Source Control Management (SCM)

- Detto anche **VCS**: Version Control System
- Obiettivi
  - Mantenere traccia dei cambiamenti nel codice; sincronizzazione del codice tra utenti
  - Cambiamenti di prova senza perdere il codice originale; tornare a versioni precedenti
- **Centralized** → architettura client/server (CVS, Subversion, ...)
  - Repository centralizzato con le informazioni del progetto
    - (codice sorgente, risorse, configurazioni, documentazione, …)
  - check-out/check-in (lock del file), branch/merge (conflitti)
- **Distributed** → architettura peer-to-peer (**Git**, Mercurial, ...)
  - Repository clonato su tutte le macchine
  - Gran parte del lavoro può essere fatto offline

Modelli di Versioning

- Naive
  - A e B modificano lo stesso file allo stesso tempo; A salva i cambiamenti nel repository; subito dopo B salva i suoi; nascondendo i cambiamenti di A.
- Lock – modify – unlock
  - Il file può essere cambiato solo da un utente per volta.
  - Semplice, ma ha una serie di problemi:
    - Lock dimenticati; serializzazione anche quando non è necessario; gestione dipendenze in altri file …
    - Era usato nei vecchi VCS centralizzati
- Copy – modify – merge
  - Si lavora su copie locali, poi si fa il merge con la copia del repository.
  - Necessita una accurata gestione dei conflitti
    - È il modello comunemente usato

Git

- 2005 by Linus Torvalds et al.
  - Pensato per lo sviluppo di Linux
- Supportato nei principali ambienti di sviluppo
  - Non è necessario installare un client specifico (ma è comodo per un uso avanzato)
- Client ufficiale
  - Scaricabile da https://git-scm.com/
  - Disponibile per i principali sistemi operativi
  - Codice sorgente: https://github.com/git/git
- Siti su cui condividere un repository
  - **github.com**, gitlab.com, bitbucket.org, …

Configurazione di Git

- Vince il più specifico tra
  - Sistema: Nel folder di installazione del programma
    - *(sconsigliato, non entriamo nei dettagli)*
  - Globale: Nel folder dell'utente corrente, file **.gitconfig**
  - Locale: Nel folder del progetto corrente, file **.git/config**
- Esempio: set globale del nome e dell'email dalla shell di Git
  - git **config** --global user.name "Emanuele Galli"
  - git **config** --global user.email egalli64@gmail.com
- Per verificare la configurazione corrente
  - git **config** --list

Repository locale – area di lavoro

- Eseguendo il comando git **init** in una directory
  - Si crea una directory **.git** → un nuovo repository Git locale
  - La directory corrente diventa l'area di lavoro del nuovo repository
- Un file nell'area di lavoro può essere, rispetto al repository
  - Sconosciuto (untracked)
  - Modificato – cambiato rispetto all'ultima versione
  - Indicizzato (staged) – segnalato a git per il prossimo commit
  - Committato – nello stato corrente
- Per verificare lo stato corrente dell'area di lavoro: git **status**

Add e commit

- Dato un file nell'area di lavoro, sconosciuto a git o modificato
  - Si aggiunge la prima (o una nuova) versione del file nell'indice: git **add** <filename>
  - Si aggiorna il branch corrente del repository locale: git **commit -m** <message>
    - <message>: un messaggio tra doppi apici che spieghi perché si cambia il repository, es: "fix ticket #42"
  - Si può fondere *add e commit* in un unico comando: git **commit -am** <message>
    - Aggiunge all'indice e commita tutti i file modificati – solo quelli già noti a Git!
- Se vogliamo controllare i cambiamenti nell'area di lavoro prima di commitare: git **diff**
  - Con l'opzione --staged si opera sull'indice (area di staging)
- Il comando git **log** mostra lo storico dei commit, con relativi messaggi
  - git **log -p** (patch) mostra i cambiamenti rispetto al commit precedente
  - HEAD indica qual è la versione a cui fa correntemente riferimento l'area di lavoro

Da locale a remoto (GitHub)

- Un utente registrato e loggato su GitHub
  - Può creare un repository remoto da questa pagina → https://github.com/new
    - Occorre specificare il repository name, ad es: xyz
    - Bottone “create repository” → nel mio caso, viene creato qui: https://github.com/egalli64/xyz
- Condivisione di un nostro repository locale
  - git **remote add** origin <Git URL> → ad es: https://github.com/egalli64/xyz.git
    - Si usa il nome “origin” (convenzionale) come riferimento all’URL del repository remoto
  - git **push -u** origin master
    - Upload dei file da locale a remoto, sul branch “master” (convenzionale, in transizione a “main”)
    - L’opzione -u (o --set-upstream) va usata solo la prima volta, poi sarà implicito il branch corrente
- Verifica del repository remoto associato via git **remote -v**
  - Se si vogliono informazioni più dettagliate: git **remote show** origin

