CONCURRENT VERSIONING SYSTEMS

Gestione delle versioni di un software

- I sistemi software
  - non sono realizzati in una sola sessione di programmazione
  - spesso non sono realizzati da un solo programmatore
  - Non sono rilasciati una sola volta
    - La correzione dei difetti può portare a nuovi rilasci
    - L'introduzione di nuove funzionalità porta a nuovi rilasci
  - Spesso un software necessita di essere rilasciato in diverse configurazioni
    - Versioni complete e versioni ridotte
    - Versioni in diverse lingue
    - Versioni diverse per diverse configurazioni hardware
    - …

- Per tutte queste necessità, è opportuno utilizzare un sistema di gestione del ciclo di vita di software, o quantomeno, della storia delle sue versioni

Requisiti di un sistema per la gestione delle versioni

- Supporto per la gestione delle versioni
  - Identificazione delle versioni e delle release
  - Gestione della memorizzazione
  - Registrazione dello storico delle modifiche
  - Sviluppo indipendente
    - Gestione dei branch e delle versioni concorrenti
  - Supporto alla build automation
  - Supporto ad attività DevOps
    - Continuous Integration
    - Continuous Delivery / Deployment

Protocolli e strumenti per la gestione delle versioni

- CVS
- SVN
- Git
- …

CVS: Concurrent Versioning System

- Sistema di controllo delle versioni di un progetto legato alla produzione e alla modifica di file. In pratica, permette a un gruppo di persone di lavorare simultaneamente sullo stesso gruppo di file (generalmente si tratta di sorgenti di un programma), mantenendo il controllo dell'evoluzione delle modifiche che vengono apportate.

- Per attuare questo obiettivo, il sistema CVS mantiene un deposito centrale (*repository*) dal quale i collaboratori di un progetto possono ottenere una copia di lavoro. I collaboratori modificano i file della loro copia di lavoro e sottopongono le loro modifiche al sistema CVS che le integra nel deposito.

- Il compito di un sistema CVS non si limita a questo; per esempio è sempre possibile ricostruire la storia delle modifiche apportate a un gruppo di file, oltre a essere anche possibile ottenere una copia che faccia riferimento a una versione passata di quel lavoro.

- Storicamente, il primo strumento open source di gestione della configurazione con ampia diffusione

