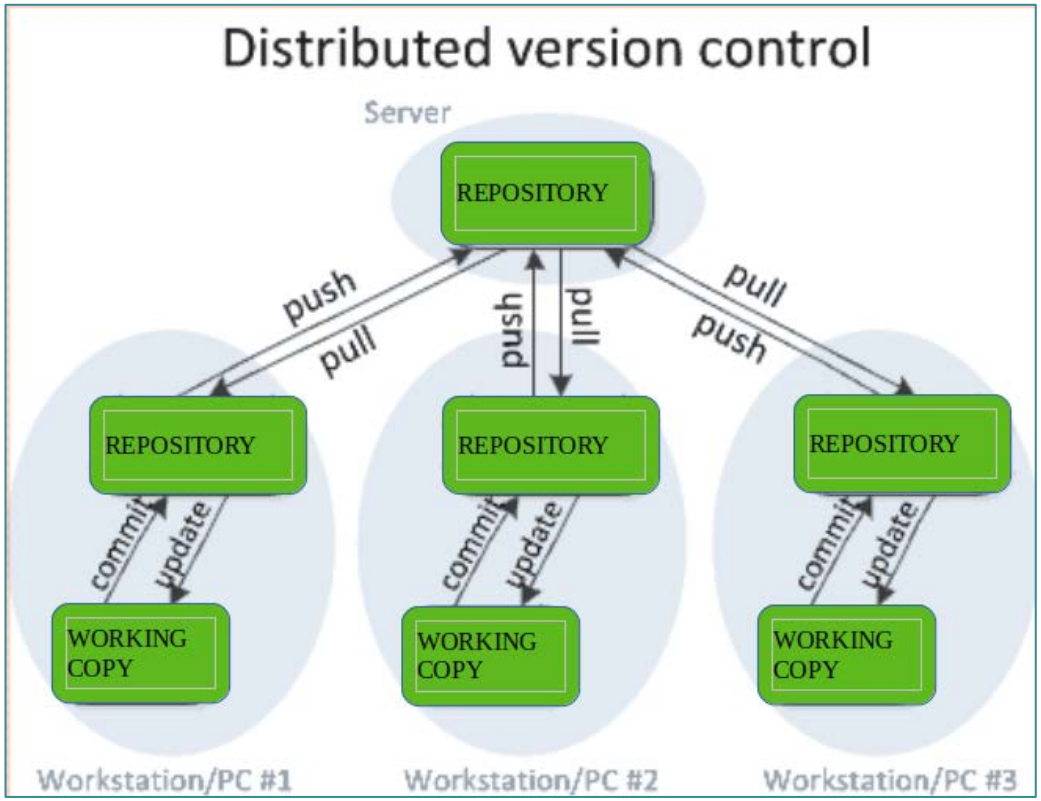
Version Control System

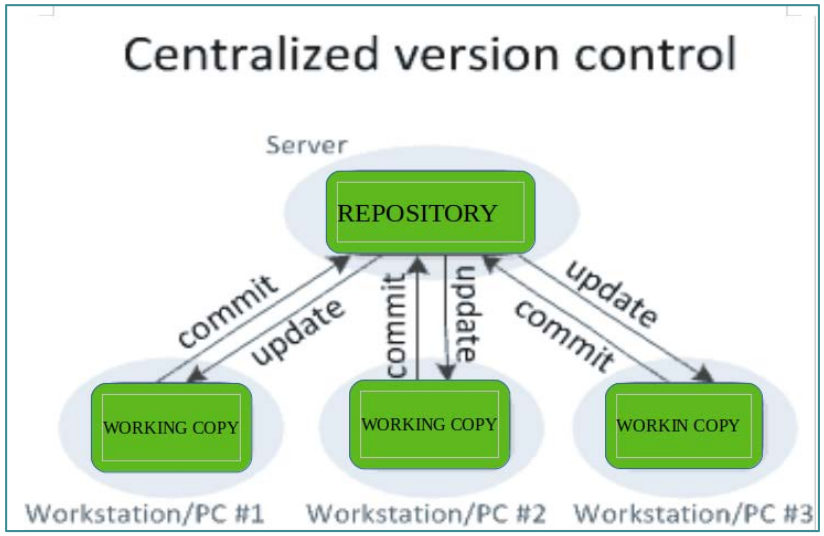
Un progetto software e` organizzato come parte di un file system.

