Prova Finale
Controllo delle versioni

Controllo delle versioni: a cosa serve?

Tenere traccia dei cambiamenti

Semplificare la collaborazione

Gestione di diverse diramazioni (branch) di sviluppo

Differenti tipologie di sistemi

Centralizzati

Distribuiti

CVS

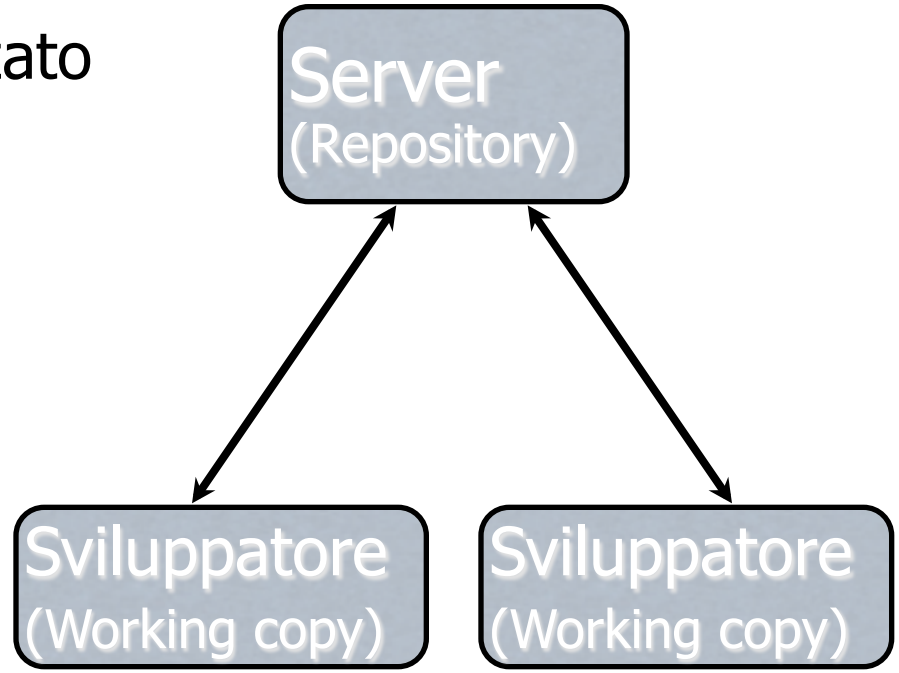
Nella Prova Finale utilizzeremo Subversion (SVN)

Subversion

Controllo delle versioni centralizzato

Molto utilizzato

Supportato da molti client e IDE



Operazioni di base

checkout	Crea una nuova working copy copiata dal repository
commit	Copia il codice modificato nel repository
update	Aggiorna la working copy con l'ultima versione disponibile nel repository
add/mv/rm/merge/ revert	Gestione dei file sotto controllo di versione

Checkout

- Il checkout crea nel client la copia locale di lavoro
- Nella copia locale vengono copiati tutti i file della copia più aggiornata presente nel repository
- Viene indicato il **numero di revisione** corrente
- Se il repository è appena stato creato?
  - Il primo checkout creerà un cartella “vuota”
  - Il numero di visione iniziale è zero
  - Vanno aggiunti i file che saranno sotto controllo di versione
    - Copio il file nella cartella
    - Eseguo il comando **add**