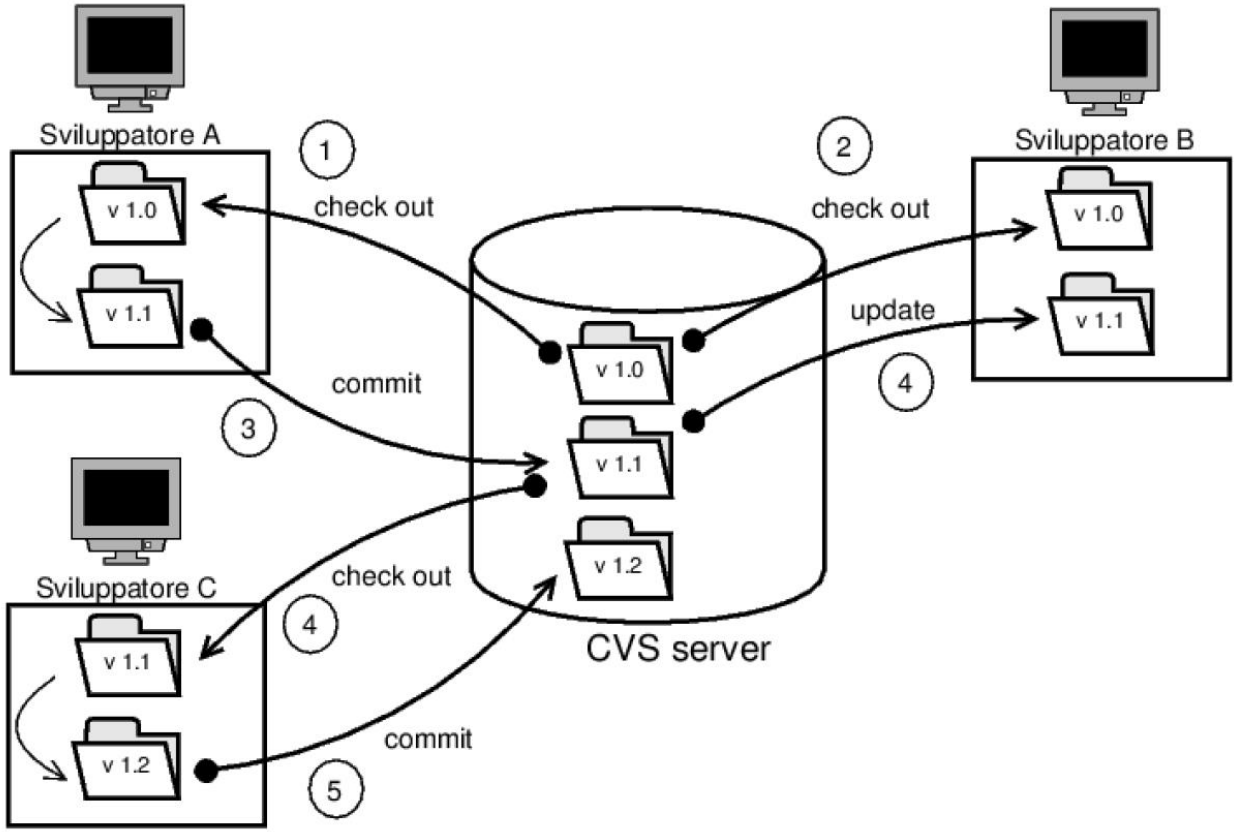
Modello Lock/Modify/Unlock

- In principio, l'unico modello secondo il quale più programmatori accedevano in concorrenza ai diversi file di un progetto era il modello "*lock/modify/unlock*"
  - Secondo questo modello un utente che vuole modificare un file del progetto, prima di tutto lo blocca (*lock*), impedendo a chiunque altro di modificarlo, dopodichè, quando ha terminato le modifiche lo sblocca (*unlock*)
  - Questa strategia, per quanto garantisca la massima sicurezza da problemi di manomissione contemporanea involontaria, non ottimizza nel modo migliore le operazioni
    - Adoperando questo modello, si tende a spezzettare il più possibile un progetto, in modo da ridurre gli impedimenti al lavoro causati dai lock

Modello Copy/Modify/Merge

- In alternativa, il modello *Copy/Modify/Merge* prevede che:
  1. Lo sviluppatore A scarica una copia del progetto (*working copy o sandbox*) dal server CVS (*repository*)
  2. Applica liberamente tutte le modifiche. Nel frattempo altri programmatori (B) potrebbero fare lo stesso
  3. Al termine del suo lavoro il programmatore A aggiorna il progetto sul server CVS (*commit*)
  4. Altri programmatori potrebbero richiedere aggiornamenti della loro working copy (update) al repository o generare delle ulteriori versioni (commit)

Modello Copy/Modify/Merge



Conflitti

- Nel caso in cui due programmatori modificano lo stesso file, il sistema CVS può fondere (*merge*) le due versioni, sovrapponendo le modifiche, allorchè si riferiscano a linee di codice diverse
- Se invece ci sono modifiche alle stesse righe di codice si verifica un *conflitto*
  - La soluzione del conflitto è in questo caso demandata ai singoli programmatori: la versione unificata che viene generata diventa la nuova versione di riferimento
  - In alternativa si potrebbe scegliere di mantenere entrambe le versioni come alternative, generando un *branch*

CVS

- Il sistema CVS è un software, presente per diversi sistemi operativi, che consente di gestire a linea di comando le principali operazioni previste dai modelli lock/modify/unlock e *copy/modify/merge*

- Il lato server gestisce il *repository*, contenente sia tutti i file da gestire che tutte le informazioni sulle versioni
  - In alternativa il deposito potrebbe anche trovarsi sulla macchina client

