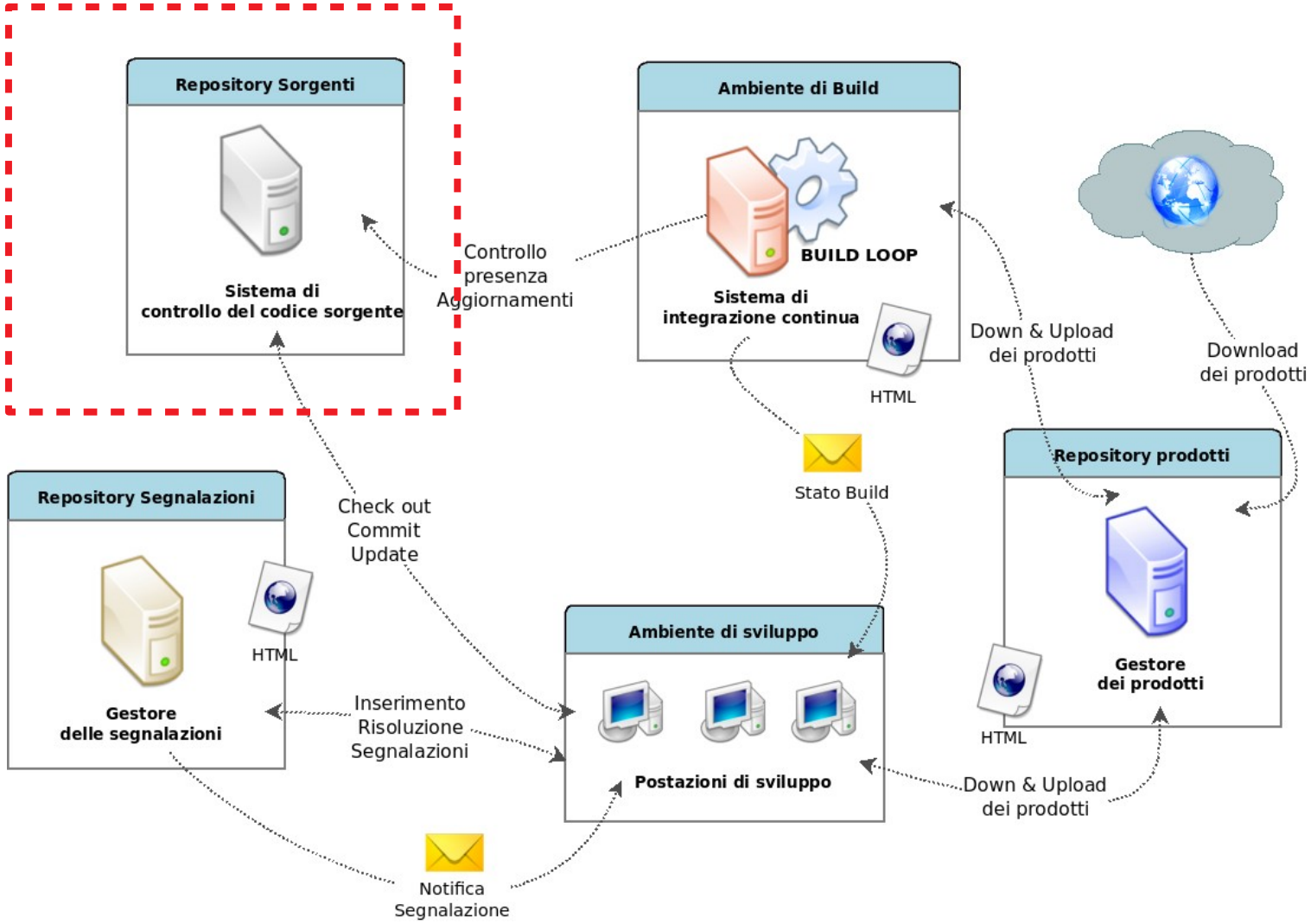
METODI E TECNOLOGIE PER LO SVILUPPO SOFTWARE

Nicola Bertazzo
nicola.bertazzo [at] unipd.it

Università degli Studi di Padova
Dipartimento di Matematica
Corso di Laurea in Informatica, A.A. 2020 – 2021

SOURCE CODE MANAGEMENT (SCM)

VISIONE GENERALE



Version Control Sytem (VCS)

DEFINIZIONE

“A component of **software configuration management**, **version control**, also known as revision control or source control, is the **management of changes** to documents, computer programs, large web sites, and other collections of information.

Changes are usually **identified by a number or letter code**, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. **Each revision** is associated with a **timestamp** and the **person making the change**. Revisions can be compared, restored, and with some types of files, merged.”

WIKIPEDIA
The Free Encyclopedia

COSA SONO GLI SCM?