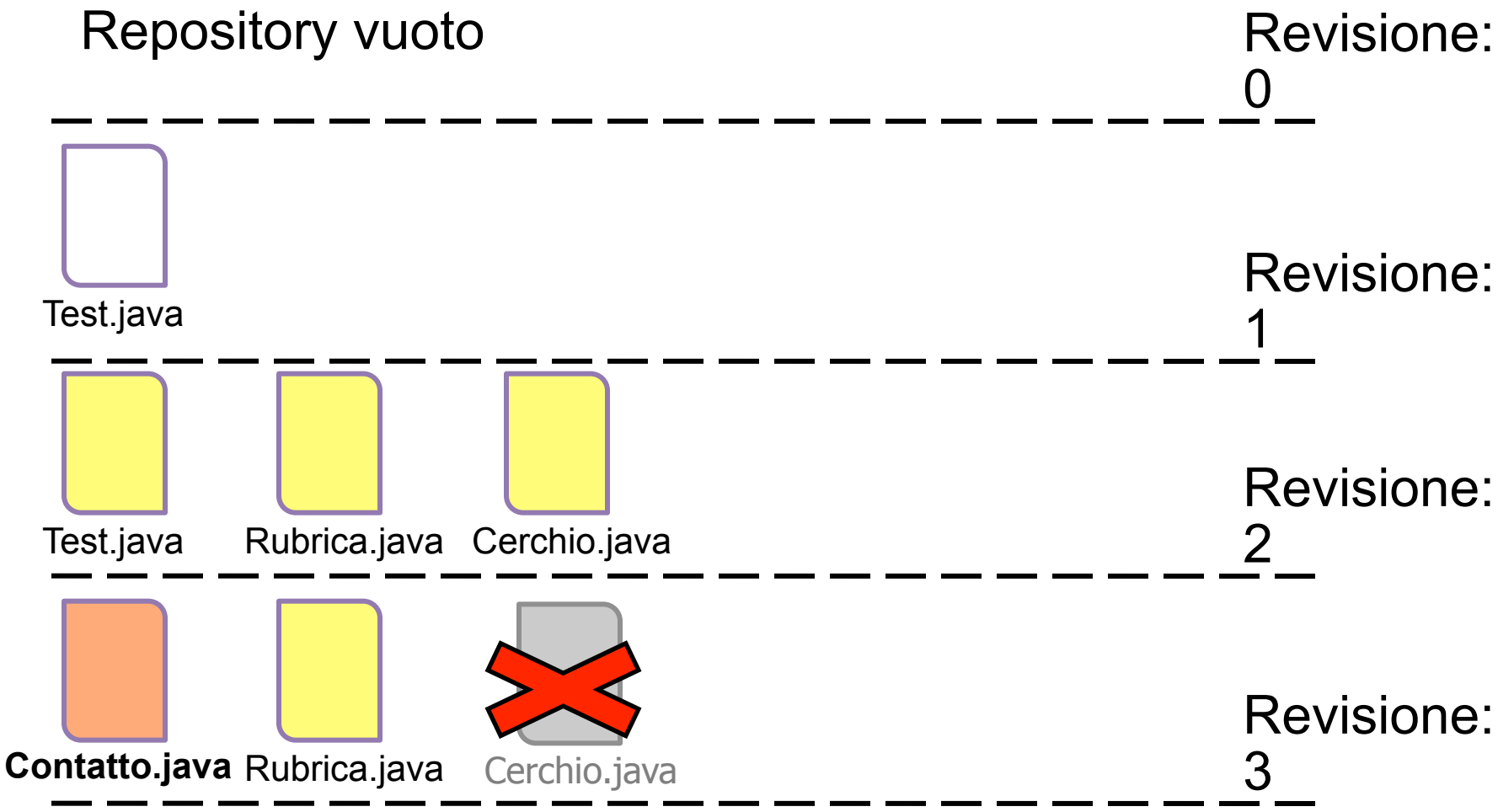
Commit e Update

- Il comando **commit** esegue un upload sul repository di tutte le modifiche locali:
  - File modificati
  - File/Cartelle aggiunti
  - File/Cartelle rimossi
  - File/Cartelle spostati o rinominati
- Il comando **update** aggiorna la copia locale con l'ultima versione presente sul repository
  - E' buona norma fare un **update** prima di un **commit**

Commentare i Commit

- E’ buona norma aggiungere un commento ad ogni commit
  - In quali casi? Sempre!
- Guardando lo storico dei commenti è possibile capire quali aspetti del progetto sono in corso si sviluppo
- Molto utile per coordinare il lavoro di più programmatori
- Utile anche per trovare bachi relativi ad una certa funzionalità
  - Basta cercare il commit in cui è stata aggiornata quella particolare funzionalità
- Sempre meglio mettere nello stesso commit modifiche correlate tra loro (implementano o aggiornano una particolare funzionalità)

Numeri di Revisione



Quali file mettere sotto il controllo di versione?

- Ora che sappiamo come funziona svn, quali file mettere sotto controllo di versione? Perché?

- I file sorgenti (Java, C, C++,…) ?
- Gli eseguibili, i file .class ?
- L'output di javadoc?
- Le immagini?
- I file Latex? (sapete cos'è?)
- I PDF?