- Il lato client consente di effettuare tutte le operazioni riguardanti la copia locale (*sandbox)* del progetto

Operazioni CVS

- Ogni persona coinvolta nel progetto, ha una copia locale dei file (*sandbox*)
- Chi avvia il progetto crea per la prima volta il repository (*Make new module*), indicando anche quali directory dovranno essere gestite
- Successivamente un qualsiasi collaboratore può aggiungere nuovi file/directory al CVS (*add*)
- Un collaboratore che voglia inserirsi nel CVS dovrà per prima cosa effettuare il *Checkout* per prelevare dal repository le versioni più recenti di ogni file

Operazioni CVS

- Sui file presenti nella propria *sandbox* si possono effettuare le seguenti operazioni:
  - *Checkout* (o update): preleva una copia aggiornata dal repository;
    - Se copia locale e copia del repository non coincidono viene segnalato un *conflict*
    - Dopo il checkout, la copia locale è in stato di *lock* e non può essere modificata
    - Di solito con checkout si intende il primo prelievo, con update i successivi
  - *Edit*: richiede il permesso di scrivere sul file locale
    - Se il file è già in stato di edit da parte di qualche altro utente, viene segnalato il rischio di modifiche concorrenti (nel caso di file binari o di politica di lock/modify/unlock viene impedito l'accesso)
  - *Commit*: rende pubbliche a tutti le proprie modifiche al file
    - Le modifiche vengono propagate al repository. Il repository incamera il file ricevuto come nuova versione; le versioni precedenti rimangono reperibili

Operazioni CVS

- *Gestione conflitti*
  - Se due utenti vanno a modificare in concorrenza lo stesso file, e il primo di essi effettua il commit, verrà impedito al secondo di fare lo stesso
    - In questo caso si consiglia al secondo di fare un update: il sistema nota la differenza tra la versione sul repository e quella locale e popone alcune soluzioni semiautomatiche (*merge*) per la soluzione dei conflitti. Al termine, il secondo utente avrà una versione locale che tiene conto sia delle proprie modifiche che di quelle degli altri utenti. Di questa versione potrà essere fatto il commit, ottenendo quindi una versione successiva
- Generazione branch
  - Genera un ramo “alternativo” nella storia del file (se ne terrà conto nella diversa numerazione: ad esempio dopo 1.2 ci sarà 1.2.1 anzichè 1.3)
    - Sono disponibili funzionalità per vedere graficamente tutta la “storia” delle versioni del files
- Fusione tra versioni diverse
- Eliminazione copia locale
- Eliminazione originale (da operare direttamente sul repository)

