

# Version Control System (VCS)

- Motivazioni all'uso del versioning
- VCS vs DVCS
- Subversion
- Git

# (Distributed) 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
- 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 VCS, architettura peer-to-peer (Git, Mercurial, ...)
  - Repository clonato su tutte le macchine
  - Solo push e pull richiedono connessione di rete

# Apache Subversion

- 2000 by Karl Fogel et al. @ CollabNet
  - Dal 2010 gestito dalla Apache Software Foundation
  - <https://subversion.apache.org/>
- Nato per superare le limitazioni di CVS
- Sistema di controllo versioni con un unico repository centralizzato
- TortoiseSVN è un client grafico (e CLI) molto usato in ambiente Windows

# 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: unlock dimenticati; serializzazione anche quando non è necessario; gestione dipendenze in altri file
- Copy – modify – merge
  - Si lavora su copie locali, poi si fa il merge con la copia del repository.
  - Necessita una accurata gestione dei conflitti

# L'uso di SVN

- svn help
- Creazione di una copia locale da un repo
  - svn checkout
  - Directory .svn → informazioni specifiche del folder
- Aggiungere un folder a un repo
  - svn import
- Informazioni sulla storia del repository
  - svn log, svn list

# L'uso comune di SVN

- Aggiornamento della copia locale
  - svn **update**
- Modifiche alla struttura delle directory locali
  - svn **add**, svn delete, svn copy, svn move
- Verifica dei cambiamenti e possibile loro annullamento
  - svn **status** (**C** → conflitto, **G** → merged), svn diff, svn revert
- Verifica di possibili conflitti, loro soluzione, pubblicazione dei cambiamenti
  - svn update, svn resolve, svn **commit -m**

# Git

- 2005 by Linus Torvalds et al.
- 24 febbraio 2019: version 2.21
- Supportato nei principali ambienti di sviluppo
- Client ufficiale
  - <https://git-scm.com/> (SCM: Source Control Management)
- Siti su cui condividere pubblicamente un repository
  - [github.com](https://github.com), [bitbucket.org](https://bitbucket.org), ...
- Gli utenti registrati possono fare il [fork](#) di repository pubblici

# Configurazione di Git

- Vince il più specifico tra
  - Sistema: Programmi Git/mingw64/etc/gitconfig
  - Globale: Utente corrente .gitconfig
  - Locale: Repo corrente .git/config
- 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`



# Nuovo repository Git locale

- Prima si crea il repository remoto → URL .git
- A partire da quella URL, copia locale del repository
  - Esempio: <https://github.com/egalli64/empty.git>
- Shell di Git, nella directory git locale:
  - `git clone <URL>`
- Possiamo clonare ogni repository pubblico
- Per il push dobbiamo avere il permesso

# Creare un file nel repository

Dalla shell di Git, nella directory del progetto

Crea il file hello.txt

Aggiorna la versione  
nel repository locale  
sincronizzandola  
con la copia di lavoro

```
echo "hello" > hello.txt  
git add hello.txt  
git commit -m "first commit"  
git push -u origin master
```

I cambiamenti nel file  
andranno nel repository

Aggiorna la versione  
nel repository remoto  
sincronizzandola  
con quella in locale

# File ignorati da Git

- Alcuni file devono restare locali
  - Configurazione
  - File compilati
- Per ignorare file o folder
  - Creare un file “.gitignore”
  - Inserire il nome del file, pattern o folder su una riga

Esempio di file  
.gitignore

```
node_modules  
*.tmp
```

# git pull

- Per assicurarsi di lavorare sul codebase corrente, occorre sincronizzarsi col repository remoto via pull
- È in realtà la comune abbreviazione dei comandi fetch + merge origin/master

# Cambiamenti nel repository

- Se vogliamo che un nuovo file, o che un edit, venga registrato nel repository, dobbiamo segnalarlo col comando `git add`
- A ogni commit va associato un `messaggio`, che dovrebbe descrivere il lavoro compiuto  
`git commit -m ".classpath is only local"`
- Per l'editing, si può fondere add con commit, usando l'opzione "a"  
`git commit -am "hello"`
- La prima commit crea il branch "master", le successive aggiornano il branch corrente

# git push

- Commit aggiorna il repository locale
- Push aggiorna il repository remoto
- Per ridurre il rischio di conflitti, **prima pull**, dopo (e solo se non sono stati rilevati problemi) 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 di risolverlo editando il file + **add/commit** del risultato

Cambiamento  
locale

Cambiamento  
remoto

```
A
<<<<<<< HEAD
K
=====
B
>>>>>>> 627ffd9686ef003803a1ecdd25d2a2f2a655a897
```

id della commit  
con conflitto

# Branching del repository

- `git branch`
  - Lista dei branch esistenti, evidenzia quello corrente
- `git branch <name>`
  - Crea un nuovo branch
  - Il primo push del nuovo branch deve creare un upstream branch
    - `git push --set-upstream origin <name>`
- `git checkout <name>`
  - Permette di scegliere il branch corrente
- `git merge <name>`
  - Eseguito dal branch principale, fusione con risoluzione degli eventuali conflitti



# Principali comandi Git in breve

- clone <url>: clona un repository in locale
- add <filename(s)>: stage per commit
- commit -m "message": copia sul repository locale
- commit -am "message": add & commit
- status: lo stato del repository locale
- push: da locale a remoto
  - push --set-upstream origin <branch>
- pull: da remoto a locale
- log: storico delle commit
- reflog: storico in breve
- reset --hard <commit>: il repository locale torna alla situazione del commit specificato
- branch: lista dei branch correnti
- branch <branch>: creazione di un nuovo ramo di sviluppo
- checkout <branch>: scelta del branch corrente
- checkout <ref> -- <filename>: revert di file
- merge <branch>: fusione del branch