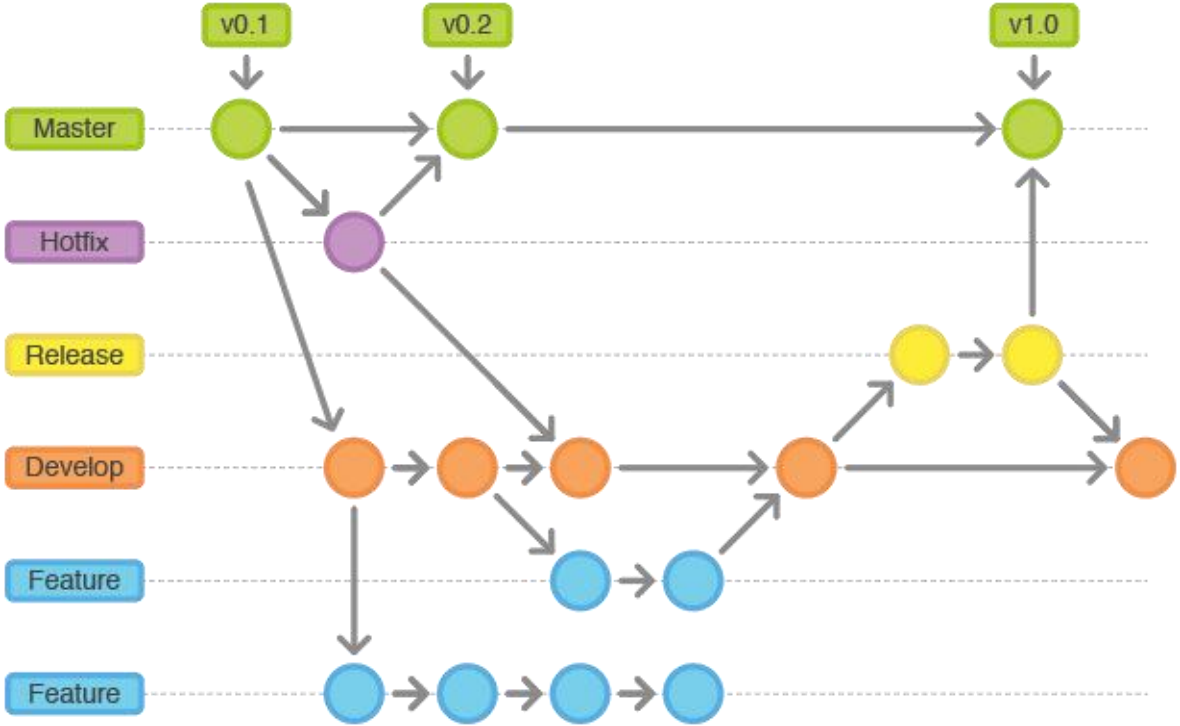
Controllo delle versioni con git

Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it

Controllo delle versioni

- Lo sviluppo di applicazioni avviene *nel tempo* e con il contributo di *squadre di* sviluppatori che possono essere anche molto grandi
- È improbabile che nello sviluppo non si verifichi l'introduzione di *bug*
- Per uno sviluppo efficiente e sostenibile è necessario:
  - tenere traccia dell'evoluzione del codice sorgente e della documentazione
  - tenere traccia della scoperta di errori
  - sviluppare *patch* per correggere gli errori ed integrarle nello sviluppo
  - tenere traccia delle nuove *funzionalità* che si vogliono aggiungere al software, integrandole in maniera fluida con il progetto principale
  - associare una *versione* dell'applicazione al codice sorgente corrispondente
- Controllo delle versioni: metodologie ed applicazioni per gestire tutti gli aspetti della risorsa "codice sorgente"