Source Code Management systems (SCM), conosciuti anche come "Version Control Systems" (VCS):

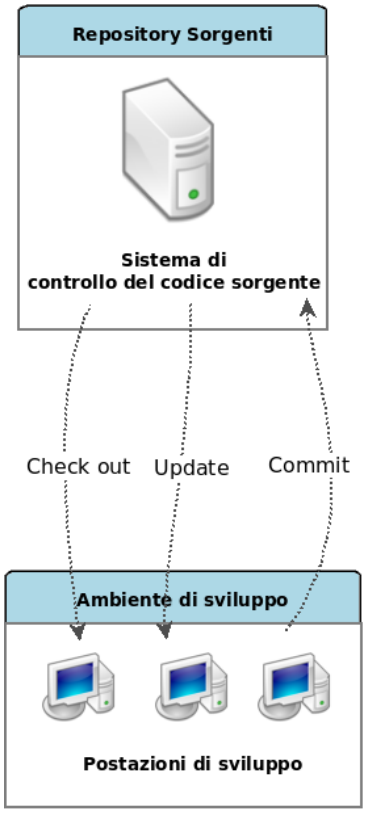
- Sono sistemi software
- Registrano tutte le modifiche avvenute ad un insieme di file
- Permettono la condivisione di file e modifiche
- Offrono funzionalità di merging e tracciamento delle modifiche

PERCHÉ USARE UN VCS?

In un Team di sviluppo, un VCS permette di:

- **Collaborare** in modo efficiente nel codice di un prodotto
  - Facilita **l'individuazione** e la risoluzione di **conflitti**
  - Facilita la **condivisione** di commenti e documentazione

- Tracciare ogni modifica (storia del prodotto):
  - Fornire una **storia completa** delle modifiche avvenute nel prodotto
  - Facilita il **ripristino** dei file ad una **versione precedente**



BENEFICI

Mantengono la **storia completa** di ogni cambiamento avvenuto per ogni file (autore, data, motivazione)

Lavorare senza interferenze in differenti rami di sviluppo (Branching). Le modifiche effettuate in un ramo di sviluppo possono poi confluire nel ramo principale

Tranciabilità: tutte le modifiche possono far riferimento alle attività registrate nel Issue Tracking System. In ogni istante è possibile capire che attività sono state effettuate in una specifica versione

DIFFERENTI TIPI DI VCS

Gli VCS possono essere classificati in tre tipologie:

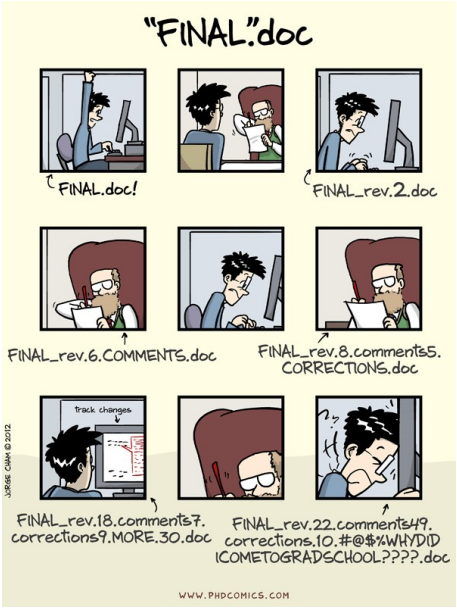
- Locali
- Centralizzati
- Distribuiti

CVS

DIFFERENTI TIPI DI VCS

LOCAL VCS

“Molte persone gestiscono le diverse versioni copiando i file in un'altra directory (magari una directory denominata con la data, se sono furbi). Questo approccio è molto comune perché è molto semplice, ma è anche incredibilmente soggetto ad errori. È facile dimenticare in quale directory sei e modificare il file sbagliato o copiare dei file che non intendevi copiare.”



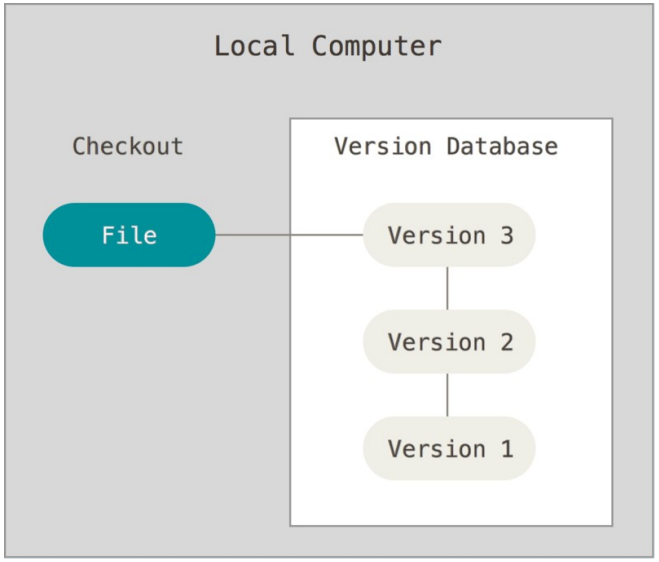
http://phdcomics.com/comics/archive.php?comicid=1531

https://git-scm.com/book/it/v1/Per-Iniziare-Il-Controllo-di-Versione

DIFFERENTI TIPI DI VCS

LOCAL VCS

- **I più vecchi**
- Registrano **solo** la storia dei cambiamenti:
  - Utilizzano un "Version Database" dove viene registrata la storia di tutti i file
  - Salva sul disco una serie di *patch* (ovvero le differenze tra i file) tra una versione e l'altra
  - È possibile ricreare lo stato di qualsiasi file in qualsiasi momento
- **Non gestiscono la condivisione**