Due tipi fondamentali di VCS aiutano a gestirlo, quando un team deve collaborare allo sviluppo:

- Centralized VCS (CVCS, ex. Subversion, CVS)
- Distributed VCS (DVCS, ex. Git, Mercurial)
- (local VCS)

```
Projects
|
|----Customer A
     |---- Project 1
     |      |---- BuildControl       (CI/nAnt/Make
     |      |---- Documentation      (In XML forma
     |      |---- Source
     |      |      |---- Component X
     |      |      |---- Component X.tests
     |      |      |---- Component Y
     |      |      |---- Component Y.tests
     |      |---- Testing
     |      Project 1.sln      (Has folders as per
     |---- Project 2
     |---- Shared/Common components
|----Customer B
|----Customer C
|----<Our Company>
     |---- Internal Project A
```





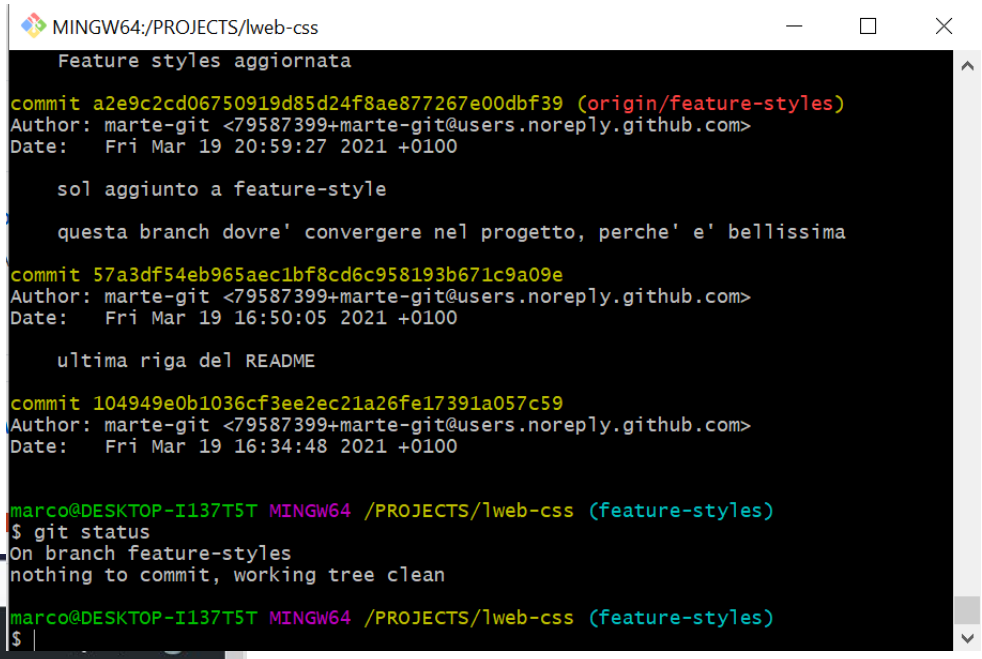
https://www.geeksforgeeks.org/version-control-systems/