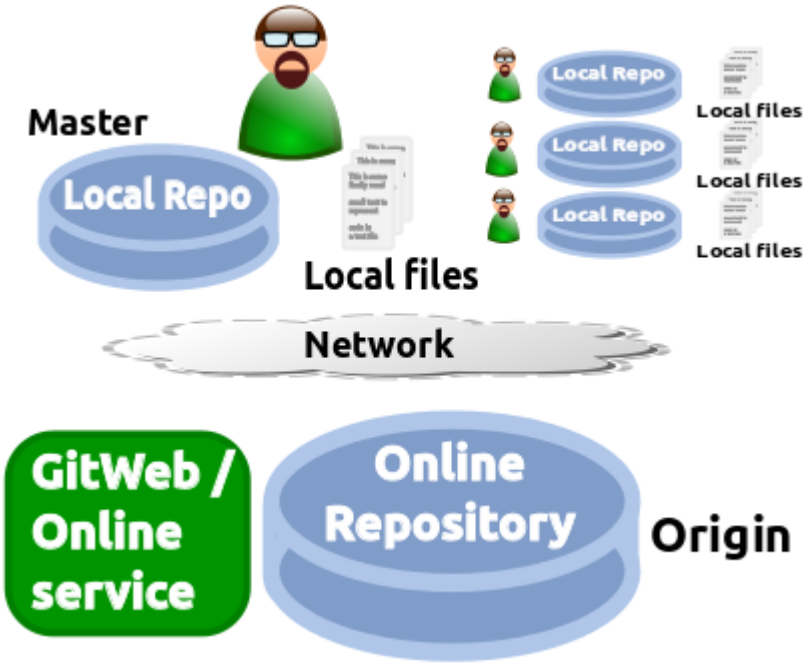
GIT & GITHUB

Git

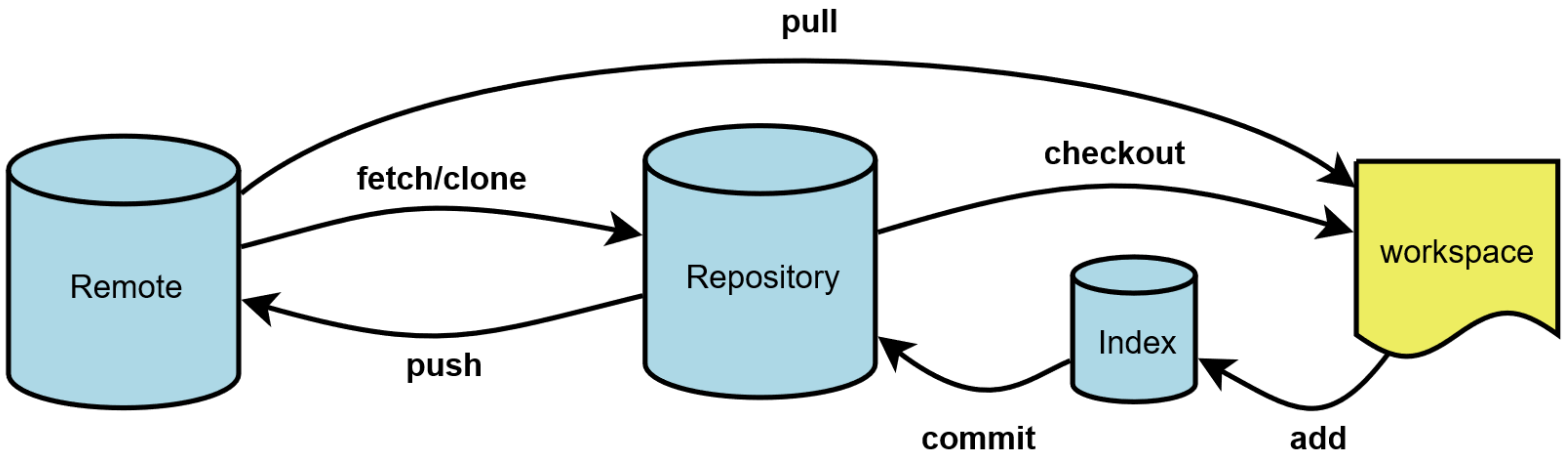
- Proposto da Linux Torvalds
  - http://git-scm.com/
- Si riferisce ad un paradigma più aperto, nel quale chiunque può partecipare ad un progetto:
  - Biforcando (Fork) un progetto esistente
  - Proponendo le sue aggiunte al progetto (patch)

Git Architecture

- L'architettura di Git è distribuita
- A differenza di CVS ed altri, non esiste un'unica copia centralizzata del progetto
- Esiste, però, una copia di riferimento del progetto (**Master**) gestita solitamente dal fondatore del progetto
- Ogni utente può creare una copia locale dell'intero progetto
- L'utente locale può chiedere al fondatore di propagare nel master una propria modifica al progetto proponendo una **Patch**