La storia di un'applicazione



Operazioni tipiche del controllo delle versioni

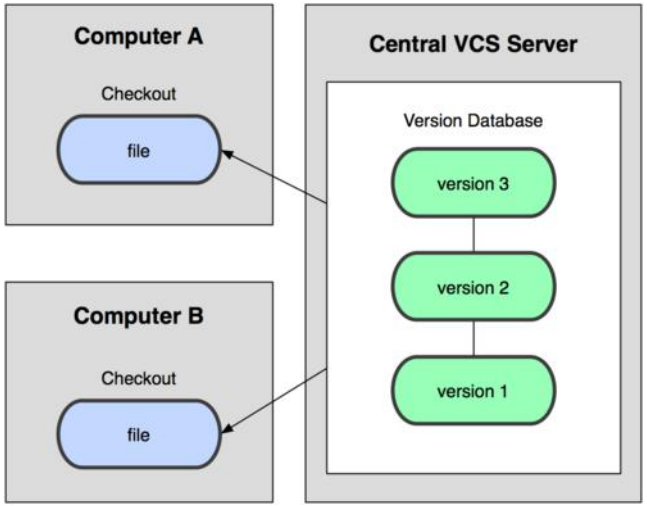
- *Commit*: è un'operazione *atomica* in cui si chiede al sistema di CV di rendere definitivo un insieme di modifiche e di renderle disponibili a tutti gli utenti
- *File locking*: sistema molto semplice di gestione "atomica" dei file. Uno sviluppatore alla volta ha accesso in scrittura alle copie nel "deposito" del codice sorgente.
- *Merge*: più sviluppatori possono modificare lo stesso file. Il primo sviluppatore che effettua il commit delle modifiche ha sempre successo. Gli altri sviluppatori possono *fondere* le loro modifiche. Possono verificarsi dei *conflitti* che devono essere risolti manualmente.
- *Tagging:* uno specifico commit può essere associato ad un'*etichetta*, ad esempio per tracciare una versione specifica dell'applicazione.

git

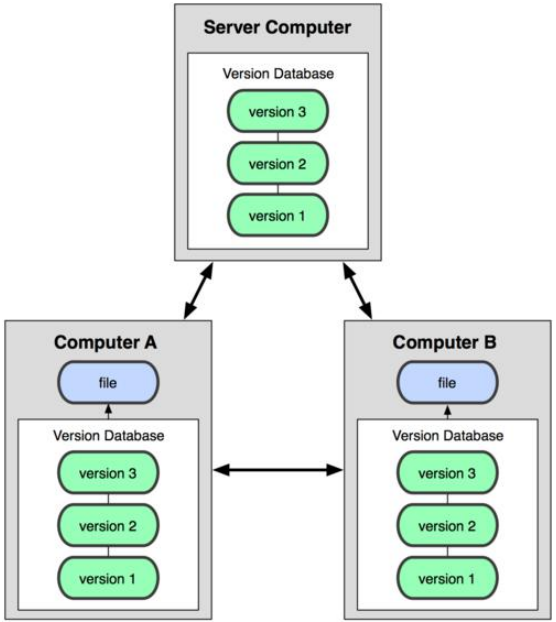
- Nato dalla comunità degli sviluppatori di Linux
  - Linus Torvalds, 2005
- Obiettivi iniziali:
  - velocità
  - supporto per lo sviluppo non lineare (migliaia di *branch* parallele)
  - completamente distribuito
  - in grado di gestire progetti grandi e complessi (es, Linux) in maniera efficace

Modello distribuito

- git utilizza un modello completamente distribuito
- le versioni sono identificate da un hash SHA-1