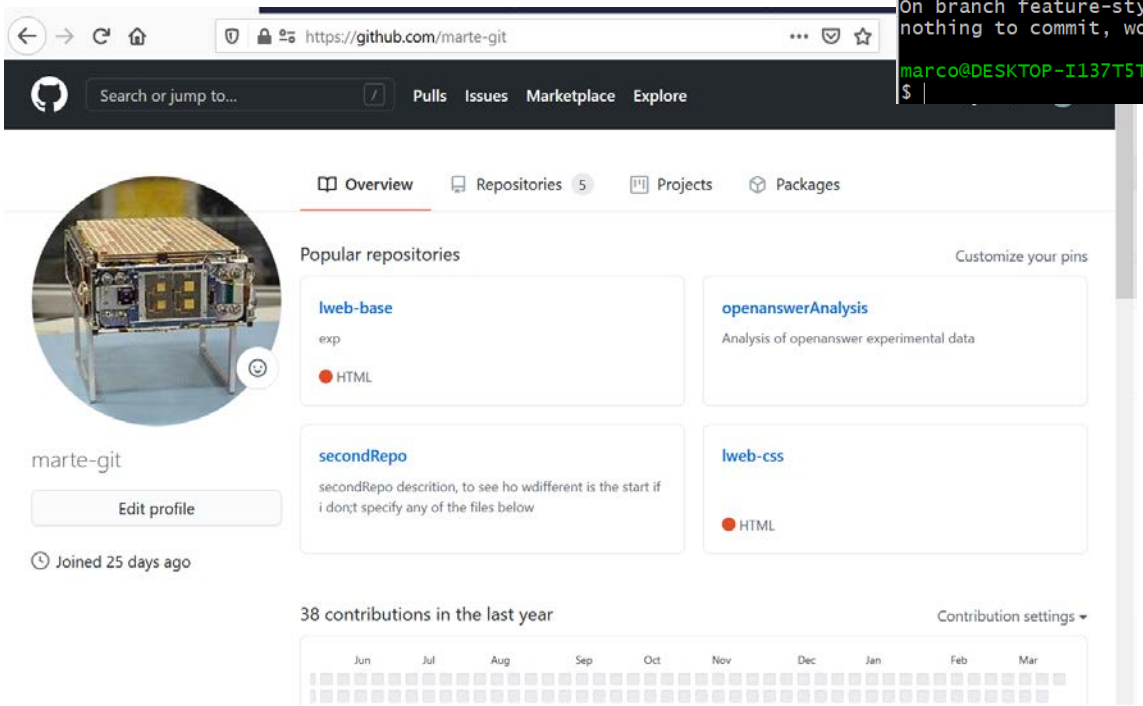
Version Control System: Git and Github

git e` il Sistema (tool, applicazione) locale che permette di gestire i repository locali

Ebbene si` : linea di comando.

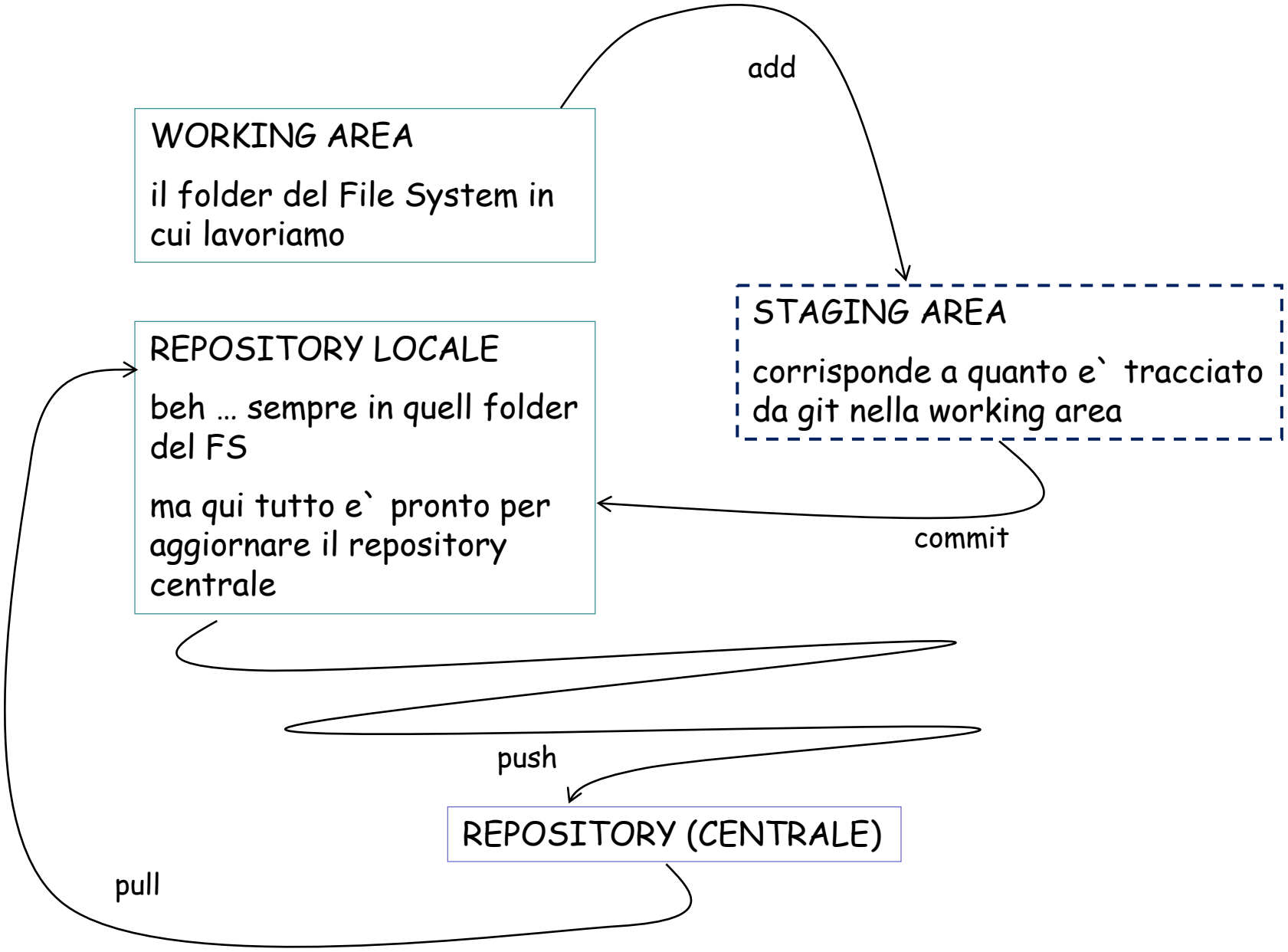
(Ma ci sono anche tools dotati di interfaccia grafica, es. GitHub Desktop).



