

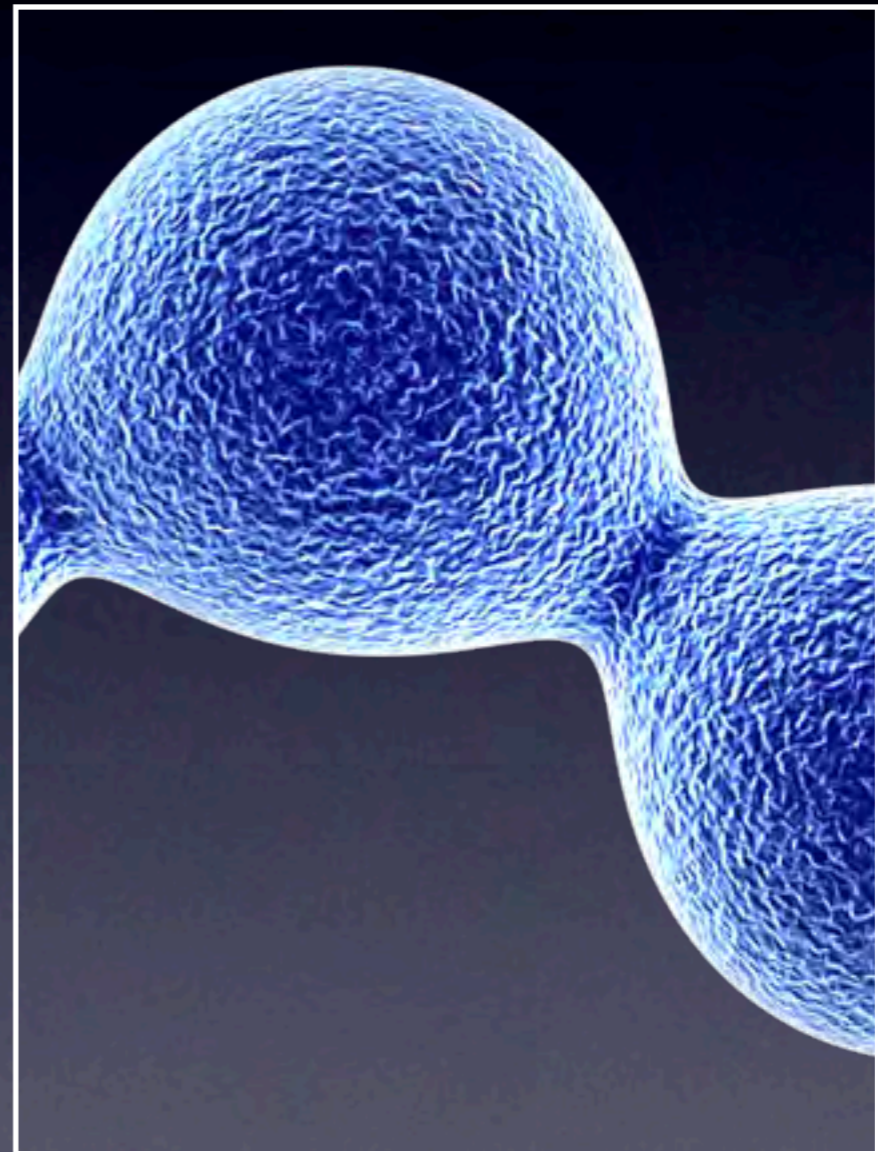
Districarsi tra i sistemi di controllo revisioni

da “Making sense of Revision-Control Systems”
di Bryan O’ Sullivan - ACM Sett. ’09

