Repository di Progetto

Un repository **GIT** di progetto può essere visto come una cartella che tiene traccia delle modifiche effettuate nel tempo a tutti i file in essa contenuti.

I repository di prodotto si trovano su gitlab GEOWEB nell'apposito gruppo dedicato ai progetti a cui è possibile accedere effettuando il login con le proprie credenziali gitlab.

Tramite GIT è possibile tenere traccia di tutte le modifiche effettuate al proprio repository, per interagire con esso potrebbe essere utile scaricare e installare **Fork**, per cui si rimanda all'apposita sezione con la guida di installazione.

Prima di procedere verificare di avere tutti i prerequisiti necessari alla gestione dei repository di progetto.

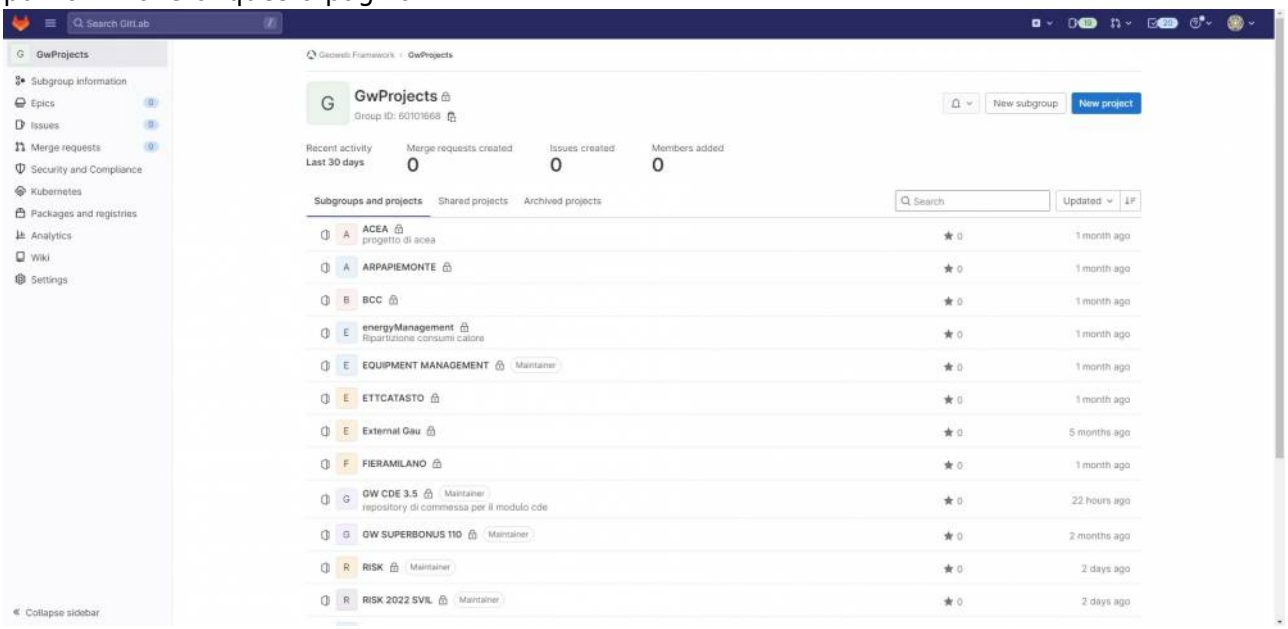
N.B La seguente guida è personalizzata per l'utilizzo del software Fork, per questo se ne consiglia l'utilizzo, sebbene possa essere sostituito da linea di comando con i comandi GIT

Creazione del repository remoto

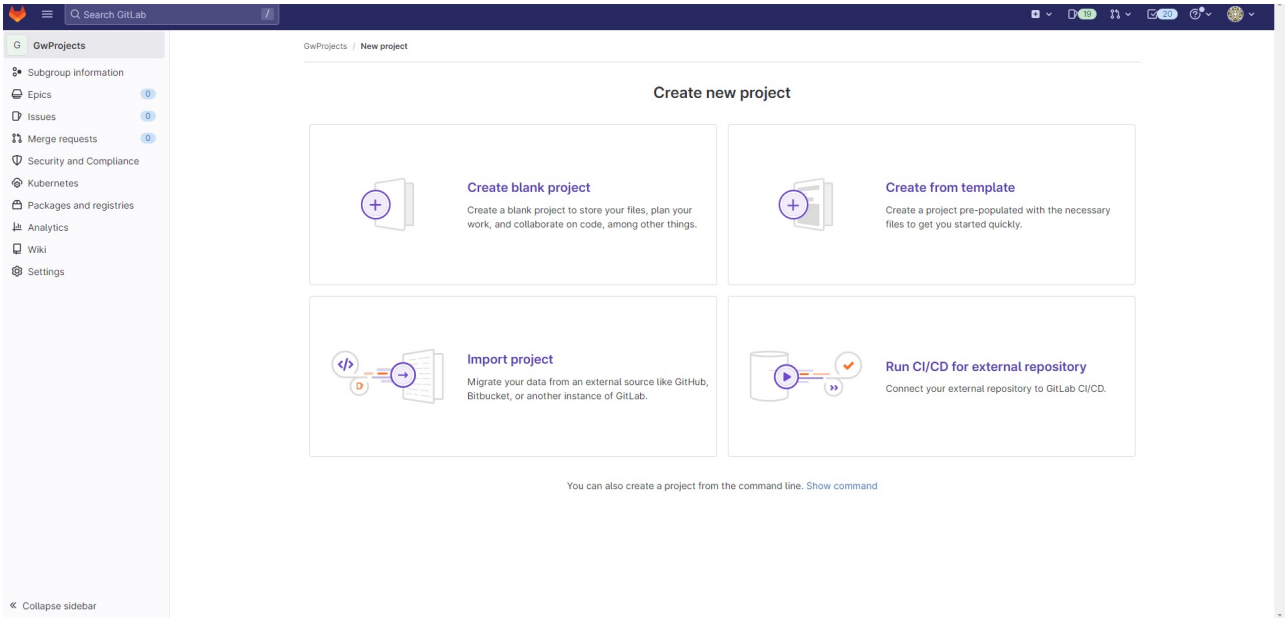
La prima cosa da fare quando si vuole creare un repository di progetto è la creazione di un nuovo progetto su Gitlab. Il nuovo repository **deve** essere creato nell'apposita sezione dedicata ai **PROGETTI** disponibile al link https://gitlab.com/geowebframework/gwprojects.

