Sistemi per il controllo versione del software (VCS)

Indice

- Concetti alla base del controllo versione
  - Versione e Configurazione
  - Memorizzazione delle versioni
  - Baseline e Release
- Alcune pratiche consigliate

Esigenze di un team di sviluppo

- Presenza di un repository (centralizzato) e condiviso tra i membri del team di sviluppo
- Gestione delle modifiche effettuate contemporaneamente a uno stesso file da parte di due membri differenti
- Gestione delle diverse versioni di un prodotto che può evolvere
- Ripristino di vecchie versioni di prodotti
- Individuazione di versioni del prodotto in cui sono stati risolti errori o introdotte nuove funzionalità

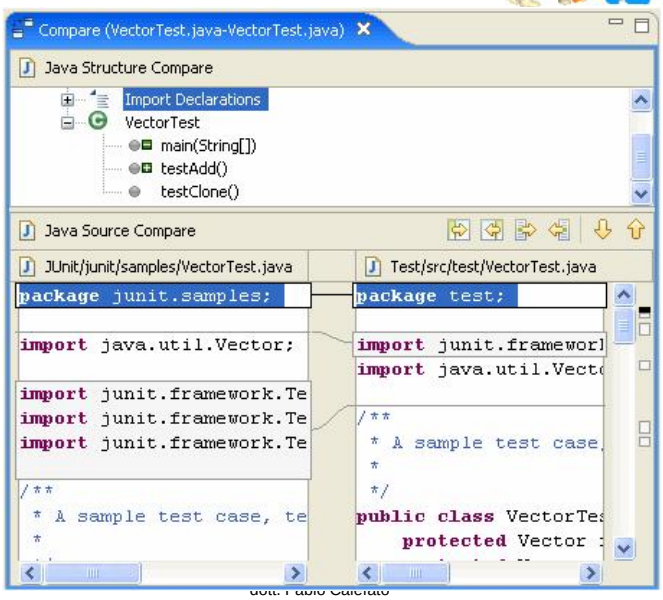
Gestione delle versioni e delle configurazioni
di prodotti e di semilavorati

diff e patch

- Utilizzo combinato dei tool Unix diff e patch
- diff permette il confronto di due file simili (F1 e F2)
  - restituisce in output una lista (d) con le differenze riscontrate
    - diff: F2 − F1 = d
- patch modifica il file originale (F1) inserendo le modifiche
  - crea una versione (F2) cosiddetta “patched”
    - patch: F2 = F1 + d
- Nessuna capacità di tenere traccia della storia di un progetto

```
--- /Users/daniel/Sites/dev/Github/SB2/README.md
+++ (clipboard)
@@ -1,4 +1,17 @@
+Sublime-Text-2
+==============

- CSS
- XML
- JavaScript (not well done)+Currently I just released my first Color Schem
+Please notice, that is my very first work.
+Feedback is always welcome.
+
+Currently tested / supported Languages are:
+
+- HTML
+- JSON (not well done)
+- PHP
+- Markdown
+
+# Screens
+![Example 1](https://raw.github.com/pogosheep/Sublime-Text-2/master/Example
+![Example 2](https://raw.github.com/pogosheep/Sublime-Text-2/master/Example
```

Versione e Configurazione

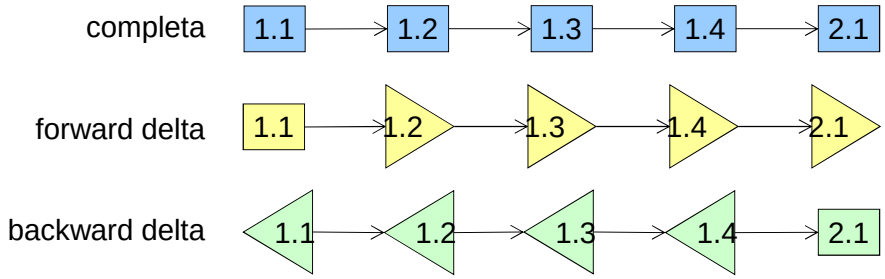
- Versione: stato di un elemento in un istante di tempo
- Configurazione: insieme di moduli software, documentazione, ecc., utilizzati per costruire un prodotto
  - Due configurazioni differiscono se contengono due versioni diverse dello stesso modulo (contenuto diverso)