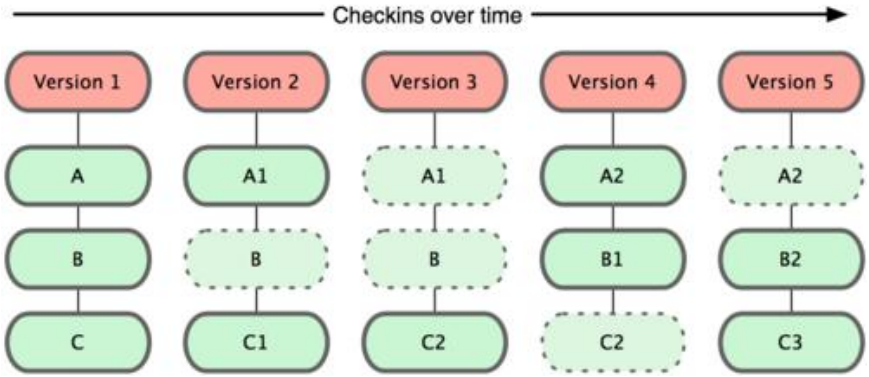
Modello centralizzato



Modello distribuito

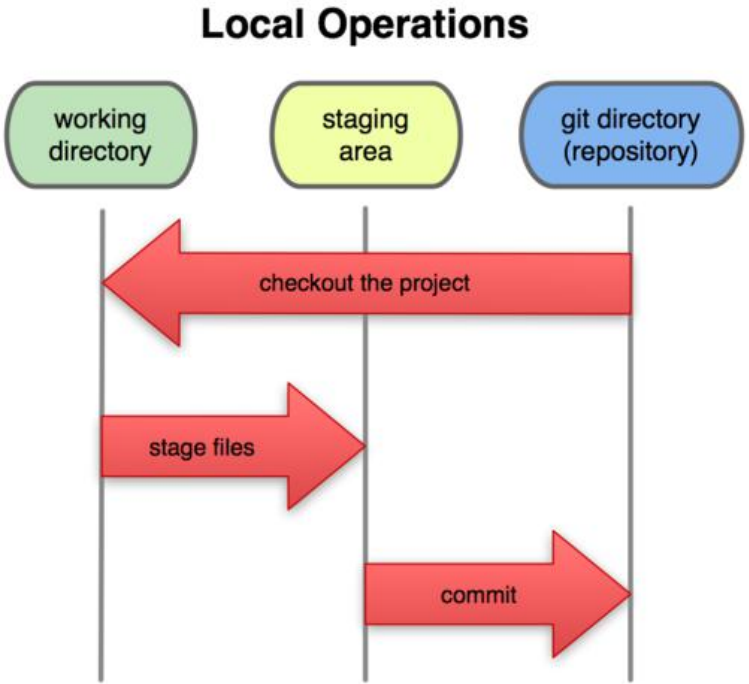
Snapshot

- Internamente, git crea delle *istantanee* del lavoro
- Basato sul concetto di *filesystem*
- Permette velocemente di portarsi avanti e indietro nel flusso di lavoro



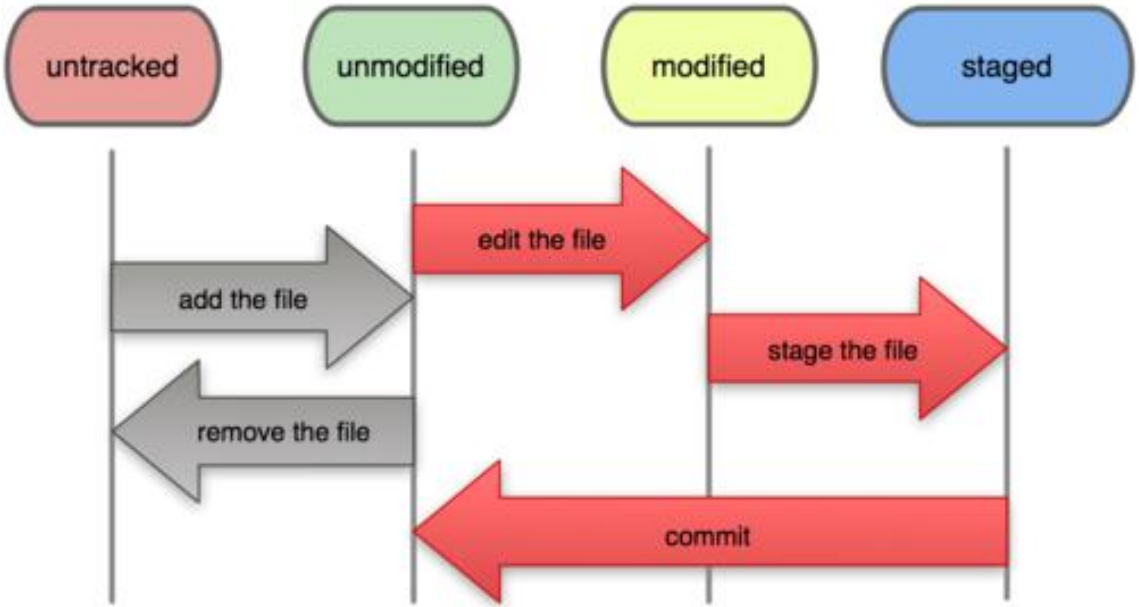
Aree di un progetto locale

- Un progetto locale ha tre aree: la *working directory*, la *staging area* e la *git directory*



Ciclo di vita di un file

- La creazione o la modifica di un file avviene nella copia locale
- L'operazione di commit avviene *localmente* dopo aver modificato il contenuto



GitHub

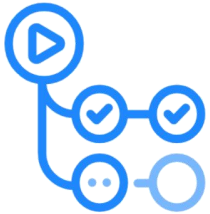
- github.com è un sito che ospita *copie* di reposiroty git
- Utilizzato da molti progetti open source (anche dal Kernel Linux, ma solo come *mirror*)
- Offre spazio gratuito per progetti open source