Per creare il repository remoto del progetto a cui stai lavorando, procedi come segue:

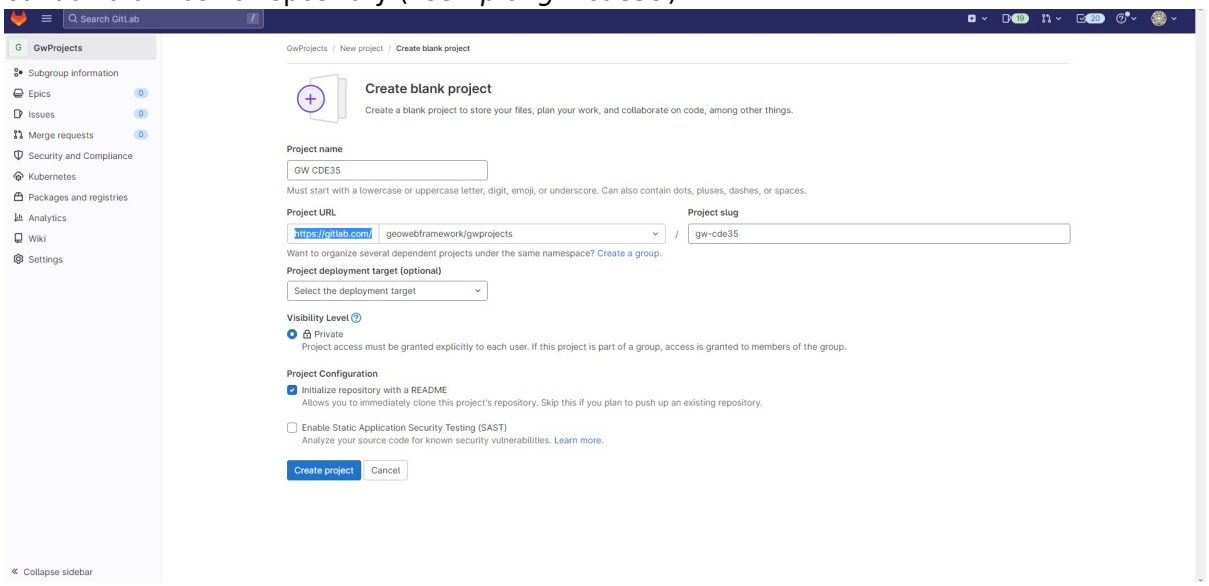
1. Vai su gitlab ed esegui il login https://gitlab.com/users/sign_in
2. Apri la sezione riservata ai progetti: https://gitlab.com/geowebframework/gwprojects. A questo punto ti troverai questa pagina



3. Cliccare su **New project**
4. Scegliere **Create blank project** e compilare il modulo con i dettagli del proprio prodotto:



- **Project Name**: è il nome del repository di prodotto (*Esempio: GW CDE35*)
- **Project SLUG**: oltre alla parte relativa all'url principale, scelta di default (https://gitlab.com/geowebframework/gwprojects/), scegliere il percorso specifico che condurrà al nostro repository (*Esempio: gw-cde35*).



5. Cliccare su **Create Project**

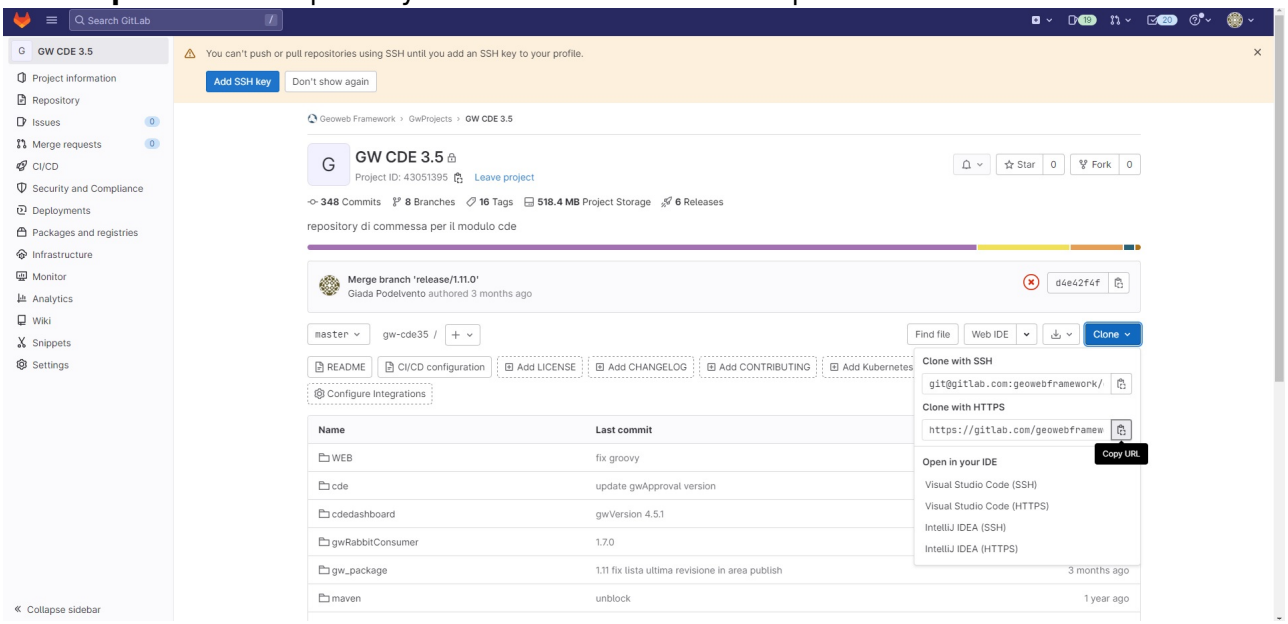
Una volta salvate tutte le informazioni, il progetto GIT (che rappresenta il nuovo repository remoto di prodotto) viene infine creato; esso è consultabile aprendo la sua pagina di dettaglio sul Gitlab (quindi da progetti cercando in lista il nome del progetto appena creato).