Memorizzazione delle versioni (1/3)

- Completa: memorizzare ogni versione di un modulo in un file separato
  - Pro: facilità ed efficienza nel reperimento di una specifica versione
  - Con: troppo spazio di memorizzazione richiesto
- Delta: memorizzare in forma completa solo una versione e di tutte le altre versioni si conservano solo le differenze
  - *forward delta*: prima versione e differenze rispetto alle successive
  - *backward delta*: versione più recente e differenze rispetto alle precedenti
  - Pro: ottimizzazione dello spazio di memorizzazione richiesto
  - Con: meno efficiente nel reperimento di una specifica versione

Memorizzazione delle versioni (2/3)

- Ogni versione è identificata da un numero

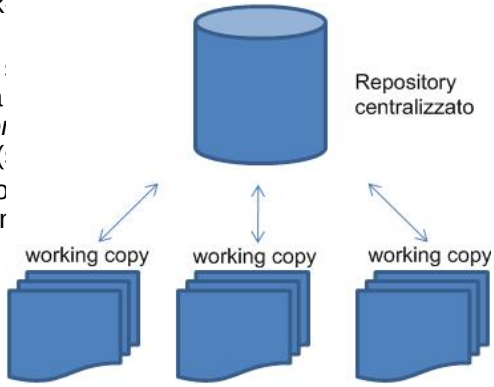


Baseline e Release

- Baseline: insieme di elementi versionati "fotografati" in un determinato istante di tempo
  - è stata sottoposta a una revisione formale ed ha ricevuto l'approvazione da parte del manager responsabile
  - è utilizzata come base per lo sviluppo successivo
- Release: "promozione" di una baseline
  - acquisisce visibilità all'esterno del gruppo/organizzazione responsabile

Modello copy-modify-merge

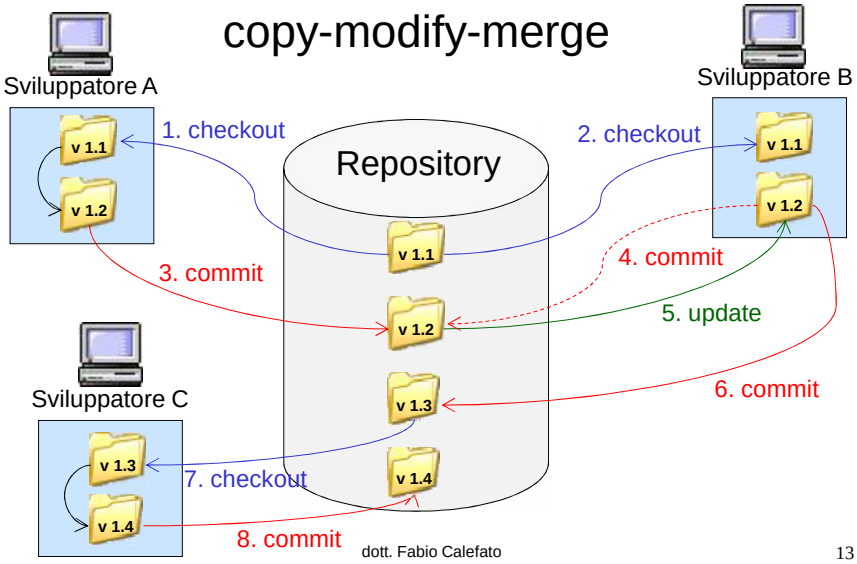
- Supera limiti del modello lock modify-unlock
- **copy**: ogni sviluppatore crea propria macchina (client) una *working copy* (o *copia di lavo* "richiedendola" al repository (
- **modify**: gli sviluppatori posso lavorare in parallelo modificar la propria copia di lavoro
- **merge**: le copie di lavoro vengono fuse quando vengono rispedite al repository



Comandi (1/4)

- [Checkout] Ottenere un nuovo modulo
  - Il client ricrea la struttura di directory del repository sulla macchina locale
- [Commit] Inviare le modifiche al server
  - Prima di qualsiasi commit eseguire sempre un update per ottenere le eventuali altre modifiche
- [Update] Ottenere le nuove modifiche
  - Le modifiche vengono "unite" ai file locali in automatico a meno che non ci siano conflitti
  - I conflitti sono risolti manualmente mediante l'ausilio del tool diff