RELAZIONE TRA OPERAZIONI LOCALI E REMOTE

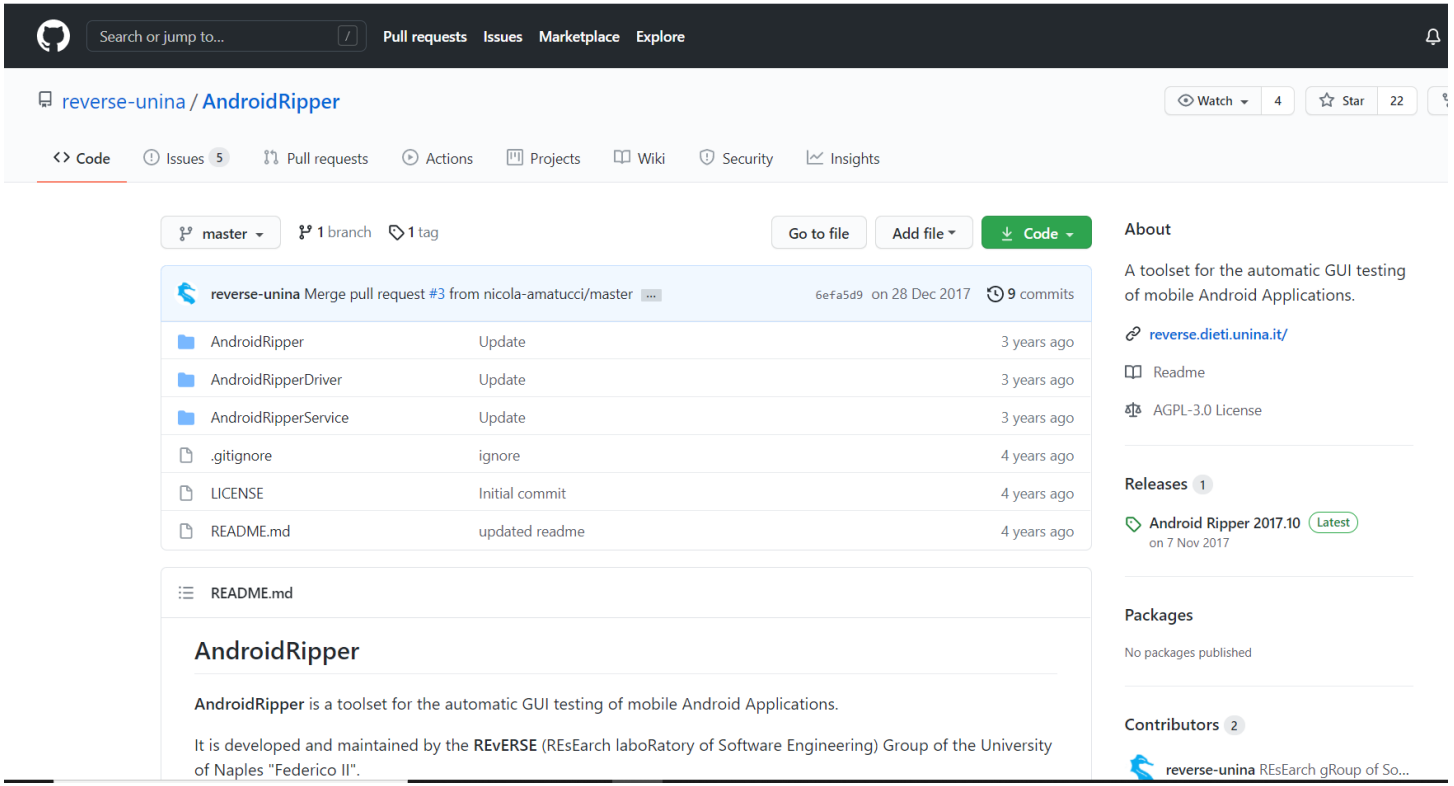


GIT HOSTING

- Il protocollo git è particolarmente utilizzato in progetti open source distribuiti sul Web
- E' particolarmente semplice sfruttare le potenzialità di git basandosi su piattaforme che mettono a disposizione funzionalità di hosting, in gran parte gratuite
  - **GitHub** è il più famoso fornitore di hosting, in continua espansione e acquisito nel 2018 da Microsoft
    - https://github.com/
  - GitLab, anch'esso in rapida espansione, che consente anche la possibilità di ospitare progetti su proprie risorse di hosting (la piattaforma GitLab è anch'essa open source)
    - https://about.gitlab.com/
  - Bitbucket, anch'esso utile anche per ospitare su proprie risorse oltre che su quelle a disposizione sul sito. E' stato acquisito da Atlassian
    - https://bitbucket.org/