- Devo utilizzare github per usare git? No!
  - git è un sistema di controllo delle versioni distribuito
  - github è solo una possibile copia nel sistema distribuito
  - è associato ad un'applicazione web che permette di visualizzare il contenuto dei progetti ed interagire con gli sviluppatori

GitHub Actions

- Un aspetto fondamentale delle applicazioni moderne è la possibilità di *testarle automaticamente*
  - per individuare *bug*
  - per individuare *regressioni*
- Tecniche moderne di ingegneria del software prevedono di eseguire automaticamente *batterie di test* per verificare la qualità del software ogni volta che vengono effettuate modifiche ai sorgenti

- GitHub Actions:
  - servizio basato su *container*
  - permette di eseguire automaticamente le batterie di test ad ogni commit



Preparazione all'uso di git

- Imposta il nome utente e l'email con cui verranno “firmati” i commit

```
git config --global user.name “Bugs Bunny”
git config --global user.email bugs@example.com
```

- Si può utilizzare `git config --list` per verificare la loro corretta impostazione
- Omettengo il flag `--global` si possono impostare variabili di configurazione locali per ciascun repository

Creare un repository locale

- Ci sono due scenari principali:
  - Voglio creare un *clone* di un repository già esistente da un server remoto:
  - **`$ git clone <url> [local dir name]`**

  - Voglio creare *da zero* un nuovo repository locale:
  - **`$ git init`**

    `$ git add README.md`

    `$ git commit –m “initial project version”`

Commit di file

- La prima volta che aggiungiamo un file al repository, *ed ogni volta che ne effettuiamo una modifica che vogliamo salvare*, dobbiamo aggiungerlo all'area di staging:
  - **`$ git add README.txt hello.java`**
- Questo comando crea uno *snapshot* della versione corrente
- Per spostare il file dall'area di staging all'interno del repository, dobbiamo effettuare l'operazione di *commit*:
  - **`git commit –m “Fixing bug #22”`**

- Attenzione: questi comandi operano unicamente sulla *versione locale* del repository!

Status e diff

- È possibile vedere lo stato attuale del repository locale:
  - **`$ git status`**

- Si può chiedere di mostrare cosa è stato modificato ma ancora non inserito nell'area di staging:
  - **`$ git diff`**

- Per mostrare le differenze nell'area di staging:
  - **`$ git diff --cached`**

Logs

- Si può far mostrare l'elenco delle modifiche che sono state effettuate nelle varie versioni:
  - **`$ git log`**

- Se si è interessati solo alle ultime 5 modifiche:
  - **`$ git log -5`**

Sincronizzare repository remoti

- Per recuperare da un repository remoto le ultime modifiche pubblicate:
  - `$ git pull origin main`
- Per pubblicare sul repository remoto i commit dal repository locale:
  - `$ git push origin main`
- Informazioni sul/sui repository remoti:
  - `$ git remote -v`

Branching

- Si possono creare rami multipli di lavoro, chiamati *branch*
- Per creare una branch:
  - **`$ git branch <nome>`**
- Per mostrare tutte le branch locali:
  - **`$ git branch`**
- Per passare da una branch all'altra:
  - **`$ git checkout <nome>`**
- Per “fondere” insieme le branch:
  - **`$ git checkout master`**
  - **`$ git merge <name>`**

Comandi di git

comando	descrizione
`git clone url [dir]`	copia un repository git in modo da potervi aggiungere qualcosa
`git add files`	aggiunge il contenuto dei file all'area di staging
`git commit`	registra un'istantanea dell'area di staging
`git status`	mostra lo stato dei file nella directory di lavoro e nell'area di staging
`git diff`	mostra la differenza tra ciò che è in staging e ciò che è modificato, ma non in staging.
`git help [command]`	ottieni informazioni di aiuto su un particolare comando
`git pull`	recupera da un deposito remoto e cerca di fonderlo con il ramo corrente
`git push`	spinge i nuovi rami e i dati in un repository remoto
altri: `init, reset, branch, checkout, merge, log, tag`	

Alcune risorse

- Dalla linea di comando:
  - git help <verb>
  - git <verb> --help
  - man git-<verb>
  - <verb> è uno dei comandi di git (commit, add, config, ...)

- Libro online: https://git-scm.com/book/en/v2
- Tutorial: https://schacon.github.io/git/gittutorial.html
- Git for computer scientists: https://eagain.net/articles/git-for-computer-scientists/
- Dispensa caricata sul sito del corso