Da remoto (GitHub) a locale

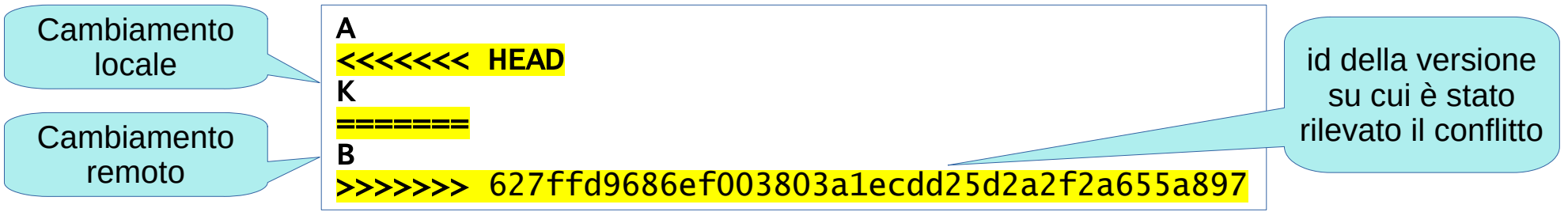
- Se un repository è pubblico tutti ne possono fare il “fork”
  - Bottone “Fork” sulla destra della pagina del repository
    - Crea una nostra copia indipendente
- Occorre avere il permesso per modificare un repository
  - Ristretto al proprietario e gli utenti da lui invitati a partecipare
- Clonazione in locale di un repository via git **clone** <Git URL>
  - GitHub riporta l’URL HTTPS del repository nel tab “Code”, pulsante verde “Code”
  - Crea una directory con il nome del repository sotto quella corrente
    - Di solito mettiamo i repository nella cartella git nella home del nostro utente
  - Al suo interno c’è la cartella **.git** con la copia (clone) del repository

Pull e push

- Aggiornamento del branch corrente locale (conviene farlo spesso)
  - git **pull** – abbreviazione di due comandi
    - git fetch – download dei cambiamenti del branch corrente da remoto (origin) a locale
    - git merge – fusione, quando necessario, nei file locali dei cambiamenti rilevati
- Per aggiornare il branch corrente su origin
  - dopo aver eseguito **commit** in locale
  - git **push**
- Ma, se nel frattempo origin è cambiato, git ci blocca!
  - Dobbiamo fare **prima** una pull, verificare che non ci siano conflitti, nel caso risolverli
  - E solo dopo (e se solo se adesso è tutto a posto) eseguire una push

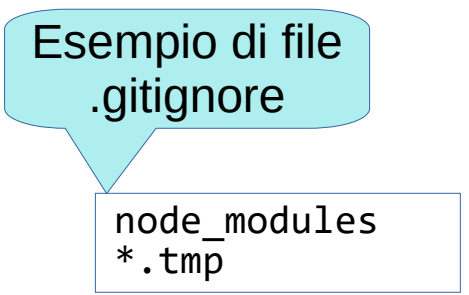
Conflitti su pull

- Il file hello.txt ha una sola riga: “**A**”
- L’utente X aggiunge una riga “**K**” e committa
- L’utente Y fa una pull, aggiunge la riga “**B**”, committa e fa un push
- Ora, il pull di X causa un auto-merging di hello.txt con un conflitto
  - Git chiede aiuto solo se non riesce a decidere cosa fare → conflitto sulla stessa riga
- Va risolto editando il file + add/commit del risultato



File ignorati da Git

- File che ***non*** vogliamo mettere nel repository
  - File di configurazione di Eclipse e altri tool
  - File generati dal compilatore
  - …
- Nel file di testo semplice "**.gitignore**"
  - Su ogni riga possiamo mettere
    - Nome di un file
    - Nome di un folder
    - Un pattern



Checkout e reset

- File modificato, vogliamo tornare alla versione nel repository
  - git **checkout** <filename>
- Se si vuole portare nell'area di lavoro una versione specifica
  - git **checkout** <version> **--** <filename>
    - Se non si specifica la versione, si intende la più recente
- Per togliere dall'indice un file che non va committato
  - git **reset** <filename>
- Per mettere nell'area di lavoro una versione specifica dell'intero progetto
  - git **reset --hard** <version>
  - Per vedere il log completo, dopo un reset hard → git **log @{1}**

Altri comandi

- Eliminare / rinominare un file
  - git rm <filename>
  - git mv <filename> <newname>
- Confronto di un file nell’area di lavoro con una versione nel repository
  - git diff <version> <filename>

Branching del repository

- Lista dei branch locali, evidenziando quello corrente: git **branch**
  - Lista dei branch remoti: git branch -r
- Creazione un nuovo branch: git **branch** <name>
  - Copia del branch principale, usato per lo sviluppo in parallelo di una nuova feature
  - Il primo push del nuovo branch deve creare un upstream branch: git **push -u** origin <name>
- Scelta del branch corrente: git **checkout** <name>
  - Creazione di un nuovo branch e sua selezione: git **checkout -b** <name>
- Versione estesa del logging: git **log --graph --oneline --all**
- Fusione del branch corrente con quello indicato: git **merge** <name>
  - git **rebase** <name> può aiutare a semplificare il lavoro (ma va fatto con maggiore attenzione)
- Eliminazione di un branch
  - Locale: git **branch -d** <name>
  - Remoto: git **push --delete** origin <name>

Suggerimenti

- Inizio giornata, dopo una pausa, prima di iniziare un nuovo lavoro: pull
- Meglio fare tanti commit e push, ognuno focalizzato su un singolo cambiamento piuttosto, che uno solo riassuntivo
- È importante scrivere messaggi di commit significativi
- Conviene lavorare su un branch a parte se il cambiamento a cui si lavora implica un lavoro lungo e complesso
  - Ricordarsi di fare spesso un merge nel branch corrente dal principale
- Una strategia usata per semplificare lo sviluppo è tenere l'ultima versione stabile del codice su un branch a parte, e il codice in sviluppo sul master