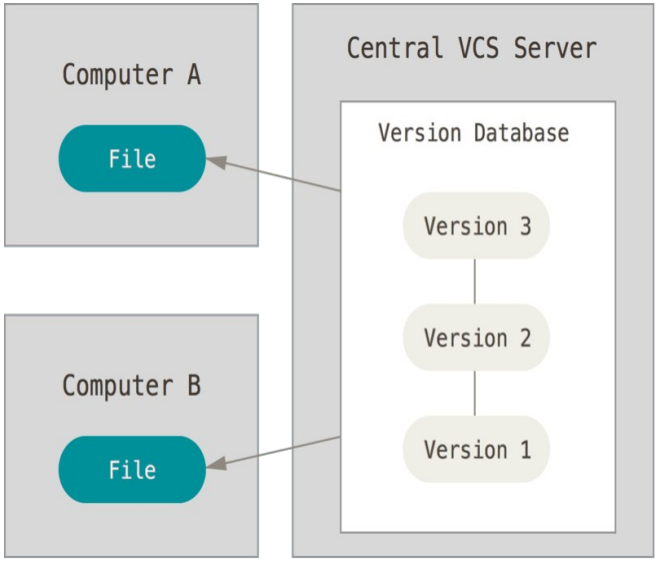


P.es: RCS, IDE (Eclipse, IntelliJ..)

DIFFERENTI TIPI DI VCS

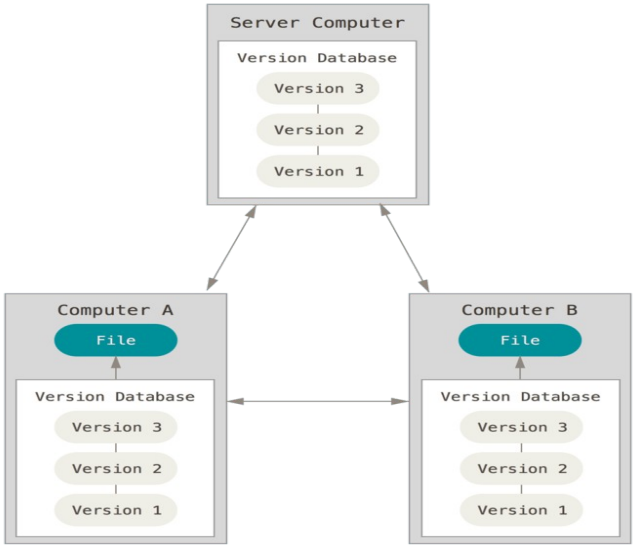
CENTRALIZED SCM (CVCS)

- Meno vecchi e **molto diffusi**
- Gestiscono sia la **condivisione** che il tracciamento delle storia:
  - La storia è gestita come nel **LOCAL VCS**
  - Il "version database" è gestito in un **server centrale (singolo punto di rottura)**
- Ogni sviluppatore è un client che ha nel suo spazio di lavoro **solo una vestione** (alla volta) del codice
- Facili da apprendere
- P.es: CVS, **SVN**, Perforce, TFS

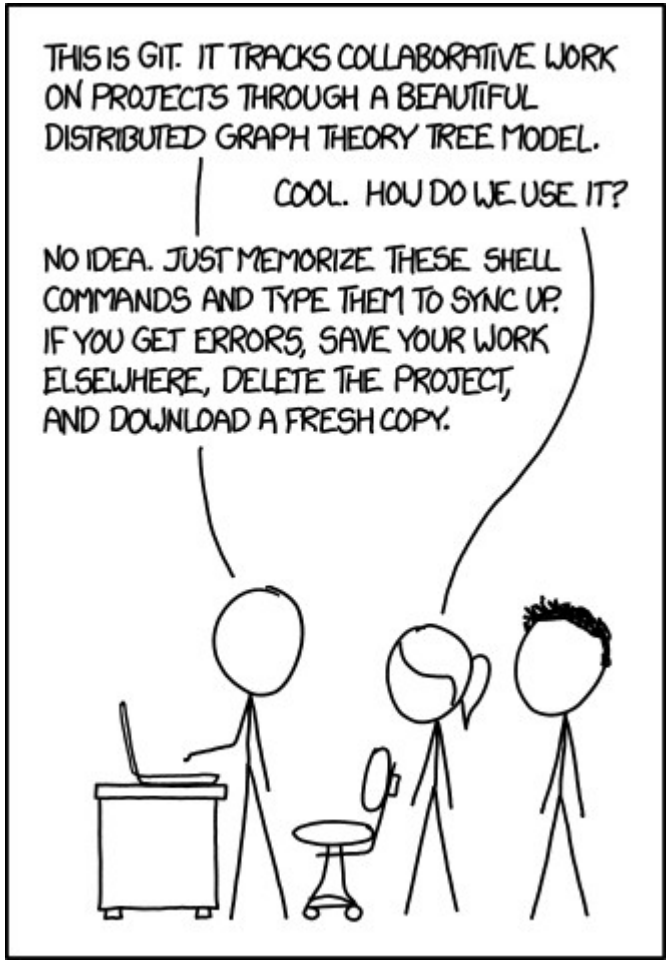


DISTRIBUTED VCS (DVCS)