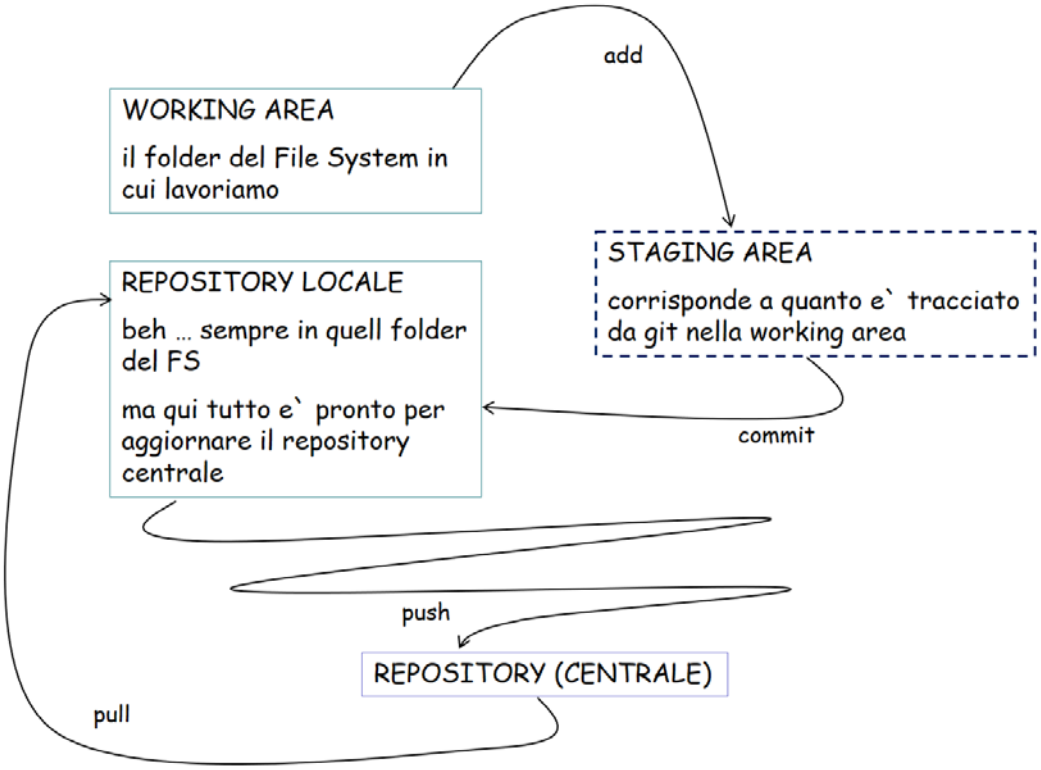


Github e` il sistema che permette la gestione del repository centrale e l'interazione con quelli locali, ed ovviamente e' basato su web.