Creazione del repository locale

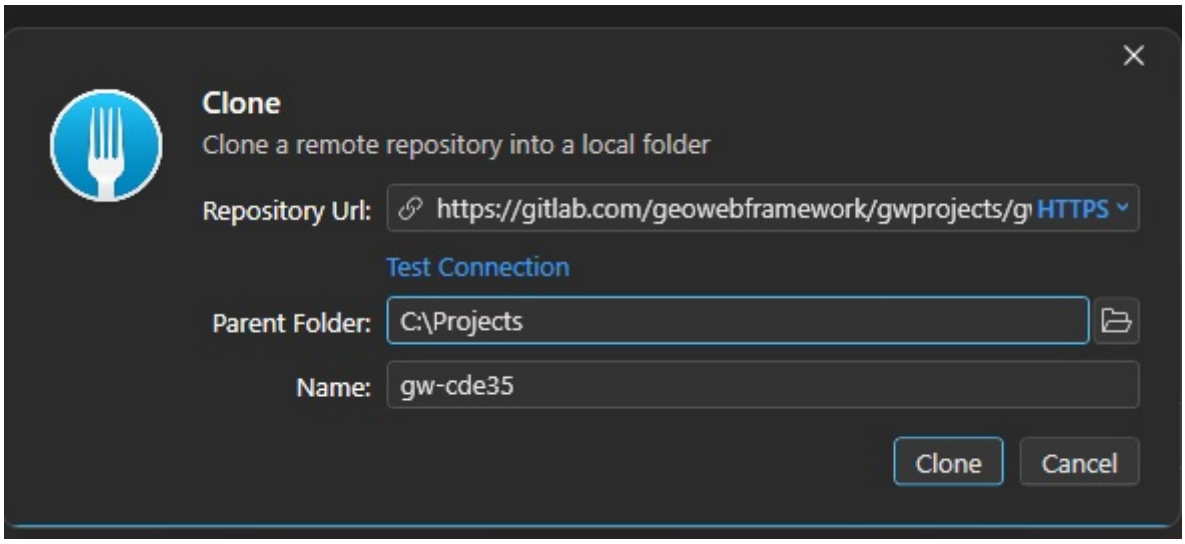
Una volta creato il repository remoto (vedi paragrafo precedente) è possibile copiarlo nel proprio PC o nel server di progetto. Prima di procedere assicurarsi di avere **GIT** (guida installazione git) e **Fork** installati (guida installazione Fork) sulla macchina dove si vuole creare il repository locale (il proprio PC o il server).

Per creare il repository locale segui i seguenti passaggi:

1. Vai su gitlab ed esegui il login https://gitlab.com/users/sign_in
2. Apri la sezione riservata ai progetti: https://gitlab.com/geowebframework/gwprojects
3. Dalla lista dei repository presenti, selezionare il progetto GIT corrispondente al progetto desiderato
4. Dalla maschera principale del progetto GIT selezionare **Clone**: si aprirà il menù a tendina con il **link https** riferito al repository selezionato come nell'esempio



5. Dal menu a tendina copiare il **link https**
6. Aprire il software **Fork** dal server di sviluppo (se si desira copiare il prodotto in questo server) altrimenti aprire Fork sul proprio PC (se si desidera copiare il progetto sulla propria macchina). **N.B** Se il server di sviluppo è raggiungibile dal proprio PC, può essere usato il Fork del proprio PC, basterà inserire successivamente il percorso con il puntamento al server. (Vedi passo 5)
7. Da Fork cliccare su **File → Clone**



- **Repository Url**: è l'url del repository remoto. Incollare il link appena copiato
- **Parent Folder**: è la cartella su cui sarà creata la copia del repository selezionato (*Esempio: C:\project* se l'applicativo Fork e il repository locale si trovano nella stessa macchina, altrimenti se si desidera copiare il prodotto nel server di sviluppo usando Fork installato nel proprio PC, assicurarsi prima che il server sia raggiungibile da questo e scrivere il path corrsipondente al server specificato *Esempio: \\{ip server}\c$\Projects*)
- **Name**: è il nome del progetto GIT (per comodità si consiglia di scegliere lo stesso nome del repository remoto)

- Cliccare su **Clone** e attendere il termine del processo

In questo modo si è creato il repository locale, raggiungibile dal percorso scelto in fase di clonazione. In prima creazione la cartella creata conterrà una sola sottodirectory già popolata chiamata "**.git**", contenente l'intero repository e la sua struttura, con tutte le informazioni necessarie a GIT, lo storico dei messaggi associati ai commit, e la cartella logs che registra una cronologia degli interventi eseguiti sui vai rami di sviluppo. La magia avviene qui dentro, quindi non toccarla!