- Simili ai CVCS ma il "Version Database" **è distribuito per duplicazione in ogni nodo**
  - Quando il **nodo centrale non è disponibile**, **è possibile continuare a lavorare** e registrare i cambiamenti
  - Hanno una **migliore risoluzione dei conflitti** che favorisce la collaborazione
  - permette di impostare **diversi tipi di flussi di lavoro** che non sono possibili in sistemi centralizzati
- **L'apprendimento** è più **complesso** rispetto ai CVCS
- P.es: Git, Mercurial, Bazaar o Darcs



DISTRIBUTED VCS (DVCS)



https://xkcd.com/1597/

CLOUD-BASED SCM

Esistono “VCS as a Service”

Il "Version Database" è gestito in un servizio Cloud

+ **Si delega la gestione** (e l’installazione) ad un servizio esterno

- Il codice sorgente non è nell’infrastruttura aziendale

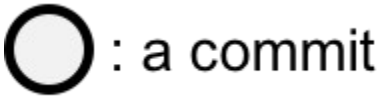
+ Forniscono altri servizi oltre al VCS come text editor online, strumenti visuali, issue tracking system, etc…

Esempi: GitHub, Bitbucket by Atlassian, Amazon CodeCommit, Visual Studio Online by Microsoft, SourceForge, GitLab, etc.

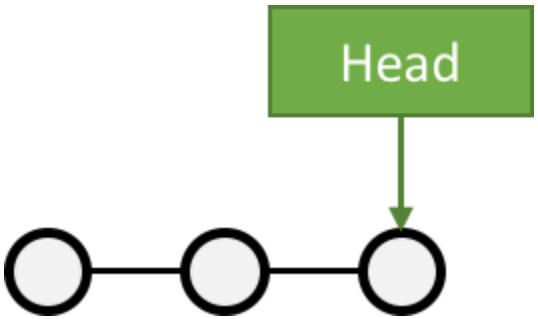
TERMINOLOGY

BASICS

- Each set of changed lines on a single file is a "diff"
- A set of diffs which have been explicitly validated are a "commit"

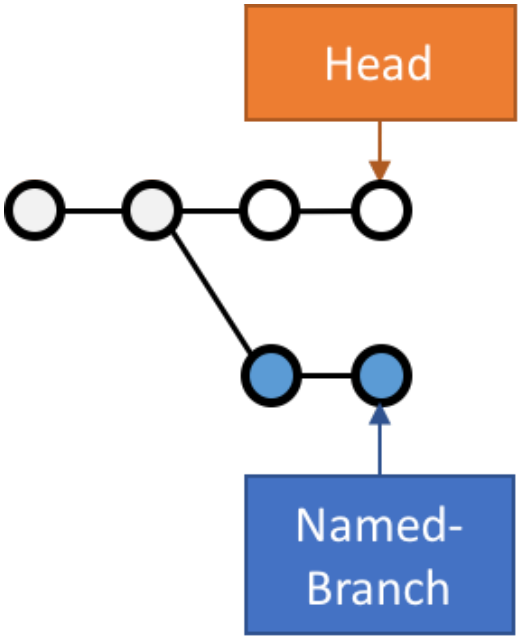


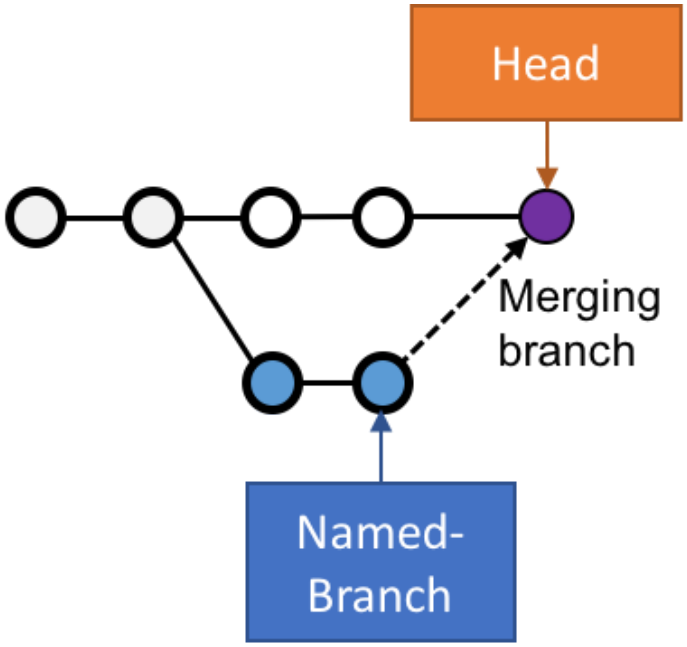
- A commit is in fact a new version of the codebase
- A commit can exist locally and remotely
- The latest commit on the history is the "HEAD"