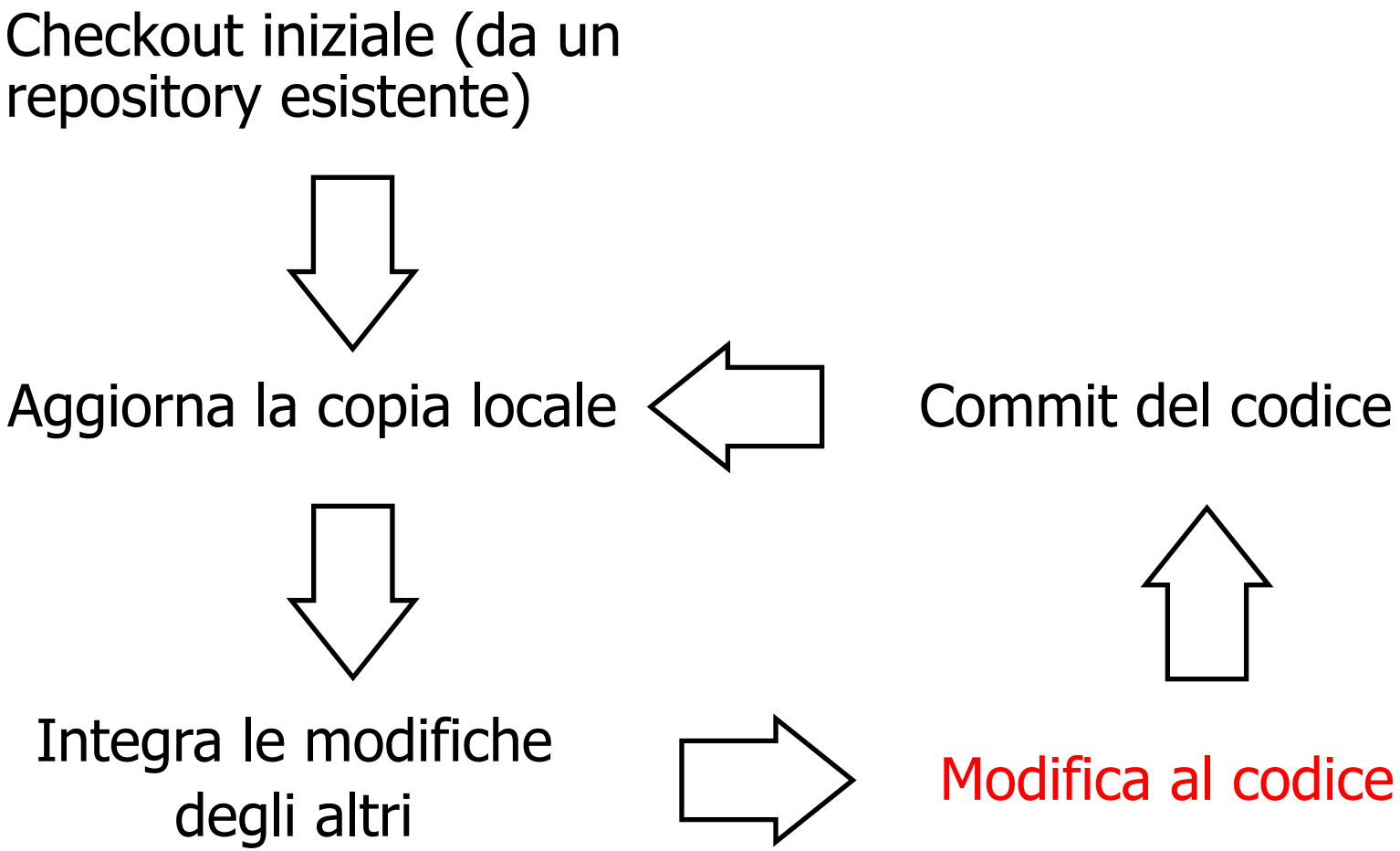
Memorizzazione nella copia locale

- Nella copia locale tutte le informazioni sul controllo di versione sono contenute nelle cartelle nascoste “.svn”
  - E’ presente una cartella “.svn” in ogni cartella sotto controllo di versione
- Se i file di progetto vengono distribuiti è importante cancellare le directory “.svn”
  - commit accidentali da parte di altri utenti!
  - Usare il comando **`export`**

Gestione dei Conflitti

- Paradigma generalmente utilizzato:
  - Ogni utente può modificare qualsiasi file
  - Se dopo un commit c'è un conflitto, lo risolvo!
- Ipotesi: generalmente i programmatori lavorano a parti diverse del progetto. Il conflitto è un evento raro.
- Se si lavora a parti diverse dello stesso file (metodi diversi di una stessa classe) non vi è nessuno conflitto.
- Subversion segnala un conflitto solo quando le modifiche coinvolgono le stesse righe di codice:
  - Vengono mostrate le due versioni in conflitto
  - Scelgo quale delle due versioni è quella corretta
  - Oppure faccio il **merge** manuale!