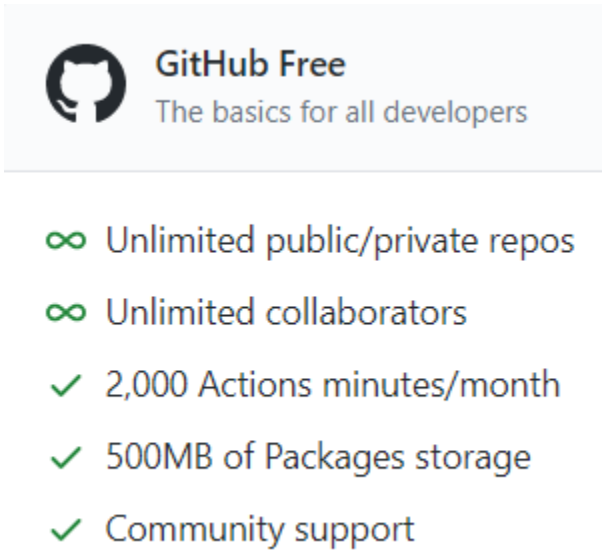
NAVIGAZIONE NEL CODICE

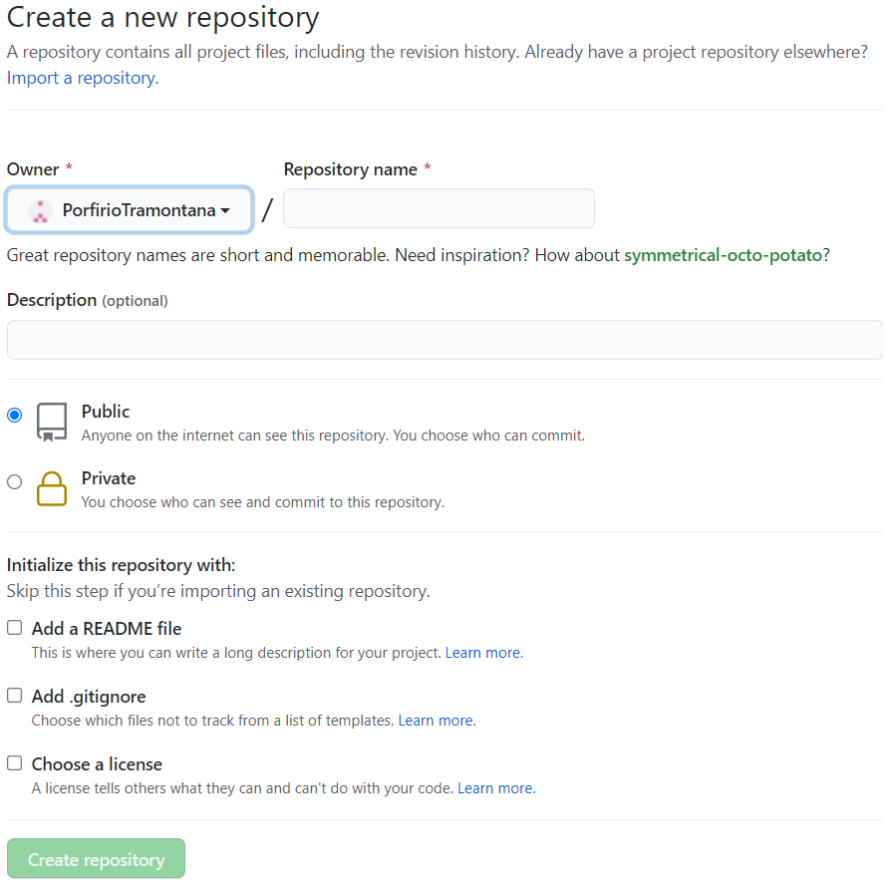
- I Repository nascono concettualmente come pozzi di storage di tutta la storia del codice e degli altri artefatti di un Progetto, durante tutto il suo ciclo di vita