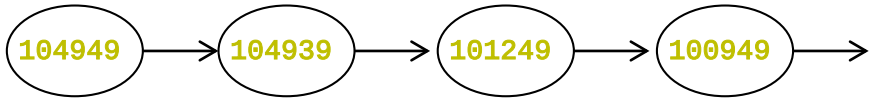
Git Workflow - basics



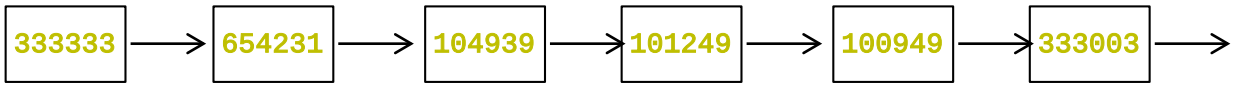
Git Workflow - basics



Tutti i cambiamenti (aggiunta file, correzione file, spostamento file ...) vengono tracciati e il sistema in sviluppo e` dato dalla serie dei commit

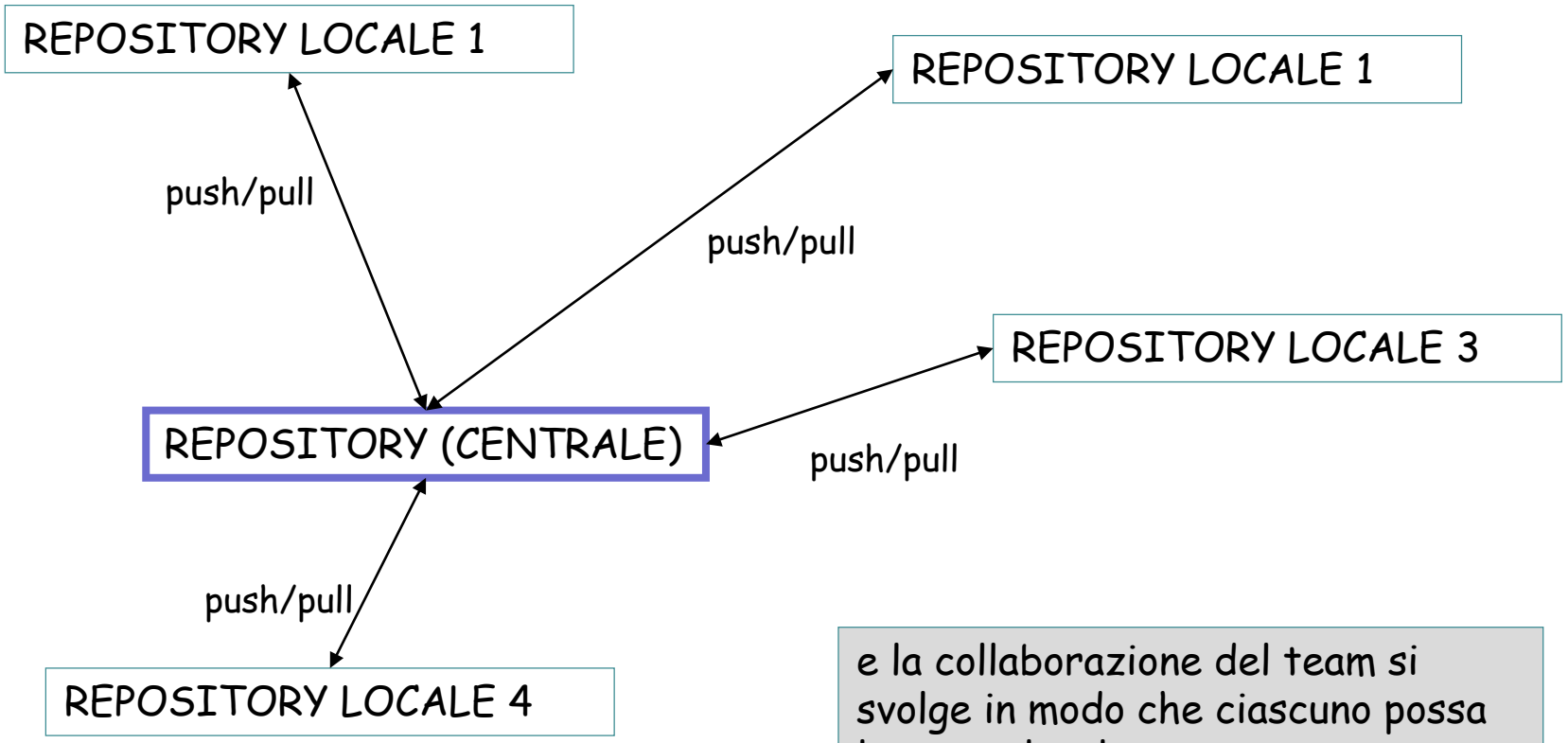


REPOSITORY LOCALE



REPOSITORY centrale

Git Workflow - basics



e la collaborazione del team si svolge in modo che ciascuno possa lavorare localmente, sincronizzandosi con il repository centrale

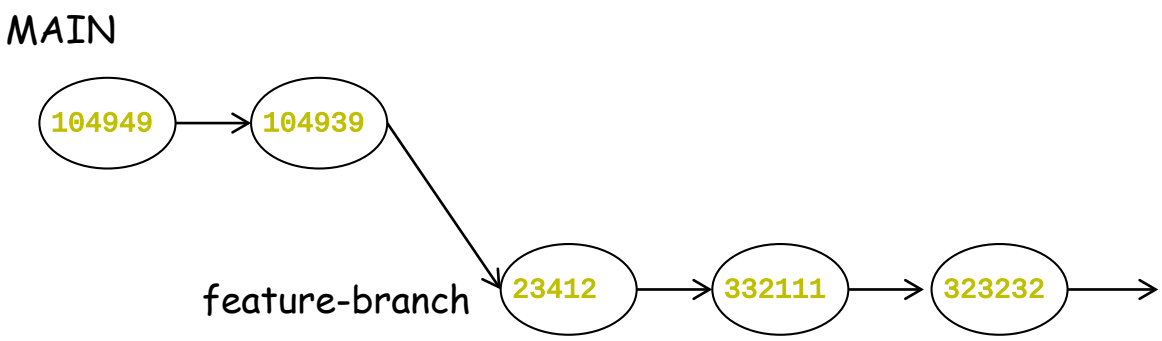
esperienze

```
