

Versioning / DevOps

- Versioning del codice sorgente
 - Version Control System (VCS) / Source Control Management (SCM)
 - Subversion
 - Git
- DevOps: sviluppo software + operazioni IT
 - CI / CD: Continuous Integration / Continuous Delivery
 - Jenkins

(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.
- Supportato nei principali ambienti di sviluppo
- Client ufficiale
 - <https://git-scm.com/>
 - 19 aprile 2020: versione 2.26.2
- Siti su cui condividere pubblicamente un repository
 - github.com, gitlab.com, 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

- Dato un repository remoto → URL `.git` (già esistente o ancora vuoto)
 - Esempio: `https://github.com/egalli64/empty.git`
- Clonazione in una directory git locale della nostra macchina:
 - `git clone` <URL>
 - Possiamo clonare ogni repository pubblico
- Per condividere un nostro progetto in quel repository (vuoto)
 - `git init` nella root del progetto
 - Commit locale dei file (vedi slide successive)
 - `git remote add origin` <URL>
- Possiamo fare il push su un repository solo se ne abbiamo i diritti

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

CI / CD

- Unit test verifica una singola unità di sviluppo
 - Occorre evitare che cambiamenti locali rendano instabili altre unità
- Periodicamente, sull'intero codebase
 - **Continuous Integration** (build, deploy, test)
 - Compilazione
 - Esecuzione di test automatici
 - **Continuous Delivery**
 - Il risultato della compilazione deve essere consegnabile all'utente

Jenkins

- <https://www.jenkins.io/download/>
 - È una Web App Java distribuita come Web Archive (war)
 - Include l'Application Server (Jetty) che lo contiene
 - Per AS alternativi: <https://wiki.jenkins.io/display/JENKINS/Containers>
- Può eseguire automaticamente, tra l'altro
 - Build del progetto
 - Test (Unit, Performance, Integration)
 - Rilascio dell'applicativo
 - Notifiche

Standalone – Jetty

- Shell dei comandi, nel nostro folder Jenkins
 - [set JAVA_HOME=...\Java\jdk-xxx] -- se necessario
 - [set JENKINS_HOME=...] -- default: directory .jenkins dell'utente corrente
 - java -jar jenkins.war [--httpPort=nnnn] -- default: usa la porta 8080
- La password dell'amministratore viene generata automaticamente
 - Nella home di jenkins, secrets/initialAdminPassword
- Per non usare la localizzazione italiana
 - Installare il plugin: Languages – Locale
 - Manage Jenkins – Configure System
 - Locale – Default language: en_US
 - Ignore browser preference and force this language to all users

CLI

- /cli
 - download `jenkins-cli.jar`
 - `/jnlpJars/jenkins-cli.jar`
- /me/configure
 - Generazione di un `API token`, ad es: `114250ef9bfb9088ffe70fcc241da5dc08`
- Shell dei comandi, nel nostro folder Jenkins
 - `SET JENKINS_USER_ID=user`
 - `SET JENKINS_API_TOKEN=114250ef9bfb9088ffe70fcc241da5dc08`
 - `SET JENKINS_URL=http://...` [in alternativa, eseguire `jenkins-cli` con l'opzione `-s http:...`]
 - `java -jar jenkins-cli.jar -webSocket`
- <https://www.jenkins.io/doc/book/managing/cli/>

Job

- /newJob - Creazione di un nuovo job (item/progetto)
 - Va specificato il **nome** e il tipo di progetto (Freestyle)
- /job/**nome**/configure
 - Connessione a SCM *[gestite da appositi plugin ...]*
 - Build trigger
 - Da remoto, fornendo un token di autenticazione
 - Periodicamente, cron-style o via shortcut come @midnight
 - Build
 - Lista di script da eseguire
 - Post-build
 - Azioni standard da eseguire dopo una build
- /job/**nome** – informazioni sul job

Integrazione con Git

- /pluginManager/available – git
 - <https://plugins.jenkins.io/git/>
- /job/**nome**/configure
 - Source Code Management
 - Repository URL, ad es: <https://github.com/egalli64/hello.git>
 - Build Triggers → Poll SCM
- Esecuzione del job:
 - clone/pull del repository nel folder **workspace** nella home di Jenkins
 - Esecuzione dei comandi indicati in Build e Post-build

Delivery Pipeline

- Esecuzione di più job in serie
- /job/**stepY**/configure: come successore o predecessore
 - Build triggers – Build after other projects are built → Projects to watch: **stepX**
 - Add post-build Actions – Build other project → Projects to build: **stepZ**
- Plugin per semplificare la gestione di pipeline
 - <https://plugins.jenkins.io/delivery-pipeline-plugin/>
 - Nella home page di Jenkins, oltre a “All”
 - Nuovo tab (“+”) di tipo Delivery Pipeline **View**
 - Enable start of new pipeline build, Enable rebuild
 - Pipelines → Components → initial Job

Deploy to container

- Plugin per deploy di applicazioni JavaEE
 - <https://plugins.jenkins.io/deploy/>
- /job/example/configure
 - Post-build Actions – Deploy war/ear to a container
 - WAR/EAR files: [ex: sample.war]
 - Containers: [ex: Tomcat 9]
 - Credential: [per Tomcat, vedi sotto]
 - Tomcat URL: [ex: <http://localhost:8080>]
- Configurazione di Tomcat, tomcat-users.xml
 - `<user username="jenkins" password="..." roles="manager-script"/>`
- Il container deve essere accessibile e in esecuzione durante la build

Role Strategy Plugin

- Role-based Authorization Strategy
 - Gestione dei permessi per utente
 - <https://plugins.jenkins.io/role-strategy/>
- /configureSecurity – Configure Global Security
 - Authorization: Role-Based Strategy
- /role-strategy – Manage and Assign Roles
 - Manage: Global role overall – read richiesto per login
 - Assign: ogni utente/gruppo può avere un ruolo specifico
- Andrebbe abbinato a un plugin per la configurazione delle autorizzazioni per progetto
 - <https://plugins.jenkins.io/authorize-project/>