Funzionamento del modello copy-modify-merge



Comandi (2/4)

- [Add] Aggiungere file e/o directory
  - I file vengono “marcati” per l’aggiunta
  - Per rendere definitive le modifiche è necessario eseguire un commit
- [Remove] Rimuovere file e/o directory
  - Per rendere definitive le modifiche è necessario eseguire un commit

Comandi (3/4)

- [Tag] Mettere un'etichetta ai file
  - Prima di rilasciare una release è necessario tenere traccia dello stato dei file
  - Marcare lo stato dei file dell'attuale configurazione con un nome simbolico
    - Usare un nome che contenga una indicazione del tipo di uso che si farà della release e un codice numerico
    - Esempio: se è una release da usare in produzione ed è la prima che produciamo, si può usare Prod1_0
- [Release] Creare una release
  - Dopo aver etichettato i file, si può creare il pacchetto relativo al modulo da rilasciare

Comandi (4/4)

- [Branch] Ottenere diverse linee di sviluppo
  - Etichettare nuovamente la release aprendo in contemporanea una linea di sviluppo alternativa
    - in gergo viene chiamata branch
  - Eseguire il checkout del branch appena aperto
  - Lavorare sulla copia locale della versione
- [Merge] Effettuare il merge da un branch
  - Se le modifiche in un branch raggiungono una fase stabile, è possibile riportarle nella linea principale di sviluppo
  - Per rendere definitive le modifiche è necessario eseguire un commit

CVS

- Concurrent Versions System
  - http://www.cvshome.org/
- Utilizzo del modello copy-modify-merge
- Sviluppato da Dick Grune nel 1986
- Rilasciato sotto GNU General Public License sul newsgroup mod.sources il 23 giugno 1986
- È una collezione di script
  - Riscritti in linguaggio C da Brian Berliner nel 1989
- Risolve i problemi lasciati aperti da RCS
  - Gestione di un progetto come un'unica entità
  - Networking (Jim Kingdon, 1990) mediante un'architettura client-server

SVN

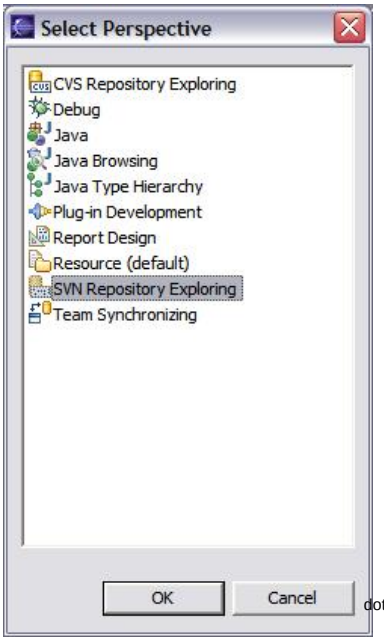
- Subversion
  - http://subversion.tigris.org/
- Utilizzo del modello copy-modify-merge
- Nasce come progetto open source su Tigris.org
  - La versione 1.0 è stata rilasciata il 23 Febbraio 2004 sotto licenza Apache/BSD-style
- Comprende gran parte delle caratteristiche di CVS e le estende

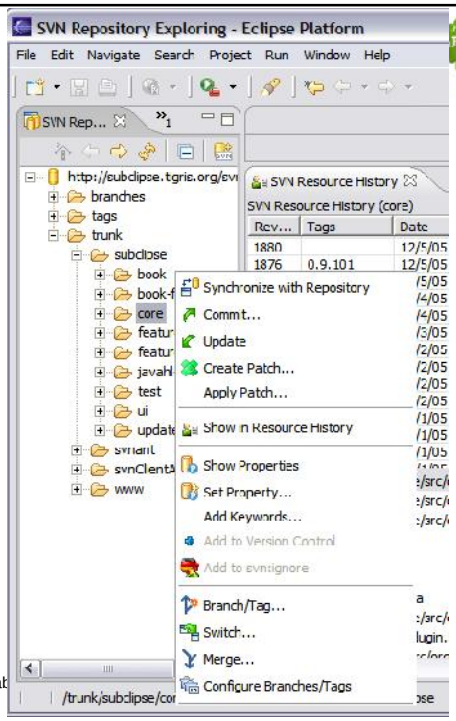
Novità rispetto a CVS

- Il controllo di versione avviene anche sulle directory
- Un file che era stato precedentemente rimosso può essere aggiunto nuovamente
- I file binari sono gestiti efficientemente
- Il protocollo client/server invia solo le differenze in entrambe le direzioni
- I commit sono atomici
- Diverse modalità di accesso al repository