git init                git clone                       git remote

git add                 git commit –m "" –m ""

git diff                git status                      git log

git push                git pull

git branch              git checkout                    git merge

git reset               git revert
```

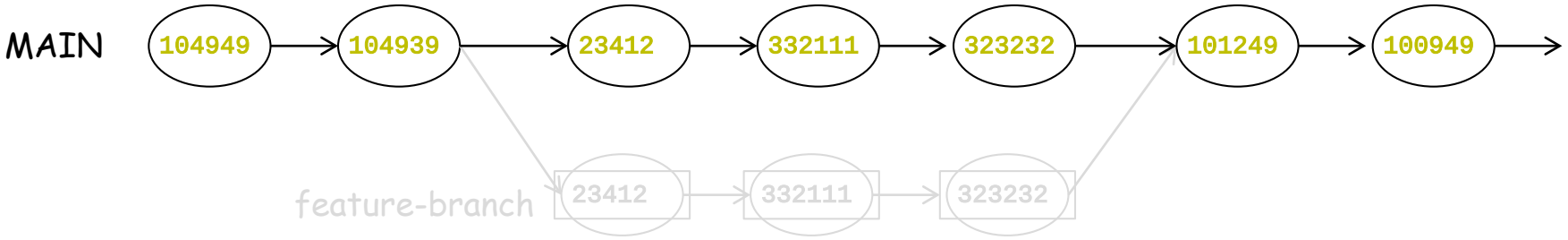
Git Workflow - branching

Servono a lavorare concentrandosi su una feature individuale del progetto …

La branch inizia in un punto del flusso di commit del repository, prosegue individualmente, senza disturbare la branch principale (MAIN, master …)