Una volta creato il repository locale si può procedere alla creazione della struttura base delle sottocartelle.

Inizializzazione del repository

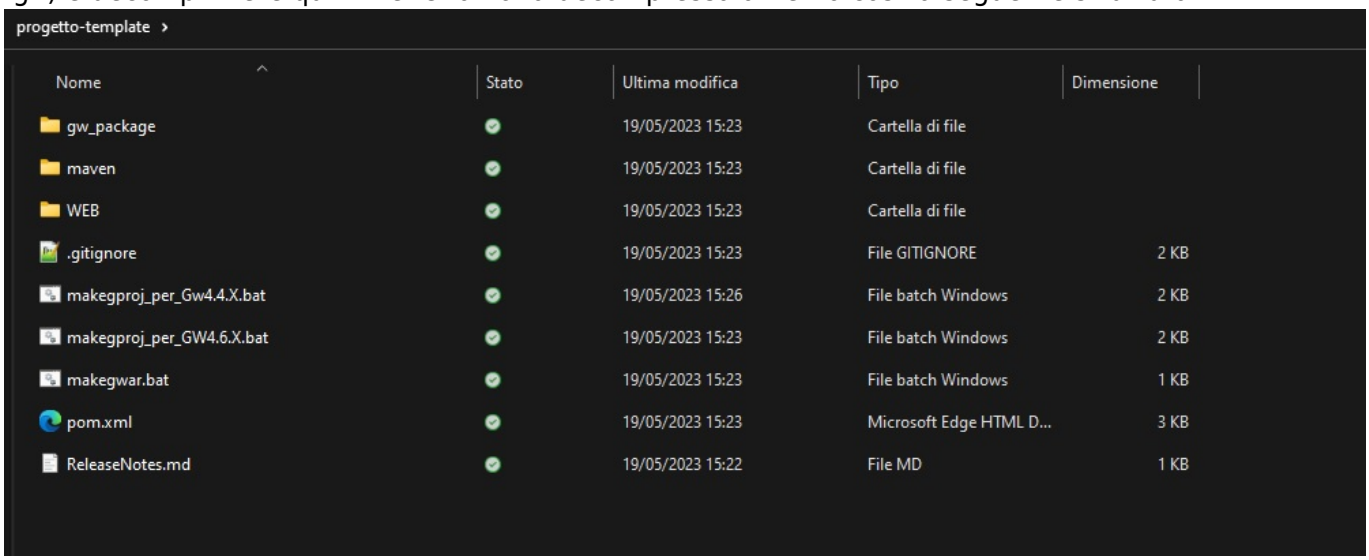
Creazione struttura cartelle di base

Una volta creato il repository remoto e averlo clonato in locale è necessario dotare il progetto di una struttura specifica, adottata per convenzione, in modo tale che le risorse possano essere usate correttamente dai vari file eseguibili messi a disposizione per dispiegarle in maniera semiautomatica.

La prima cosa da fare è cliccare sul seguente

link

e avviare il download della cartella ZIP contenente la struttura iniziale delle cartelle. Al termine del download copiare il file zip nella cartella principale del repository locale (dove è già presente il file .git) e decomprimere qui il file. Una volta decompressa avremo così la seguente struttura



Di seguito riportiamo una breve descrizione di tutte le cartelle qui contenute.

Cartella gw_package

La cartella **gw_package** è necessaria per il tool **GwResourceDeployer**, opportunamente descritto

qui.

Per il corretto funzionamento di GwResourceDeployer, le risorse all'interno di questa cartella devono essere preventivamente preparate secondo una struttura convenzionale, in maniera tale che possano essere successivamente e correttamente dispiegate dal tool. Per la descrizione della struttura seguire la procedura descritta qui

---

Cartella maven

La cartella **maven** contiene il file **gw-repo-settings.xml**. Si tratta di un file di configurazione che contiene le impostazioni Maven già personalizzate per la creazione dei war di progetto. Il file non necessita di modifiche da parte dell'utente.

Cartella WEB

La cartella **WEB** contiene i contenuti statici del progetto opportunatamente raccolti come consueto, distinti nelle rispettive cartelle *download*, *images*, *groovy*, *template*, ...

File .gitinore

Il file **.gitignore** è un file di configurazione utilizzato nel sistema di controllo delle versioni Git. Esso specifica i file e le cartelle che Git dovrebbe ignorare, ovvero di cui non tenere traccia delle modifiche. Ciò è utile per escludere file generati automaticamente, file con dimensioni troppo grandi, file temporanei o file sensibili che non devono essere inclusi nel repository Git. Ne sono un esempio quei file che hanno estensione ".war" o anche ".jar": se un file con questa estensione (ad esempio) fosse presente nel repository locale, verrebbe automaticamente escluso da eventuali commit e quindi non sarebbe condiviso nel repository remoto.

Il file compreso nel pacchetto zip contiene già la lista specifica di quanto necessario ignorare e non richiede ulteriori modifiche da parte dell'utente, a meno di particolari eccezioni.