Client Subversion

- TortoiseSVN
  - Estensione GUI per windows
  - http://tortoisesvn.tigris.org/
- Subclipse
  - Plugin per Eclipse
  - http://subclipse.tigris.org/
- **Subversive ←**
  - Plugin per Eclipse
  - http://www.eclipse.org/subversive/





dott. Fab

Alcune pratiche consigliate

- Eseguire [Update] di frequente e integrare spesso le modifiche sul server
- Eseguire il [Commit] solo di codice perfettamente compilabile
  - Testare a fondo prima del commit
- I [Merge] per eventuali conflitti emersi durante un [Commit] devono essere tempestivi

Centralized vs. Distributed VCS

- Sia CVS che SVN sono sistemi centralizzati
  - Modello client-server
  - Server mantiene storia completa del progetto (codice, metadati)
  - Client mantiene porzione limitata del progetto
- Sistemi decentralizzati (metà ~2000)
  - Modello P2P
  - Nessun server, ogni repository clone completo
  - Commit resta locale, bisogna renderlo pubblico

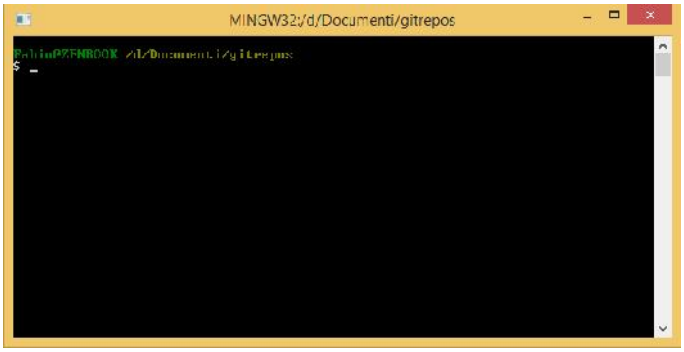
Git

- Inizialmente sviluppato da Linus Torvalds per il progetto Linux
  - 2002: da CVS a BitKeeper (BitMover - proprietario)
- DVCS timeline
  - GNU arch (2001), Monotone (2003) [1st gen.]
  - Darcs (2002), Git (2005), Mercurial (2005), Bazaar (2005) [2nd gen.]

DEMO GIT

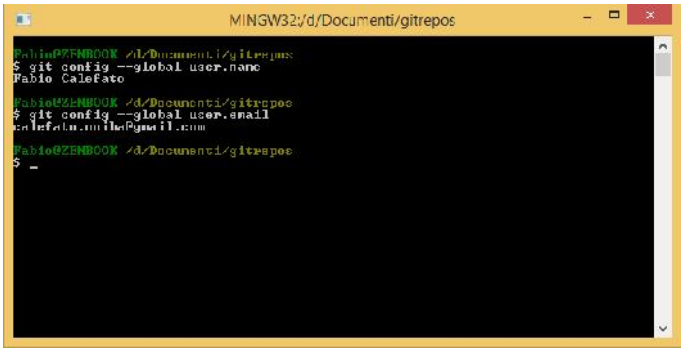
Installazione

- git (riga di comando)
  - http://git-scm.com/downloads
  - Windows, Linux, Mac



Configurare git (1/2)

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



Configurare git (2/2)

- Stabilisce il comportamento di default in caso di push senza fornire un refspec
- $git config –global push.default …
  - nothing (*do not push anything unless a refspec is explicitly given*)
  - matching (*push all branches having the same name on both ends*)
  - upstream (*push the current branch back to the one it was cloned from*)
  - current (*push the current branch to a branch of the same name, even if not its upstream*)
  - **simple*** (*like upstream in centralized workflows or current in decentralized ones, refuses to push if the upstream branch's name is different from the local one*)

** default a partire da Git 2.0*

Creare un progetto in git

- Modalità 1: versionare artefatti esistenti in git
  - $ cd YOUR_PROJECT_FOLDER
  - $ git init