Massimiliano Picone

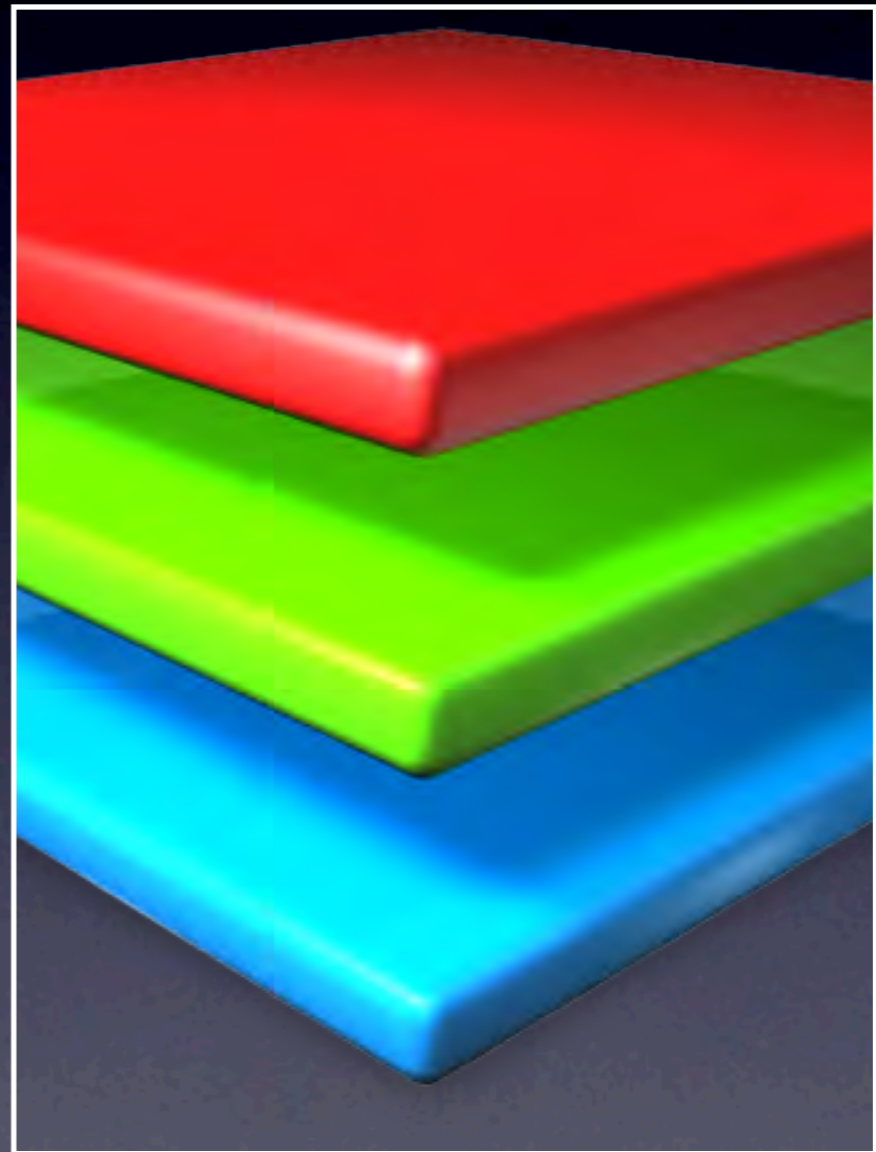
Il software è complicato

- Esigenza di tracciarne lo sviluppo tramite sistemi di controllo revisioni
- CVS e Subversion: modello client-server
- Git e Mercurial: modello peer-to-peer. Più recenti
- Non esiste un modello migliore, dipende dal team



Caratteristiche comuni

- Storico granulare delle modifiche
- Sviluppo in sotto-progetti senza interferenze (*branches*)
- Branch può essere reintegrata nel progetto principale (*merging*)