File makegproj_per_Gw4.4.X.bat e makegproj_per_Gw4.6.X.bat

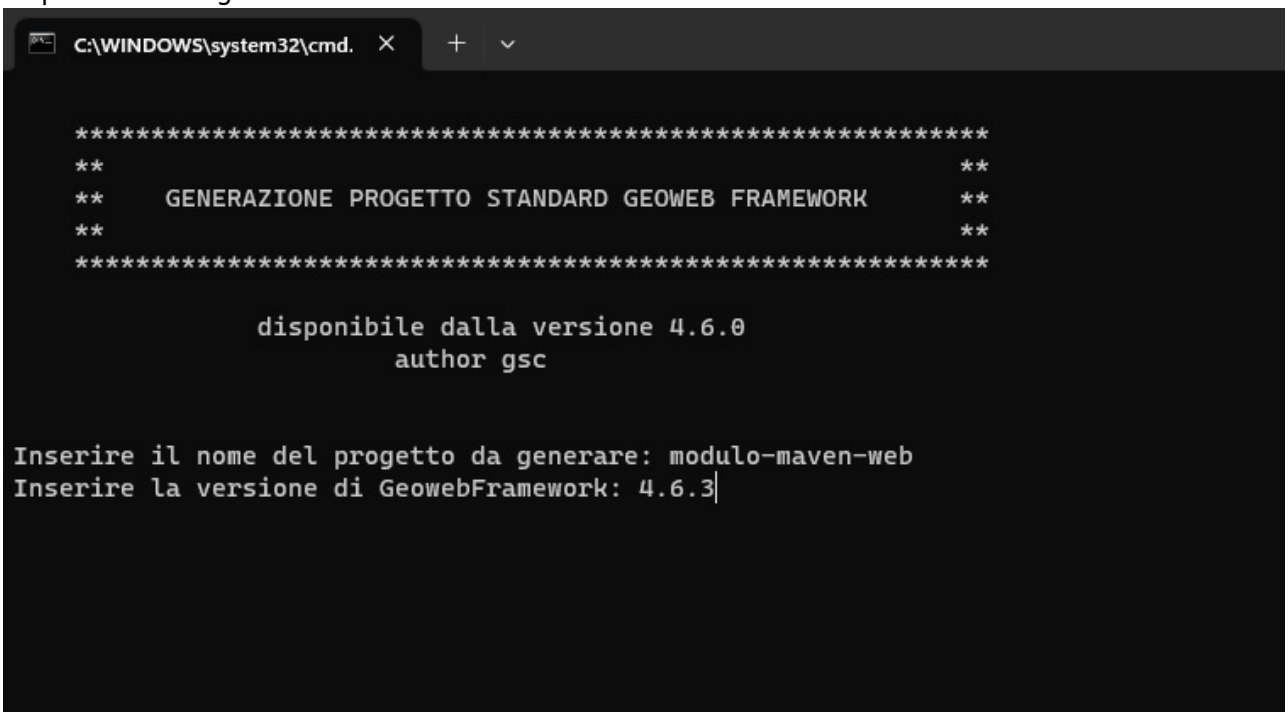
I file **makegproj_per_Gw4.4.X.bat** e **makegproj_per_Gw4.6.X.bat** consentono di creare i moduli maven previsti nel progetto (da cui poi saranno generati i war). Questi devono essere usati in maniera esclusiva (o l'uno o l'altro), a seconda che il repository git si riferisca ad un progetto basato su versione, rispettivamente, 4.4.X o 4.6.X di Geoweb Framework. Essendo la procedura identica in entrambe le casistiche, presentiamo qui una descrizione del procedimento per una versione più aggiornata del framework, quindi 4.6.X.

Prima di procedere assicurarsi di avere Apache Maven installato, altrimenti seguire la procedura di

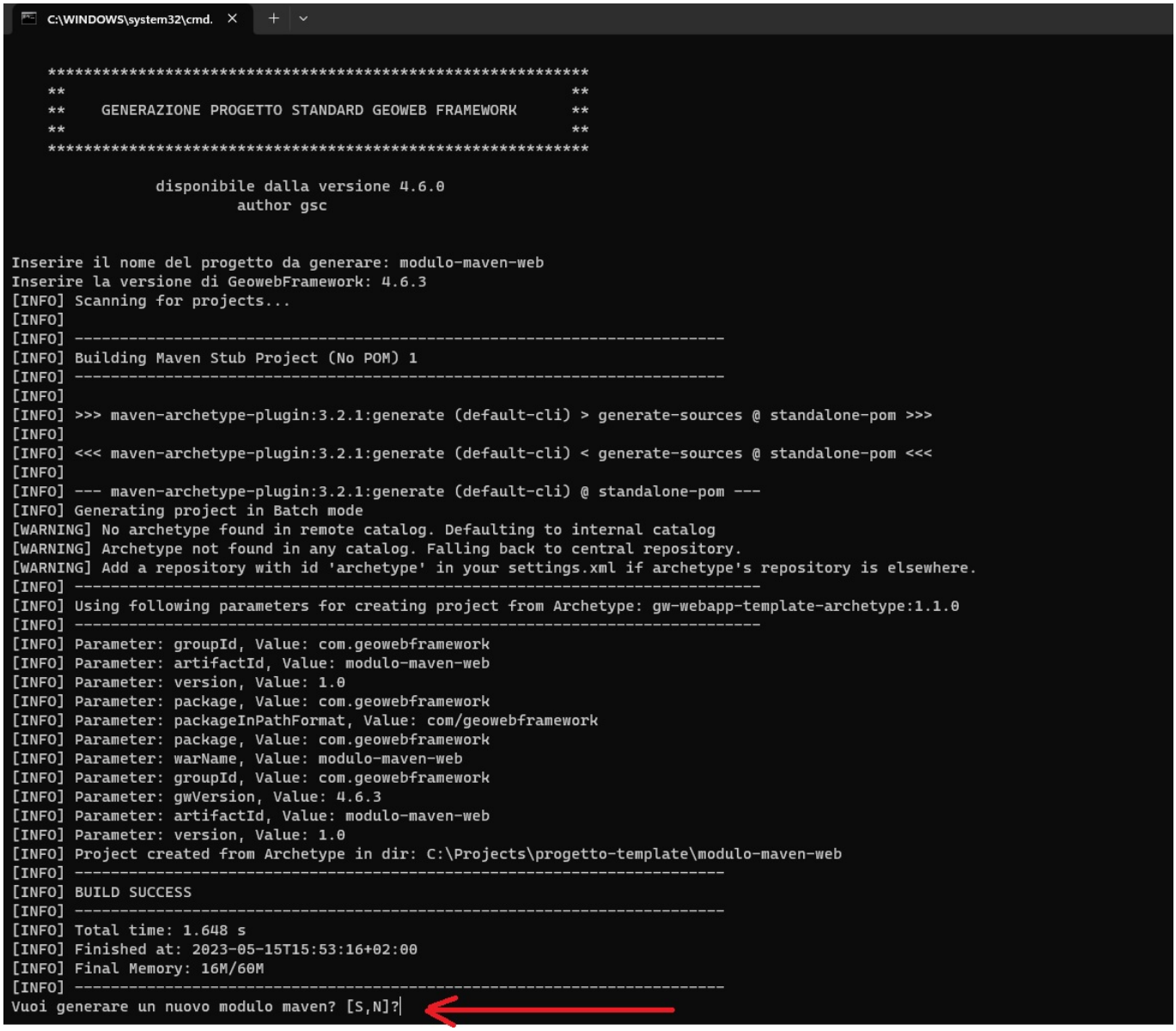
installazione.