    "*Initialized empty Git repository in .git/*"
  - $ git add -A
  - $ git commit –a –m "*primo commit*"

Aggiunge e committa tutti i cambiamenti

Conserva la versione nel repository locale

Inizia a tracciare i cambiamenti di tutti i file e cartelle nella directory corrente (snapshot)

Creare un progetto in git

- Modalità 2: clonare un repository git esistente
  - $ git clone URL [YOUR_PROJECT_FOLDER]
- Il repository locale è il *clone*, quello originale all'indirizzo URL è l'*origin*
- Esempio
  - $ git clone https://github.com/collab-uniba/wp-calendapp-plugin.git
- *Fork* è l'azione di creare un clone di un repository origin

Recuperare modifiche da origin

- Per aggiornare la copia clone locale con le ultime modifiche
  - $ git pull

  Scarica le modifiche dal repository remoto origin e le integra in quello clone locale

- Nota
  - $git status

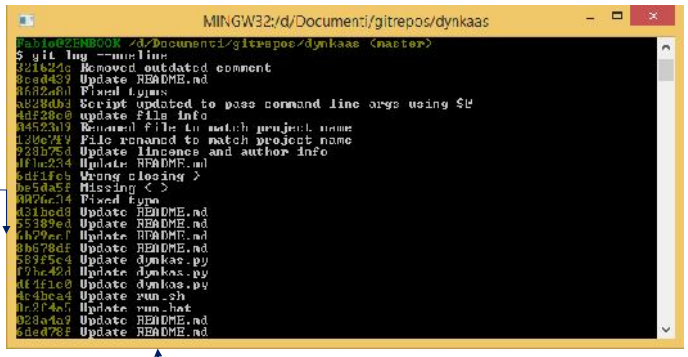
  Fornisce info sulle modifiche non committate nel repository locale

  - $ git fetch

  Fornisce info sulle modifiche pendenti nel repository remoto non integrate nel clone locale

Verificare la storia delle modifiche

- $ git log –oneline



Ritornare a una vecchia versione

- $ git checkout *revisionID*
- *Esempio: tornare al primo commit*
  - *$ git checkout 8a04d37*

Ritornare all'ultima versione del progetto

- $ git checkout master

Tag di una versione

- Annotare una versione del repository con un nome simbolico
  - Es. *1.0.0* più facile da ricordare del commit ID *36c3a0c*
- $ git tag 0.0.1 [–m "*Release iniziale 0.0.1*"]
- $ git push origin tag *0.0.1*
- $ git tag
- $ git checkout [*branch name*|*tag name*|*revisionID*]
- $ git tag –d 0.0.1
- $ git push origin :refs/tags/0.0.1

Applica tag 0.0.1 in locale

Salva tag 0.0.1 sull'origin remoto

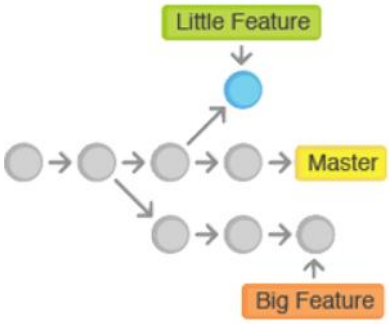
Elenca tag esistenti

Cancella tag locale

Cancella tag da repo remoto origin

Branch

- Un *branch* (ramo) è una linea di sviluppo
- La linea principale si chiama *master*
- Per ogni nuova caratteristica da aggiungere al progetto
  - Si crea un nuovo branch X dedicato
  - Si esegue lo sviluppo su X
  - Si effettua il merge (fusione) dei cambiamenti di X in master

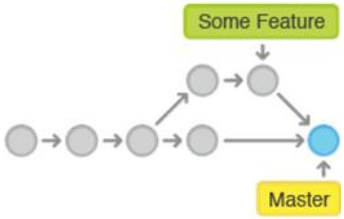


Operazioni su branch…

- $ git branch — *Elenca branch esistenti ed evidenzia quello corrente*
- $ git branch *nome-feature* — *Crea nuovo branch a partire da quello corrente*
- $ git checkout *nome-feature* — *Passa nome-feature come branch corrente*
- $ git push origin *nome-feature* — *Ricrea il branch nomefeature sul repo remoto d'origine*
- $ git checkout master — *Ritorna a master come branch corrente*