BRANCHES

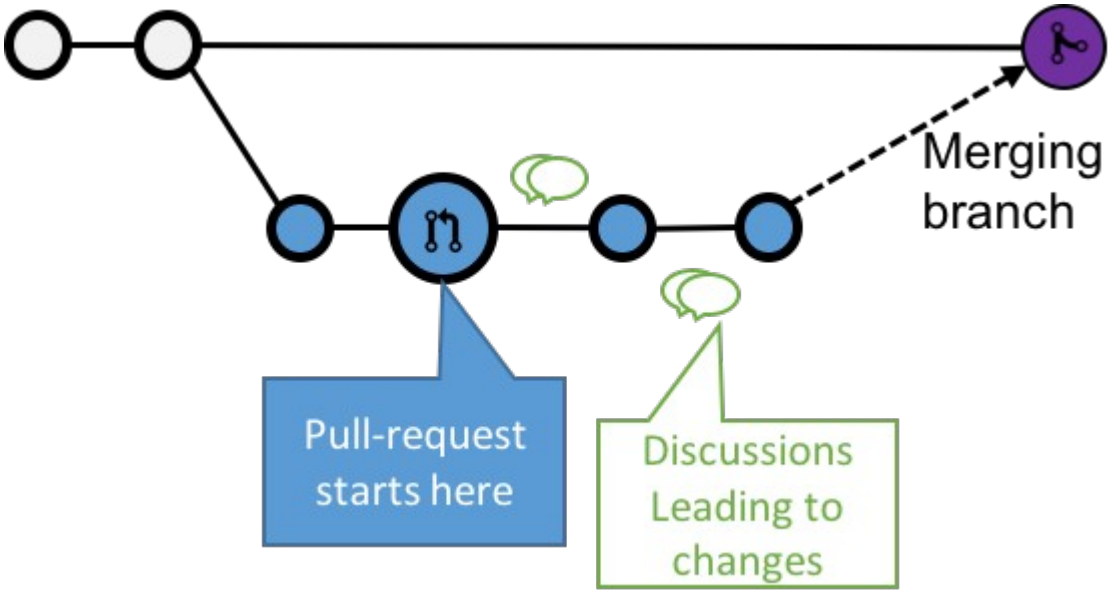
- A branch, in SCM terminology, is a pointer to a single commit
  - HEAD is the "latest" branch, also known as the "master" branch
  - To integrate a branch, you have to merge it:





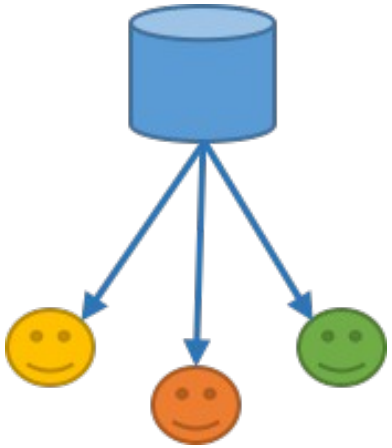
PULL REQUEST

- A Pull-Request is a way of handling branch merges to "master"
  - A branch is pushed to the central server without having been merged
  - It "asks" to be merged on a central repository
  - Opportunity is given to review the changes before merging
  - Pull request ends by being closed or merged to destination branch