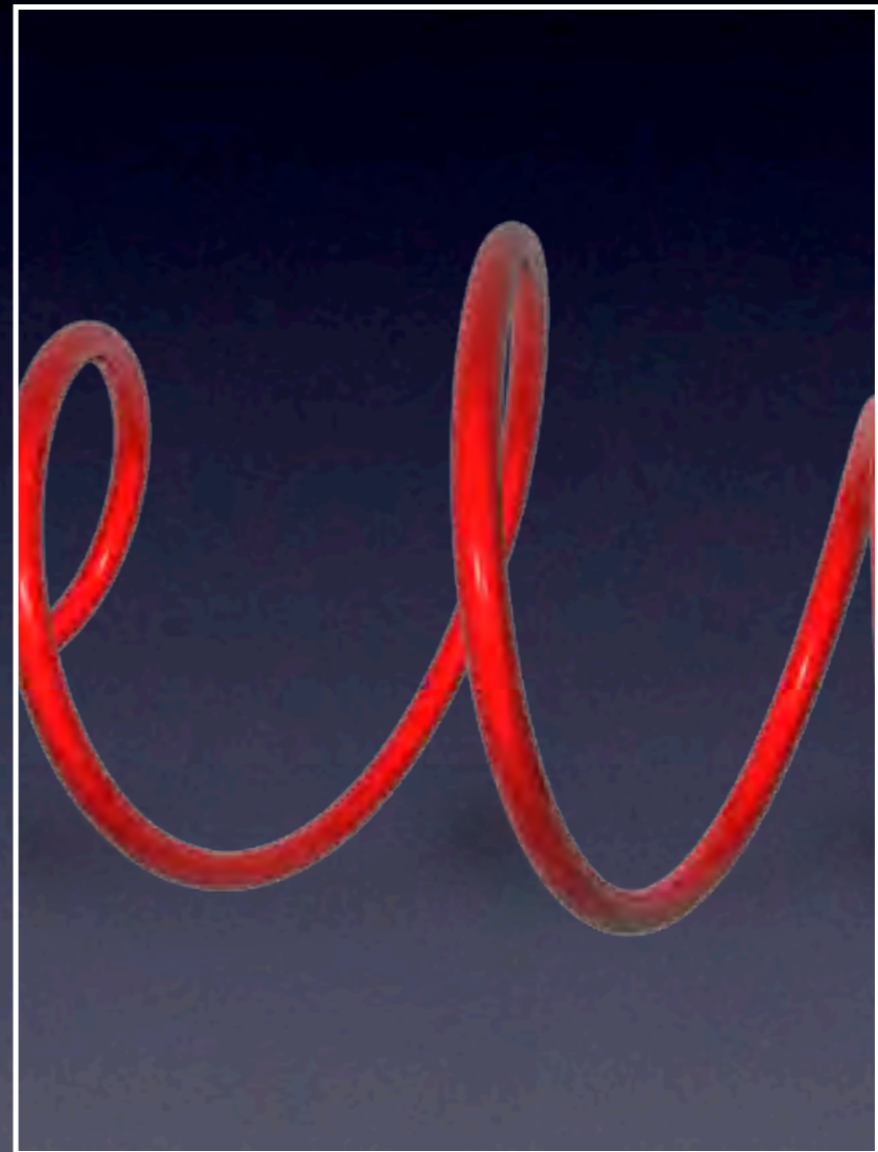
Sviluppo concorrente

- Problematico per progetti grandi
- Sviluppatori preferiscono lavorare in loro branch ma il merging può introdurre conflitti e bug
- Tipi di conflitto: concorrenza, dipendenza, nomi dei file



Subversion

- Sistema di branch lineare.
Es: Alice e Bob lavorano sulla stessa revisione n. 105. Se Alice invia il suo lavoro prima di Bob (rev. 106), questo sarà costretto a fare un merge con le modifiche di Alice prima di poter inviare le sue (rev. 107)
- Se il merge va storto si può perdere il lavoro



Mercurial e Git

- I metadati sono distribuiti su tutti i client, uno storico di tutto il progetto
- Si può quindi salvare il proprio lavoro senza essere costretti a fare un merge



Gradi di libertà

- Subversion non impone una struttura per i branch (propone convenzioni) il che può generare inconsistenze
- Con Git e Mercurial non è possibile invece pubblicare inconsistenze perchè considerano come unità di lavoro tutto il progetto (*repository*)