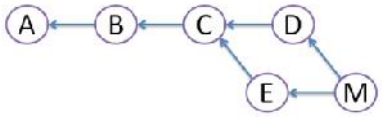
…Operazioni su branch

- $ git merge *nome-feature* — *Porta le modifiche di nomefeature nel branch corrente*
- $ git branch –d *nome-feature* — *Rimuove branch (verifica che sia stato fatto un merge prima)*
- $ git branch –D *nome-feature* — *Forza rimozione branch (anche senza merge)*



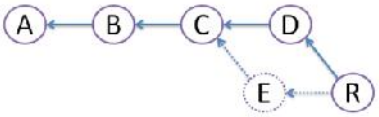
Merge o rebase?

Merge (three-way merge)



- Le revisioni D ed E in conflitto fuse con C rimangono memorizzate nella history

Rebase



- La revisione D fusa con C, poi E è fusa con D e cancellata dalla history

Il risultato delle fusion è identico (rev M == R)

Che guaio…

- *… ho copiato dei file che non voglio versionare in una cartella git!*
  - $ git clean –f –d

  Rimuove tutti i file e cartelle non versionate all'interno di un repository git
- *… voglio scartare tutte le modifiche fatte nella cartella di progetto!*
  - $ git reset --hard HEAD
- *… ho dimenticato di aggiungere dei file all'ultimo commit!*
  - $ git add 1.py 2.py
  - $ git commit --amend [--no-edit | - m "nuovo messaggio"]
- *… voglio cancellare gli ultimi N push sull'origin!*
  - git revert --hard HEAD~N
- *… ma chi đ#**Ă è stato?!?*
  - $ git blame fileincriminato.py

Restituire modifiche dal clone all'origin

- Premessa: modifiche devono essere state aggiunte al repo locale tramite comando commit
- Dopodiché
  - $git push
- E' necessario avere i permessi di scrittura sul repo origin remoto (non è sempre possibile)
- In tal caso occorre chiedere permesso attraverso una «pull request» (anche dette «merge request»)
  - Si inviano modifiche al proprietario dell'origin che decide se e quando incorporarle
  - Pull request vanno gestite via web sulla pagina di progetto

Memorizzare comandi git

- Scaricare e tenere a portata un *git cheat sheet*
- Esempio:
- http://www.git-tower.com/blog/git-cheat-sheet-detail

VCS BRANCHING

Branch & project workflow

- Workflow dei progetti moderni supportato attraverso diversi branch
  - Più grande e complesso un progetto, maggiore il numero
- Branch si distinguono per
  - Durata della “vita”
  - Stabilità del codice
- Direttamente proporzionali
  - Più a lungo vive un branch, maggiore la stabilità

Common branching workflow in git

Main branch

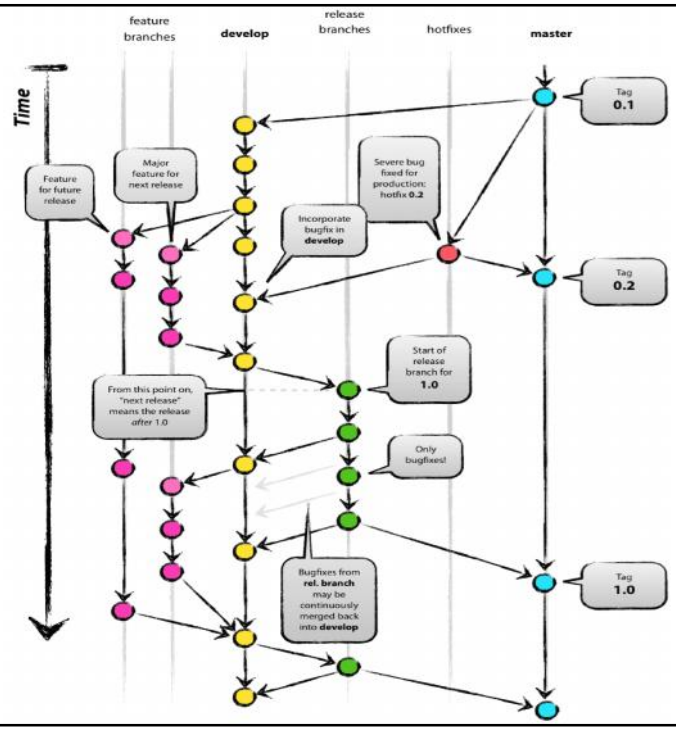
aka long-running

- master
  - codice stabile, pronto per andare in produzione
  - build stabili
- develop
  - cambiamenti al codice, pronti per essere integrati nella prossima release
  - build nightly