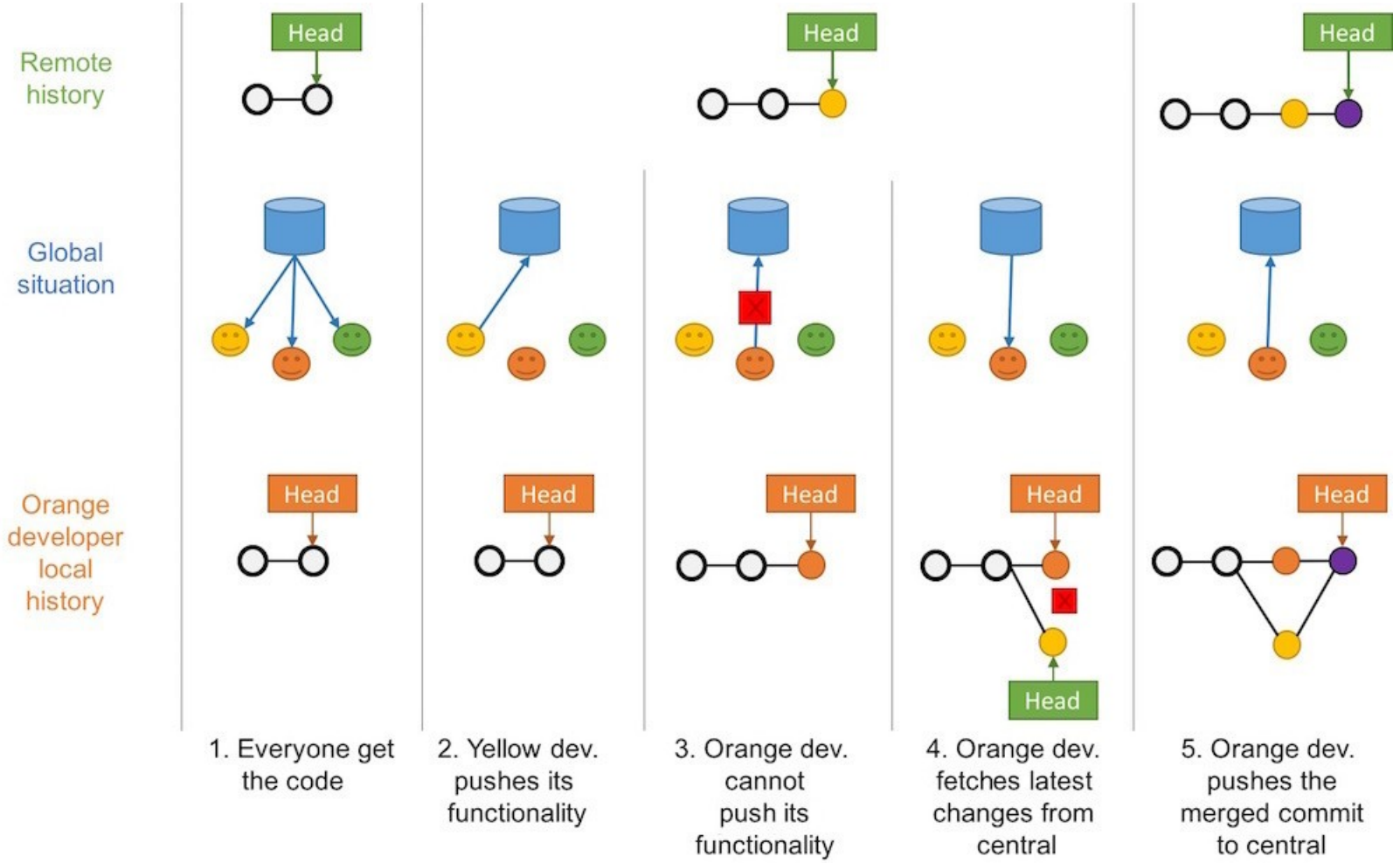
CENTRALIZED

- This pattern is the natural usage of a CVCS like SVN or CVS
- It is easy to understand and use, and sufficient enough for a lot of cases
- Collaboration is blocked when centralized server is down or history is broken