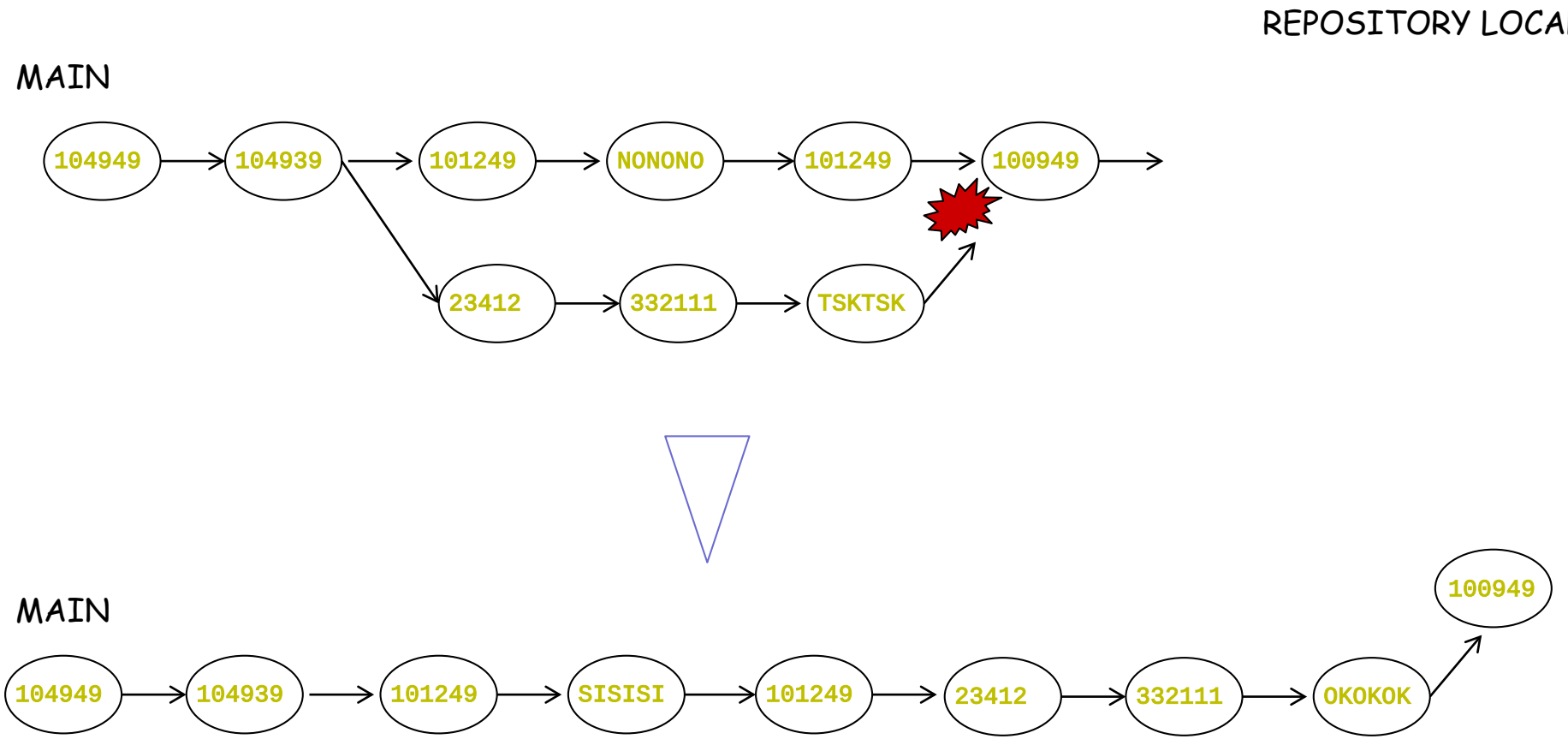
Poi, se si vuole, ci puo' essere un ricongiungimento e il progetto continua con la nuova feature (la branch puo' essere cancellata, oppure lasciata dov'e'.