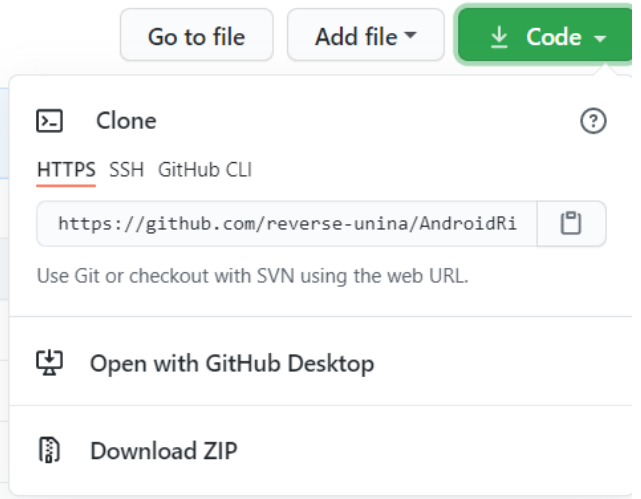
CREAZIONE DI UN PROGETTO

- I progetti possono essere pubblici o privati
  - Dato lo spirito open source della piattaforma, i progetti Pubblici sono completamente gratuiti e hanno meno limitazioni di quelli private
  - Alcune funzionalità più avanzate sono disponibili in quantità limitata o possono essere sbloccate solo a pagamento





IMPORTAZIONE DI UN PROGETTO ESISTENTE



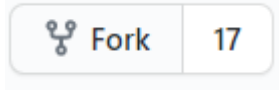
- Lo stato attuale di un progetto può essere scaricato istantaneamente con il pulsante Clone (Code)
- Con Download Zip il progetto viene scaricato in locale e può poi essere importato in un IDE
- Dall'indirizzo fornito, invece, può essere gestito in un client git (ad es. Github Desktop) oppure direttamente da un IDE
  - Login e Password sono necessari nel caso si voglia successivamente continuare ad evolvere il progetto

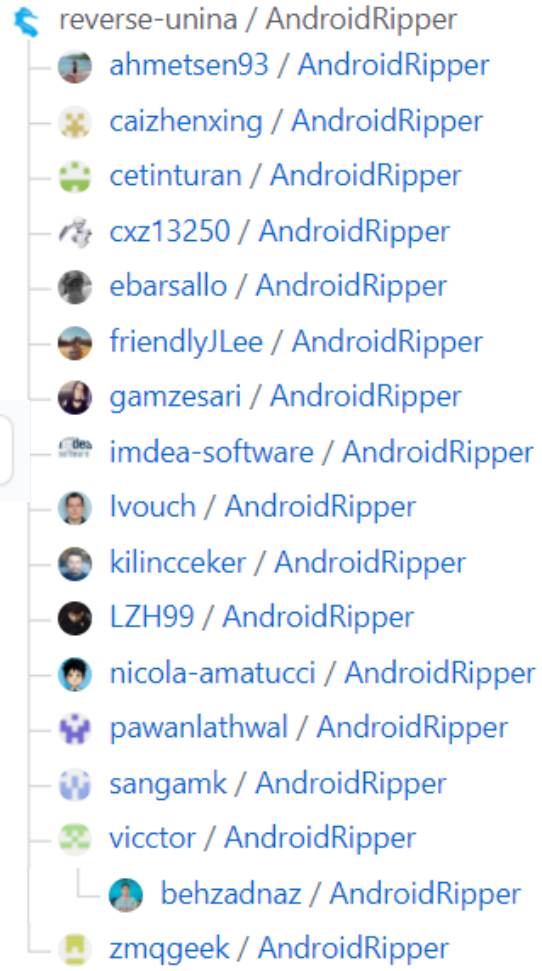
NAVIGAZIONE NEL CODICE

- Si possono individuare tutte le versioni del software, identificate come:
  - Commit
    - Ad ogni operazione di commit su github è associato un titolo: cliccando su questo titolo si possono vedere le modifiche introdotte da quell commit
  - Tag
    - Ad uno specifico commit può essere associato un Tag (etichetta): si individua così una release. Cliccando sulla release si può avere lo stato del Progetto intero al tempo di quella release
  - Branch
    - Il Progetto può avere delle ramificazioni (branch) identificate da un nome: ogni ramificazione procede da quell momento in maniera indipendente e selezionandola si può vedere un ramo di Progetto diverso