WORKFLOW PATTERNS

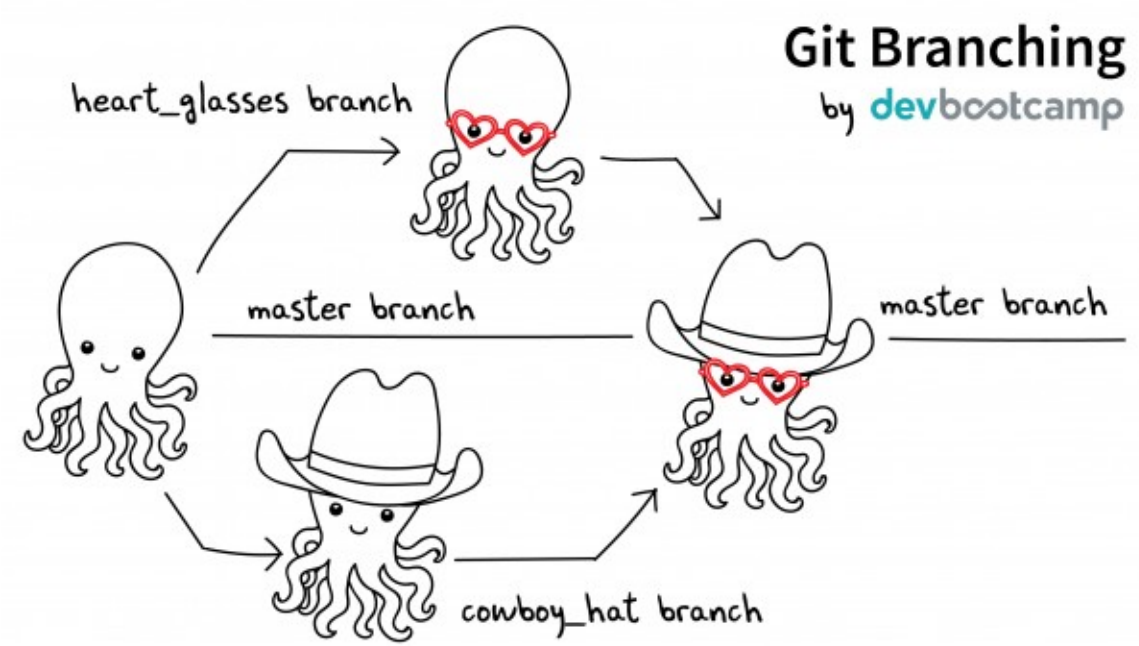
CENTRALIZED



FEATURE BRANCH

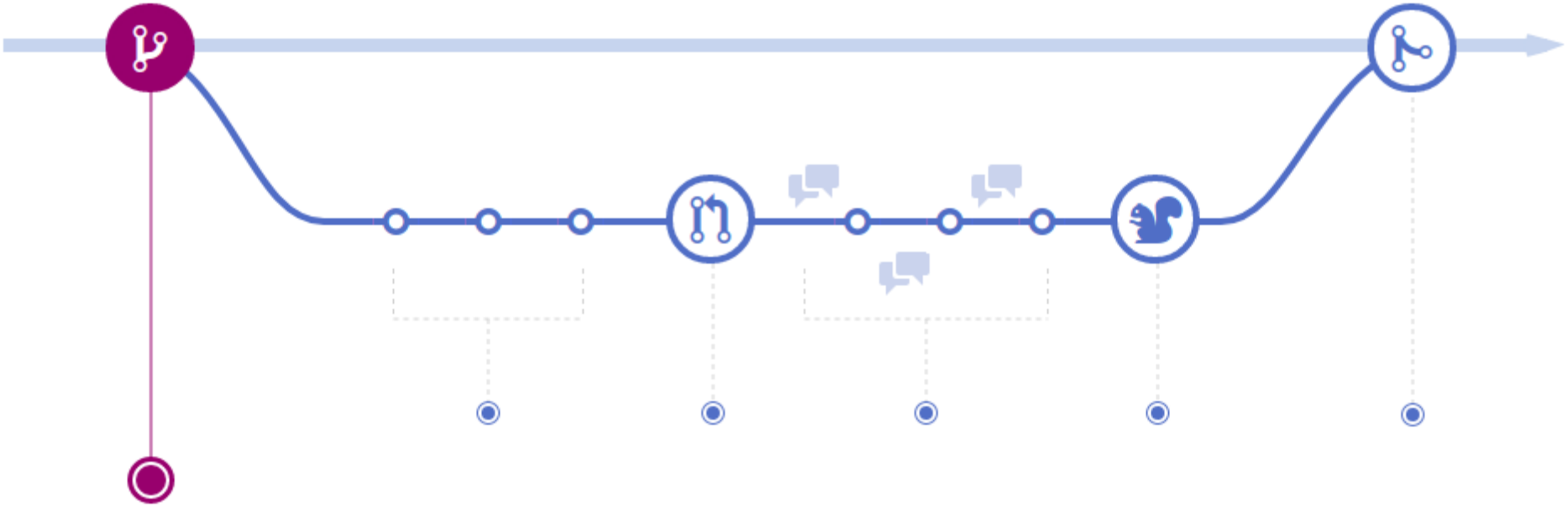
Goal of this pattern: using one branch per feature

- Encapsulation allows working without disturbing the main codebase
- Allows easier collaboration
- Merge conflicts maps the conceptual conflicts: easier to track



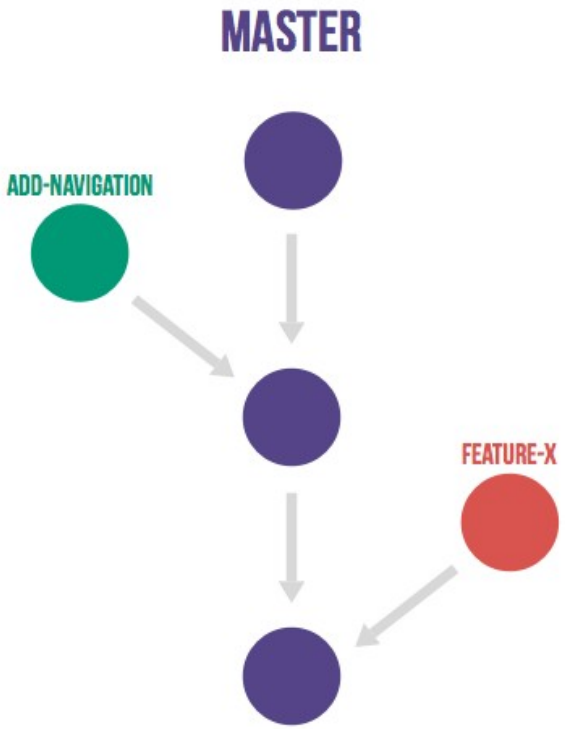
https://twitter.com/jay_gee/status/704016330425372672?s=03