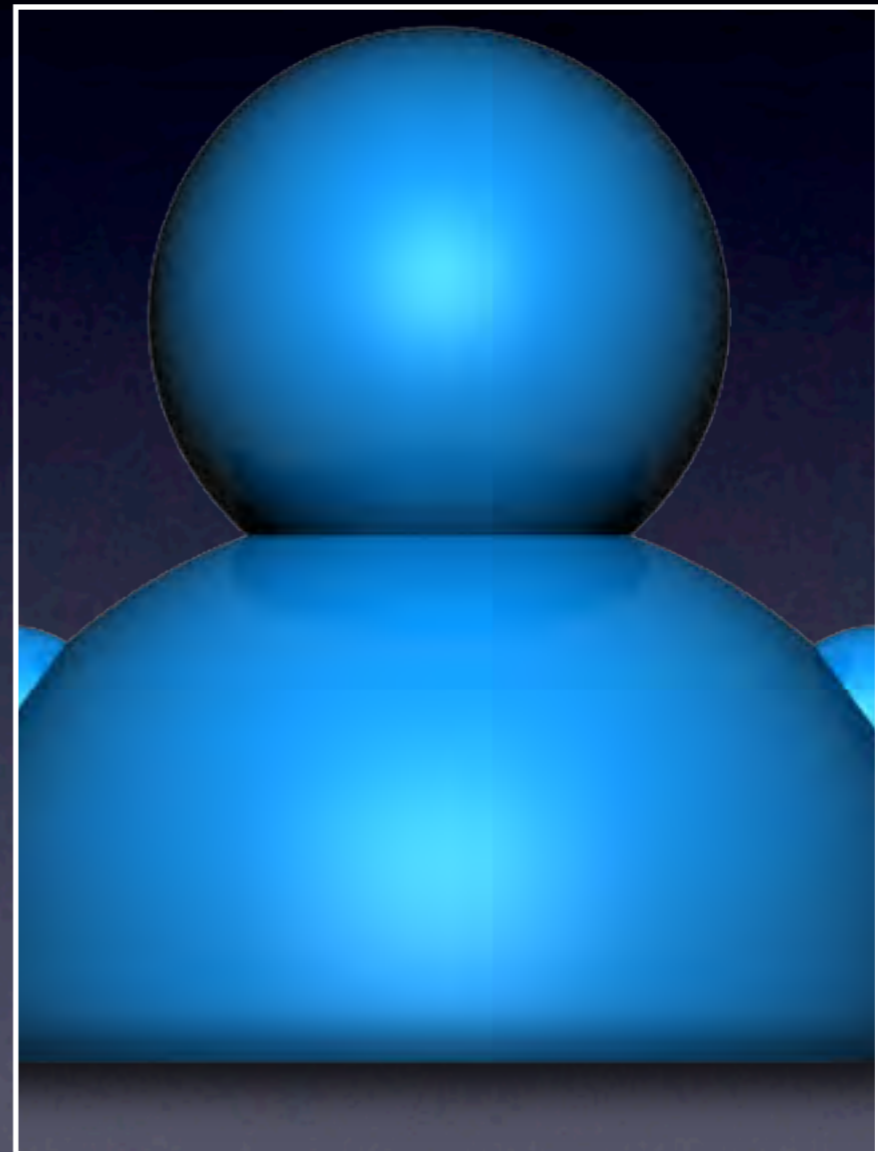
Publicare le modifiche

- In Subversion inviare una modifica (*commit*) implica la pubblicazione. Più adatto in ambienti LAN con diritto di scrittura per tutti
- In Git e Mercurial la pubblicazione va esplicitata, sono adatti per RAD: bastano portatili e Wi-Fi per creare un ambiente di sviluppo. Sicurezza inferiore



Sviluppo “Pull”

- Nel modello distribuito il paradigma “rivedi prima del merge” è implicito. I manager di progetto possono approvare le modifiche dopo averle controllate
- Per ottenere lo stesso comportamento da Subversion occorre configurarlo ad-hoc



I nomi dei file

- Ridenominazioni dei file prima e dopo i merge sono problematici per Subversion
- Mercurial gestisce molto bene queste situazioni anche tra piattaforme e filesystem diversi



FooBar.txt

Bisect: il “trova bug”

- Comando bisect: introdotto da Git e poi adottato anche da Mercurial
- Trova la revisione del progetto nella quale è comparso per la prima volta un bug
- Facilmente automatizzabile con script per creare test



Trasferire i bugfix

- Problematico da una branch all'altra per via delle dipendenze
- Affidabile in Darcs ma è un sistema lento
- Per Git e Mercurial: correggere un bugfix indietro nello storico e farne un merge nel branch attuale



I sistemi centralizzati

- Punti di forza: gestione di file binari (il costo dello spazio su disco si paga una volta sola sul server centrale), possibilità di locking (non presente sui sistemi distribuiti)
- Gestione di politiche di automazione e sicurezza per l'intero team



I sistemi distribuiti

- Riducono i rischi dei merge perchè questi avvengono molto più frequentemente dei sistemi centralizzati
- Incoraggiano lo sviluppo rapido per team dinamici che viaggiano



Il futuro

- Il campo del controllo revisioni ha avuto scarsa attenzione dall'accademia
- Miglioramenti possibili sulla gestione dei grandi progetti con storici molto lunghi e sulle politiche di sicurezza



Conclusioni

- In generale, sistemi centralizzati: più adatti in caso di frequenti modifiche a grandi file binari. Sistemi distribuiti: più adatti per team agili
- Git lento su HTTP e con comandi più difficili
- Subversion ha controlli di sicurezza sul singolo file