FORK DI UN PROGETTO ESISTENTE

- Una fondamentale opportunità fornita da github è quella di creare una fork di un progetto
  - Basta premere l'apposite pulsante
  - Github crea una copia esatta del progetto scelto e la aggiunge ai progetti di chi ha fatto la fork
    - Si può così fare proprio lo stato del progetto e iniziare a modificarlo dal proprio profilo come si preferisce
    - Viene mantenuto un conteggio delle fork di ogni progetto





ALTRE INTERAZIONI CON I PROGETTI

- Watch
  - Selezionando questo pulsante notifiche sulle attività di questo progetto saranno incluse nella nostra Dashboard
- Star
  - Selezionando questo pulsante aggiungiamo questo progetto ai nostri preferiti, intendendo quindi dare un feedback positivo agli sviluppatori

Git

- All'operazione di **checkout** (copia dell'immagine attuale del progetto in locale) corrisponde l'operazione di **Clone** (copia di tutto il progetto in locale)
- Si può pubblicare la propria copia locale, che diventa un nuovo progetto Git di proprietà dell'utente stesso. Tale nuovo progetto è da considerarsi concettualmente come un **Branch** del progetto originale
- L'operazione di **Commit** diventa un'operazione locale, quindi quasi immediata e senza alcuna necessità di review
- L'operazione di **Patching**, invece, corrisponde ad una richiesta di Commit indirizzata verso il progetto Master
  - Se il proprietario del master approva, viene creata una *patch*, cioè un insieme di operazioni per trasformare la copia master in una copia che integri anche le modifiche dell'utente

COMMIT & PUSH

- Commit è un'operazione che aggiorna *localmente* il nostro Progetto supportato da git (in generale va a modificare file .git)
- Push è un'operazione che invia al repository remote (ad esempio ospitato da Github) una richiesta di aggiornamento di tutti I file modificati in locale
  - Solo un proprietario (owner) del progetto può fare richiesta di push: tutti gli altri al Massimo possono fare una *pull request*
  - La richiesta di Push viene analizzata in cerca di possibili conflitti tra la versione precedente e quella successive alla modifica
    - I conflitti sono risolti in generale come in cvs, con la possibilità di scartare modificare, fare merge manuali o automatici, creando branch

Differenze con CVS

Il sistema Git:

- È più sicuro: non esiste un'unica copia centralizzata del progetto
- E' più scalabile: il numero di partecipanti al progetto può aumentare liberamente
  - Invece in CVS e SVN, ad esempio, aumentando il numero di partecipanti aumenta il tempo in cui le risorse sono bloccate e/o il numero di conflitti sui commit
    - Il sistema Linux è stato sviluppato storicamente sotto Git, con il master sotto il diretto controllo di Linus Torvalds
- Necessita di opportune licenze di utilizzo
  - Spesso è legato a software open source distribuito con licenze Creative Commons
  - Non è generalmente utilizzato all'interno di aziende che realizzano software closed source

PULL REQUEST

- La Pull Request è un elemento fondamentale della visione open source di Github
- Un qualsiasi sviluppatore, anche esterno al progetto, può realizzare un potenziale miglioramento dell'applicazione e proporre al proprietario di inglobarla nel progetto tramite una Pull Request
  - In questo caso la Pull Request parte da un fork del progetto sul quale sono stati fatti dei commit successivi
  - Una Pull Request può anche essere fatta a partire da un branch del progetto stesso

ISSUE

- Si può contribuire anche da esterni ad un progetto altrui facendo notare difetti, problemi o semplicemente proponendo evoluzioni, tramite le Issue