Tipica sequenza di utilizzo



E’ sempre possibile tornare indietro

Il repository contiene **tutto** il codice che e’ stato oggetto di un’operazione di commit

E’ sempre possibile tornare indietro ad una versione precedente di un file o di tutto il progetto

Questo previene le perdite di dati e limita gli effetti degli errori

Collaborazione

Non e' necessario sincronizzare manualmente differenti copie

Quando effettuano un'operazione di **commit** gli sviluppatori sono forzati a risolvere gli eventuali conflitti

Il codice nel repository contiene tutti i cambiamenti fatti dagli sviluppatori

Diramazioni (branching)

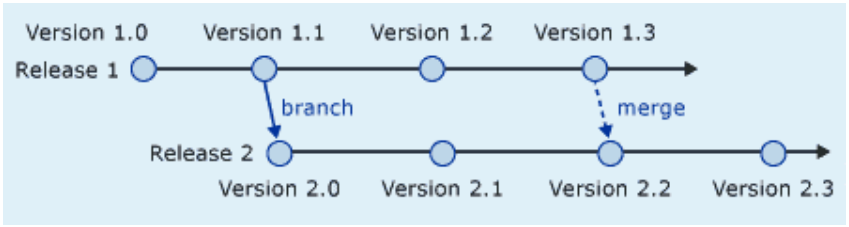
Di quante versioni ho bisogno?

trunk	versione principale del codice
branches per **nuove caratteristiche**	nuove caratteristiche possono essere sviluppate in modo indipendente
Branches per **versioni diverse**	codice specifico per alcune piattaforme, codice legacy

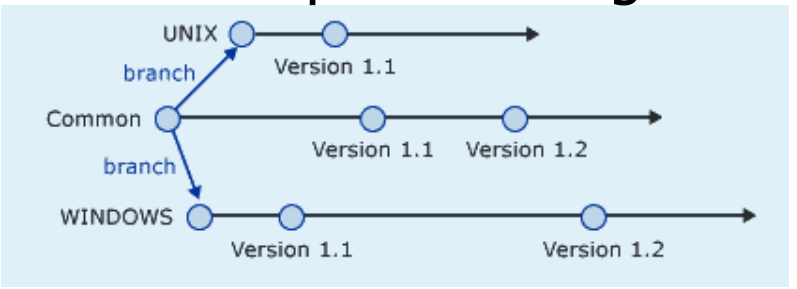
Si usa l'operazione **copy**

Esempi di branching

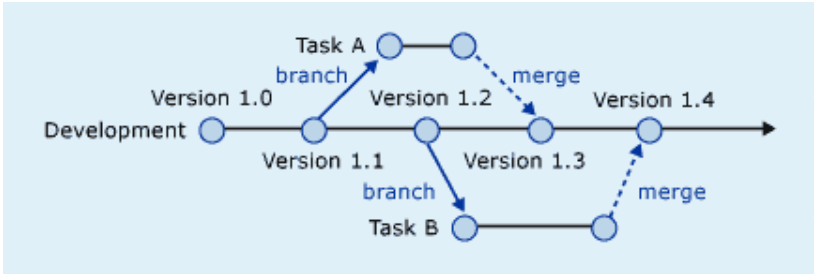
branch per rilascio



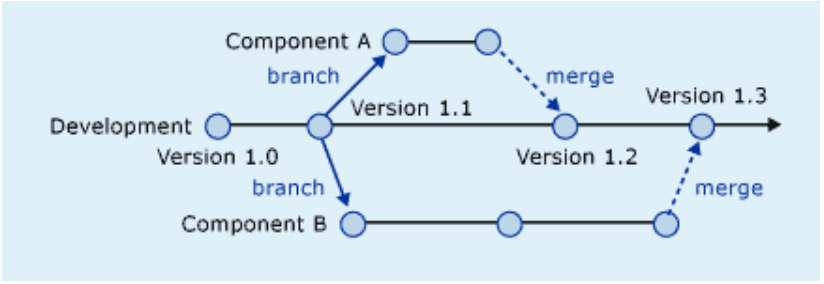
branch per tecnologia



branch per attivita'



branch per componente



Etichette (Tag)

- Le etichette sono un modo per dare un nome esplicito ad una particolare versione del progetto
- Per esempio: “Release 1.0”
- Anche in questo caso si usa l’operazione **copy**
- Nessuna duplicazione effettiva di dati (nel repository)
- Permette di trovare facilmente nel repository i file che fanno parte di una versione taggata
- E’ un’alternativa più elegante dal segnarsi il numero di revisione di una particolare release

Alcuni consigli

Effettuare spesso le operazioni di **commit** per evitare perdite di dati e conflitti difficili da risolvere

Effettuare operazioni di **commit** solo con codice che funziona, altrimenti sara' danneggiato anche il codice di chi collabora

Codice che funziona significa non solo codice compilabile, ma anche codice testato!

Riferimenti Subversion

- Guida HTML.it
  http://programmazione.html.it/guide/leggi/147/guida-subversion/

- Guida Ufficiale
  http://svnbook.red-bean.com/index.en.html

- Tutorial Introduttivo:
  http://www.simonecarletti.it/blog/2007/03/strumenti-di-sviluppo-subversion-svn/