GITHUB FLOW



https://guides.github.com/introduction/flow/

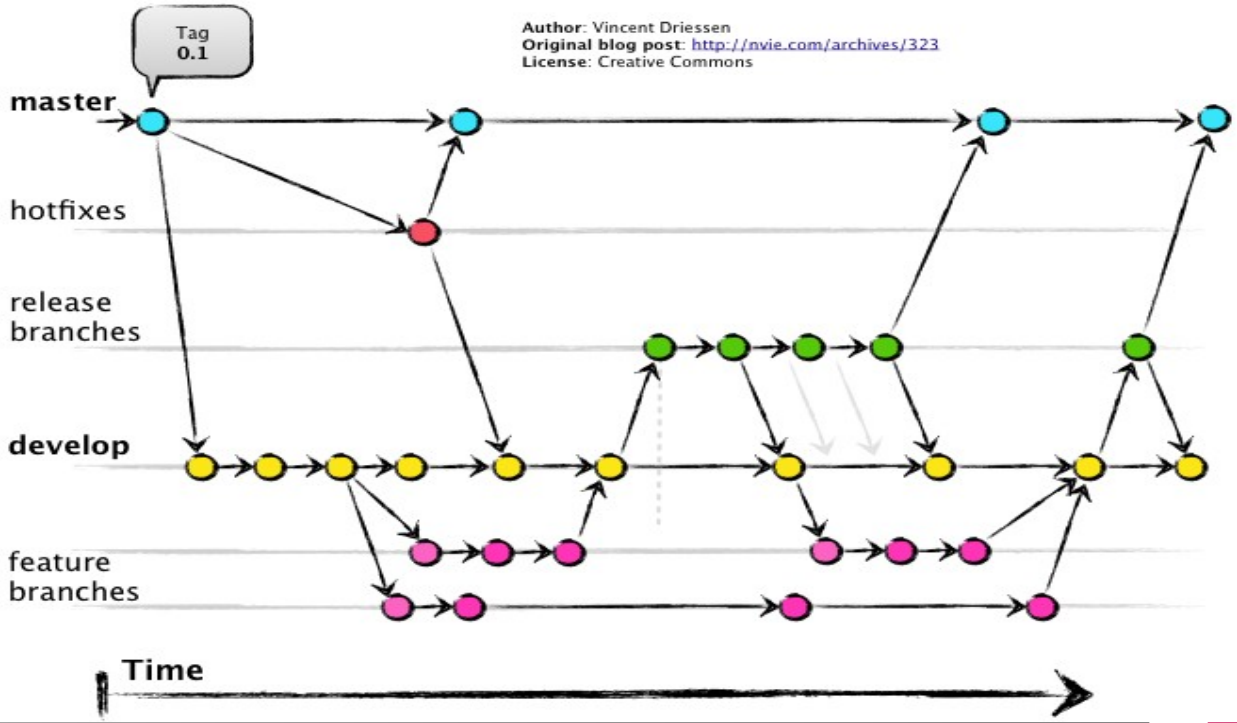
GITLAB FLOW



https://docs.gitlab.com/ee/topics/gitlab_flow.html
https://about.gitlab.com/topics/version-control/what-is-gitlab-flow/
https://about.gitlab.com/topics/version-control/what-are-gitlab-flow-best-practices/

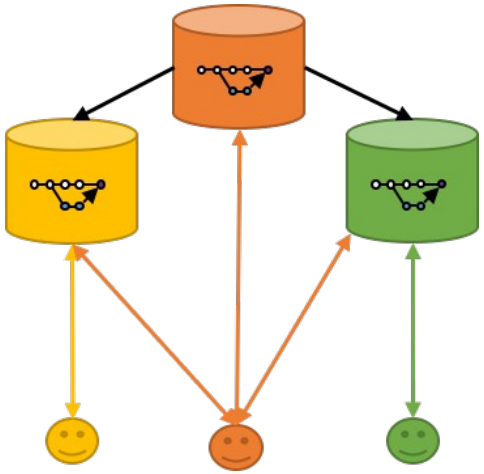
GITFLOW

- Extension of the Feature Branch Workflow pattern
- Proposed by Vincent Driessen at nvie
- Target large-scale projects or software products
- Strict branching implementation: roles for specific branches



FORK

- Pattern inherited from (GitHub/Bitbucket)-like services
- Push forward concepts of distributed file systems
- Each user "forks" the main repository and can propose pull requests between repositories
  - Authorization management improved
  - Autonomy for better collaboration process
  - Decentralized for new patterns ("promiscuous integration")



CVCS VS DVCS

CVCS

+ L'apprendimento è più semplice
- Meno recenti
- *Single point of failure.* Se si corrompe il server centrale si perde tutta la storia
+ Permettono il lock dei file
- Favoriscono l'utilizzo del centralized work flow. Non è possibile adottare tutti i work flow precedentemente descritti
- Le operazioni di commit sono più lente. Non possono essere fatte off-line

DVCS

- L'apprendimento è più difficile
+ Più recenti *Standard de facto*
+ Architettura distribuita per replica. Meno possibilità di perdere tutta la storia
- Non permettono il lock dei file
+ È possibile adottare più tipi di work flow
+ Le operazioni di commit sono più veloci perché avvengono in locale e possono avvenire anche off-line

COSA ABBIAMO IMPARATO

- Il codice sorgente di un progetto deve essere versionato in un VCS, che è un sistema software che tiene traccia di tutti i cambiamenti e facilita la condivisione
- Esistono 2 tipi di VCS: Centralized and Distributed VCS
- In base alle esigenze e alla conoscenza del team, si può scegliere il workflow pattern più adatto tra: Centralized, Feature Branch, GitFlow, Fork etc..
- Per la scelta del Servizio da utilizzare (on premisis o cloud) bisogna tenere in considerazione (almeno) i seguenti fattori:
  - Backup
  - Crash
  - Privacy
  - Esigenze del progetto

RIFERIMENTI & APPROFONDIMENTI

- https://en.wikipedia.org/wiki/Version_control
- https://it.atlassian.com/git/tutorials/what-is-version-control
- https://git-scm.com/book/it/v1/Per-Iniziare-Il-Controllo-di-Versione
- https://martinfowler.com/bliki/VersionControlTools.html
- https://www.atlassian.com/git/tutorials/comparing-workflows
- https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
- http://martinfowler.com/bliki/FeatureBranch.html
- http://nvie.com/posts/a-successful-git-branching-model/
- https://www.simple-talk.com/opinion/opinion-pieces/branching-and-merging-ten-pretty-good-practices/
- http://rogerdudler.github.io/git-guide/
- https://docs.gitlab.com/ee/topics/gitlab_flow.html
- https://about.gitlab.com/blog/2016/07/27/the-11-rules-of-gitlab-flow/