Git Workflow - branching

ovviamente, come qualunque altra cosa nel VCS, … non e' magico …

possono esserci conflitti (che vanno risolti prima del merge)



esperienze

```
git init                git clone                       git remote

git add                 git commit –m "" –m ""

git diff                git status                      git log

git push                git pull

git branch              git checkout                    git merge

git reset               git revert
```

Risorse (1/2)

VCS

- *https://www.geeksforgeeks.org/version-control-systems/*

git

- *http://git-scm.com/*

- *eccellente tutorial* *https://www.youtube.com/watch?v=RGOj5yH7evk*

- *un sito con molte informazioni, peraltro citato anche dal precedente: https://www.atlassian.com/git/tutorials*
- *dal sito qui sopra, sui primi passi nell'istallazione di git: https://www.atlassian.com/git/tutorials/setting-up-a-repository*

e github *https://github.com/*

markdown *https://guides.github.com/features/mastering-markdown/*
https://github.com/ikatyang/emoji-cheat-sheet/blob/master/README.md

about branching *(almeno la prima parte e' utile; poi diventa piu' bello ... cioe' piu' complicato ma comunque fuori del nostro scope ... adatto solo a chi vuole approfondire: https://martinfowler.com/articles/branching-patterns.html*

Risorse (2/2)

SSHKEY

gestione della SSH key ... per evitare di doversi autenticare ad ogni push bisogna aver generato una propria ssh key (con keygen) ed aver configurato il sistema locale e github per usarla.
Le cose si fanno diversamente dipendentemente dal sistema operativo che si usa. Informazioni sono su
https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh
e anche https://www.atlassian.com/git/tutorials/git-ssh

le istruzioni di cui sopra vanno seguite … comunque, in breve

- quando, da git, viene richiesto un push, o anche un clone, l'interazione tra git (locale) e github (remoto) e' soggetta ad autenticazione; viene usata la chiave pubblica, che deve essere nota a github

- **`$ ls -al ~/.ssh`** si vede se c'e' gia' una chiave private nel Sistema (di solito si')
  Se c'e' si vede qualche file come *id_rsa.pub*, o *id_ecdsa.pub* , o *id_ed25519.pub*
- *se non c'e'… bisogna "generarla":*

```
$ ssh-keygen -t rsa -b 4096 -C "email@usata.per.github"
```

si puo' specificare un nome di file per contenere la chiave, altrimenti viene usato un nome di default; tipicamente id_rsa. Se non avete motive per definire un nome di file usate il default battendo return. Idem per la password … possiamo lasciarla in bianco. Vengono create due file "<nome>" e "<nome>.pub".

- poi, ad esempio con **`$ cat nome.pub`** *bisogna prendere il contenuto del file nome.pub (non quello nome!!!) e usarlo per inserire una nuova chiave su Github (menu' settings/SSH…)*

Attivita' pratica

git1

Considerare gli esercizi XHTML-CSS.

Sceglierne uno in cui sia necessario produrre un piccolo sito web con vari document web.

La scelta migliore e' per un esercizio che si e' svolto in parte.

Costruire un repository locale con git, in cui sia contenuto questo progetto; associarlo ad un repo remote su Github.

Proseguire il progetto e terminarlo mediante aggiunte e correzioni di file nel repository locale.

Quando si aggiunge un file o si fa una correzione, si opera nell'area di lavoro (che e' locale); poi si fa add delle cose su cui si e' un po' sicuri/e. Tenere presente che oltre ad "add" c'e' anche "rm". Poi si fa commit con le cose che sembrano funzionare. Occasionalmente puo' succedere di dovre ritrattare dei commit e porteare il repo in una situazione precedente. Spesso si fa push per avere un backup in remote, poter collaborare sul progetto, poter diffondere il progetto.

git2

Si sceglie un esercizio non ancora fatto; si costruisce un repo su Github e lo si usa per portare l'esercizio Avanti, anche se non a termine

Poi si clona il progetto in locale e si ripetono un po' di volte I seguenti passi

- modifiche in locale e push
- modifiche in remote e pull

git3

Partendo da uno dei progetti precedent si sviluppano due ulteriori feature

1) un gruppo di nuove pagine descrive un nuovo argometo; ci sara' qualche modifica in pagine esistemti (per esempio la home) in modo da indirizzarle;
2) alcune pagine esistenti vengono modificate, producendo un sito lievemente modificato (ad esempio nello stile di presentazione)

Le due feature vengono realizzate mediante branch.

Poi, quando almeno una delle branch si e' sviluppata abbastanza, si fa merge.

git4

ripetere l'esercizio git3, ma collaborando con un college.