Topic branch

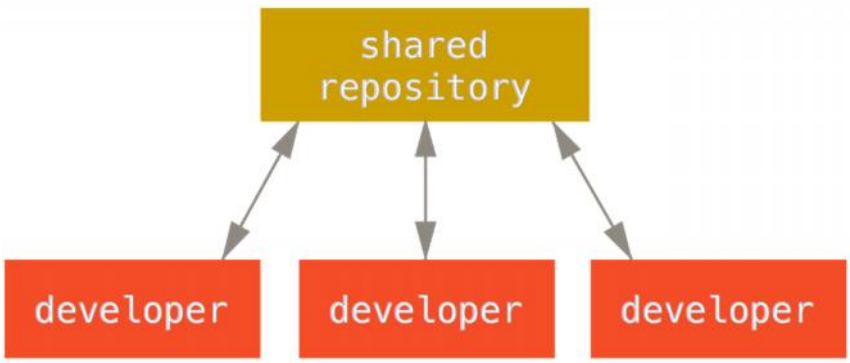
aka short-lived, supporting

- feature
  - aggiunta di nuove funzionalità
  - e.g., *filedownload*, *logging*
- issue
  - risoluzione di un issue specifico
  - e.g., issue3, iss91a, iss91v2
- hotfix
  - risoluzione di difetti gravi di codice in produzione



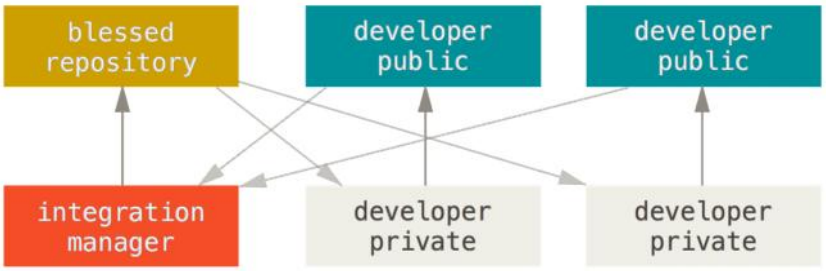
WORKFLOW DECENTRALIZZATI CON DVCS

Centralized workflow



Possibile con VCS centralizzati e decentralizzati

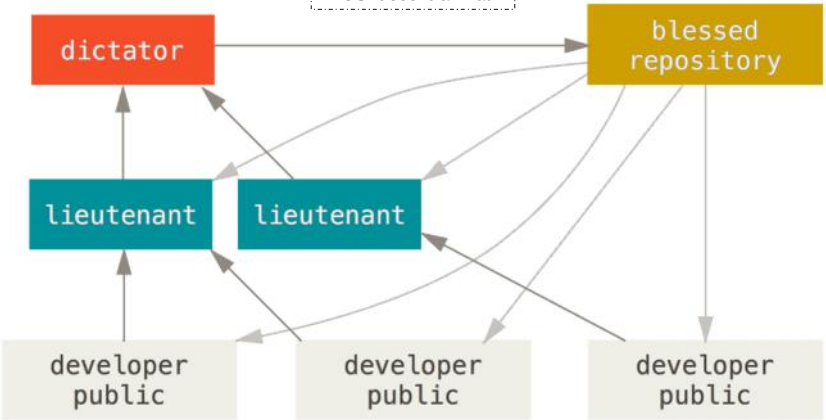
Integration-Manager workflow



Possibile solo con VCS decentralizzati

Dictator and Lieutenants Workflow

Possibile solo con VCS decentralizzati



Riferimenti

- Scott Chacon, Pro Git, Apress (2009), http://git-scm.com/book
- Atlassian git Tutorials, https://www.atlassian.com/git/tutorial/git-basics
- git Tutorial, http://www.vogella.com/tutorials/Git/article.html
- A quick note on collaborative development models, https://help.github.com/articles/using-pull-requests
- A succesful Git branching model, http://nvie.com/posts/a-successful-git-branching-model