Eseguiamo il file **makegproj_per_Gw4.6.X.bat** facendo doppio click su di esso; una volta eseguito, richiederà due parametri:

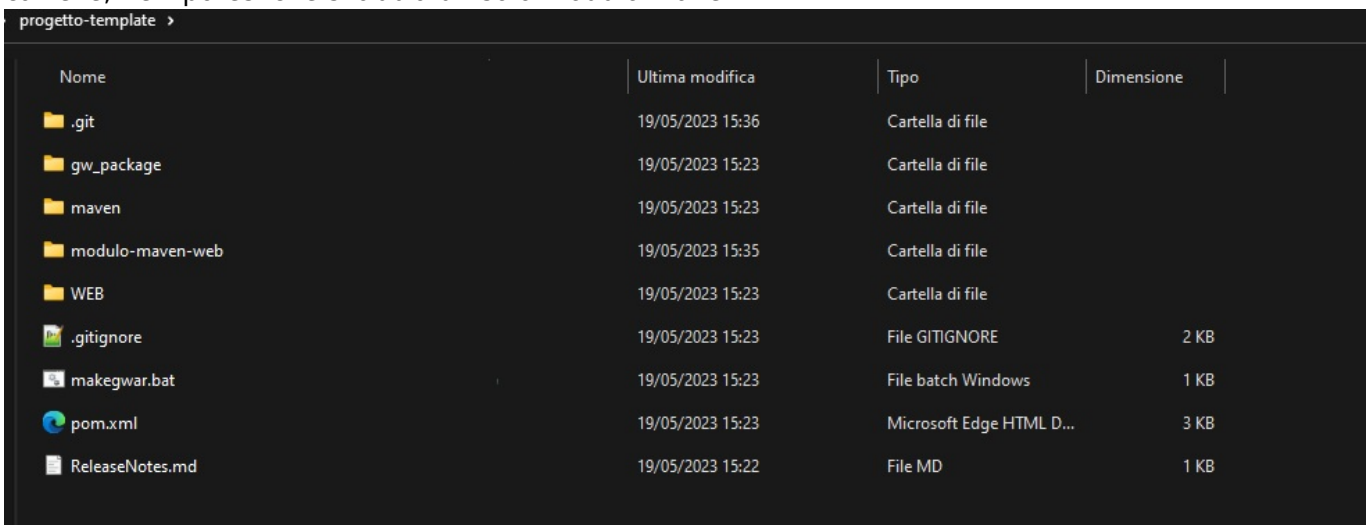
1. **il nome del nuovo modulo maven** da generare, seguito dal tasto “Invio” (sarà il nome assegnato al corrispondente war. Achtung!!! Non scegliere il nome del repository git! In assenza di fantasia, potete aggiungere il suffisso “-web” al nome del repository git. *Esempio: il nome della cartella dove sto lavorando è “cde35” ⇒ scelgo come nome del modulo maven “cde35-web”*
2. **la versione di GeowebFramework** seguita dal tasto “Invio”, scelta tra quelle rilasciate e disponibili su Artifactory e compatibile con la scelta fatta precedentemente per il file .bat (avendo scelto makegproj_per_Gw4.6.X.bat scriverò una versione del framework ad esso compatibile. *Esempio: 4.6.3*) . L’elenco delle versioni rilasciate e delle rispettive Release Notes è disponibile al seguente link.



Una volta inseriti i parametri richiesti, dopo aver premuto il tasto invio, partirà il processo maven che consentirà la creazione della cartella con le configurazioni e le librerie necessarie riferite alla versione geoweb inserita. Al termine del processo di build verrà chiesto all’utente se vuole procedere con la creazione di un nuovo modulo maven:

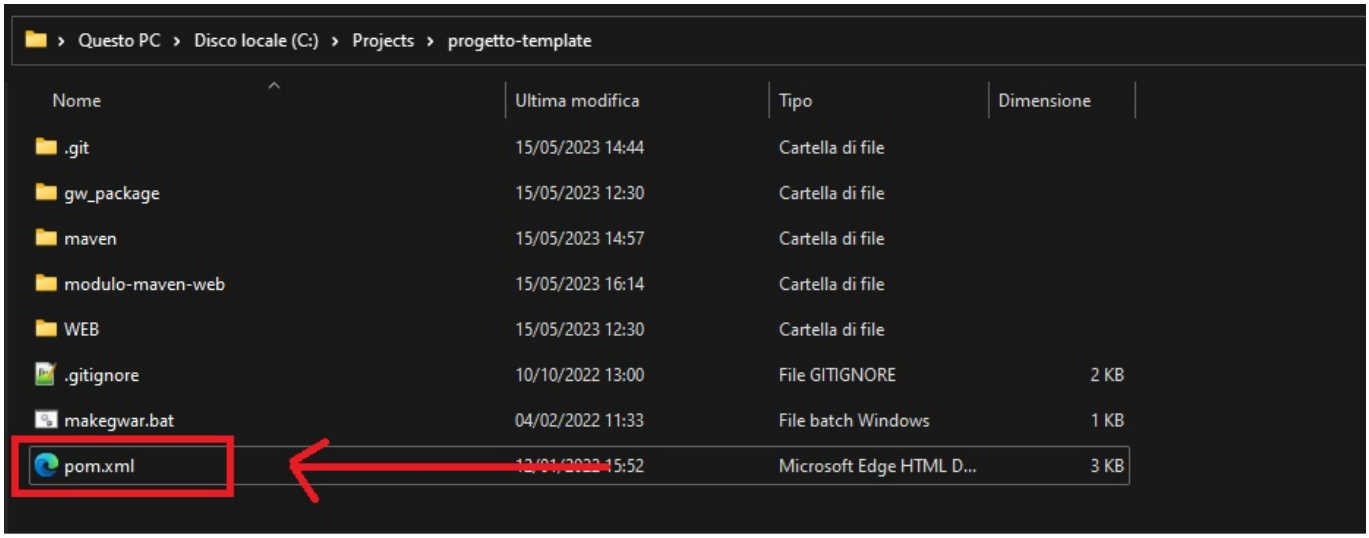


Rispondere "**S**" nel caso in cui il prodotto sia **multimodulo** e preveda più di un progetto maven (cioè, banalmente, lo stesso repository git deve contenere più war), in questo caso vengono ripetuti i punti 1 e 2 della procedura descritta. Rispondere "**N**" se si ha **un solo modulo maven** (la maggior parte dei casi) e si vuole concludere qui il processo. La finestra si chiuderà automaticamente e i file che consentono la genesi del progetto maven saranno eliminati, lasciando la seguente struttura di cartelle, nell'ipotesi che si abbia un solo modulo maven:



FILE pom.xml ESTERNO

Con file **pom.xml esterno** ci si riferisce all'omonimo file contenuto nel repository git, contenente le configurazioni maven di progetto, ed evidenziato nell'immagine seguente:



Questo file contiene informazioni essenziali sul progetto, viene letto da Maven durante la build prendendo le informazioni presenti per scaricare le dipendenze necessarie, compilare il codice sorgente e generare l'output desiderato (quindi il pacchetto WAR).

Aprire il pom.xml con un editor di testo (come Notepad) e modificare le sole parti evidenziate dai commenti:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.geowebframework</groupId>
    <!--sostituire NOME_MIO_PROGETTO con il nome del repository di prodotto-->
    <artifactId>NOME_MIO_PROGETTO</artifactId>
    <!--inserire la versione del prodotto-->
    <version>1.0.0</version>
    <packaging>pom</packaging>

    <!--introdurre tanti tag module quanti i moduli maven previsti nel proprio reposiotry di prodotto-->
    <!--il nome di ogni modulo corrisponde al nome della cartella del modulo maven corrispondente -->
    <modules>
        <module>modulomaven-1</module>
        <module>modulomaven-n</module>
    </modules>

    <!--NON MODIFICARE -->
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>

        <lombok.version>1.18.12</lombok.version>
        <logging.version>3.3.1.Final</logging.version>
        <keycloak.version>12.0.4</keycloak.version>
        <hibernate.version>5.4.10.Final</hibernate.version>
```

In particolare:

- `<artifactId>NOME_MIO_PROGETTO</artifactId>`: è il nome attribuito al progetto. Può essere assunto il nome del repository git.
- `<version>1.0.0</version>`: è la versione del progetto, corrispondente al tag del repository
- `<modules> <module>modulomaven-1</module> <module>modulomaven-n</module> </modules>` : contiene la lista dei moduli maven generati al passo precedente (quindi i nomi delle cartelle create e corrispondenti ai WAR). Nel caso di un solo modoulo maven sarà presente un solo tag di tipo module come da esempio:

```
 <modules>
    <module>modulo-maven-web</module>
</modules>
```

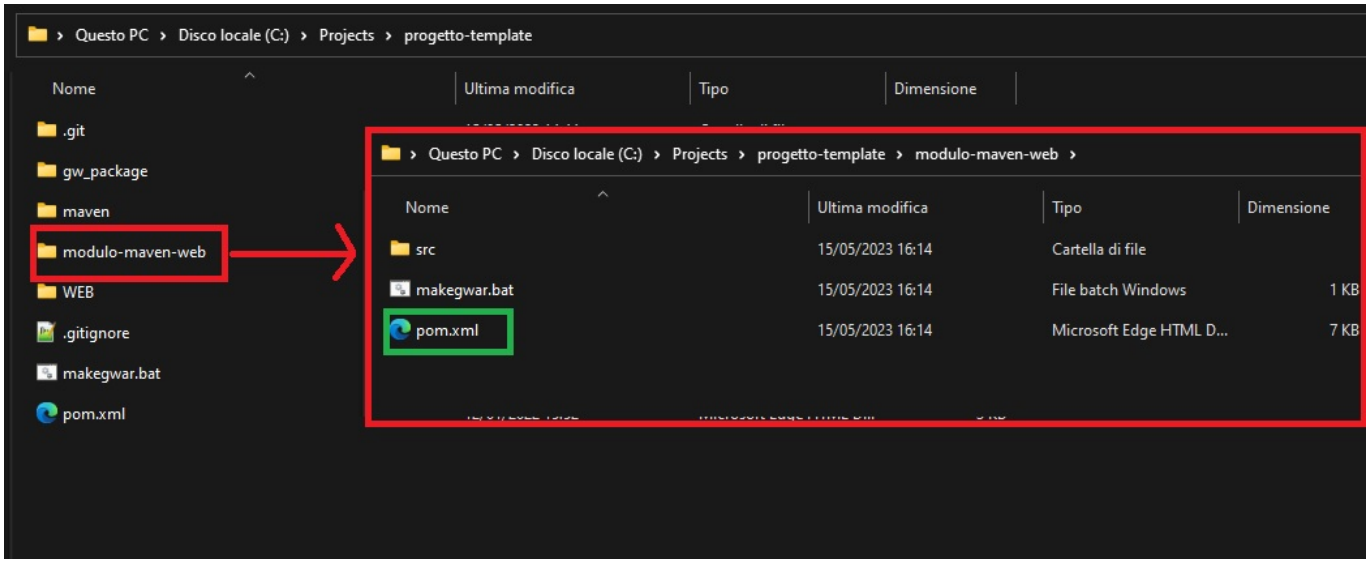
Tutto ciò che è contenuto sotto la stringa `<!— NON MODIFICARE —>` non necessita variazioni. Salvare le modifiche apportare e chiudere l'editor di testo.

FILE makegwar.bat

Il file **makegwar.bat** consente di creare i war attraverso la lettura dei file “pom.xml”. Prima di procedere alla creazione dei war editare il file **pom.xml** di progetto, come descritto qui, e i file di configurazione all’interno dei moduli maven creati ai passi precedenti e procedere alla loro personalizzazione in base alle installazioni previste come descritto qui.

Personalizzazione dei file di configurazione nei moduli maven

Generalmente, per ogni modulo maven generato come da procedura , va modificato il file **pom.xml** posto al suo interno al fine di personalizzare il WAR che sarà creato per il progetto considerato. Nel nostro esempio sarà il file evidenziato in verde:



Il file pom.xml contiene già la versione di GeowebFramework inserita nella shell durante la creazione del modulo maven (corrispondente al tag `<com.geowebframework.version>`), il nome del modulo che sarà lo stesso che verrà assegnato al WAR (corrispondente al tag `<finalName>`) , e una lista

generica di plugin geoweb. Tale lista è quella da sottoporre a personalizzazione. In particolare sotto al commento "Geoweb optional plugins" sono presenti una serie di dipendenze opzionali di Geoweb. Eliminare o commentare le dipendenze non necessarie e aggiungere quelle richieste, consultando la lista dei plugin disponibili al link o su Artifactory.

Per fare in modo che il plugin venga importato è sufficiente aggiungere un tag di tipo dependency come nell'esempio

```
<!--   G E O W E B   O P T I O N A L   P L U G I N S   -->
    <dependency>
        <groupId>com.geowebframework</groupId>
        <artifactId>gw-cde</artifactId>
        <version>${com.geowebframework.version}</version>
    </dependency>
```

Si consiglia di confrontarsi con uno sviluppatore per la scelta dei plugin.

In aggiunta, devono essere eventualmente modificati anche alcuni file di configurazione. In particolare, nel percorso [progetto-maven]\src\main\resources\ (*nell'esempio ..\modulo-maven-web\src\main\resources*) ci sono:

- Il file **configuration.properties**, che raccoglie tutti i dati configurabili del progetto, che rappresentano entità costanti, come ad esempio i dati di accesso al database; anche in questo caso, ogni parametro è memorizzato come una coppia di stringhe, una contiene il nome del parametro (cioè la chiave) e l'altra memorizza il valore. Per informazioni dettagliate su questo file consultare le appositesezione per la versione 4.4.X del framework o per la sezione 4.6.X dello stesso.

- Il file **log4j.properties** (per versioni 4.4.X del framework) o **log4j2.properties** (per verswioni 4.6.x del framework), che contiene tutte le informazioni (codificate come coppie chiave-valore) necessarie alla scrittura dei log di Maven. Il log, infatti, è uno strumento molto utile che va configurato nel modo corretto: esso raccoglie gli output dei diversi processi e, analizzando questi output, permette di capire cosa è accaduto in caso di errori. Anche qui, per informazioni dettagliate consultare le apposite sezioni per la versione 4.4.X del framework o per la sezione 4.6.X dello stesso.

- Il file **remapConfiguration.properties** che ha lo scopo di mappare i nomi delle proprietà all'interno del configuration.properties con nomi di variabili d'ambiente o di sistema.

- Il file **webconfig.ini** che serve per alcune configurazioni della piattaforma Mapguide.

Mentre, nel percorso [progetto-maven]\src\main\webapp\WEB-INF\ (*nell'esempio ..\modulo-maven-web\src\main\resources*) si trovano: - Il **dispatcher-servlet.xml**, che serve da controller per le applicazioni web basate su Spring. Il contenuto è standard, ma, in particolari ambienti, alcuni tag non vengono usati e devono essere cambiati. -il file **spring-security.xml**, che raccoglie informazioni di autenticazione e di connessione al metadata-source. -il file **spring-security-keycloak.xml** (solo versione 4.6.X del framework), che raccoglie informazioni di autenticazione nel caso in cui il server di autenticazione adottato sia keycloak -Il file **web.xml**, che descrive l'esecuzione dell'applicazione web e contiene configurazioni per l'accesso ai database. Nel caso di versione 4.6.X editare il file in modo da scegliere il file di configurazione corretto rispetto all'autenticazione scelta per il progetto, procedendo come segue:

- Aprire il file web.xml con un editor di testo (es. Notepad)
- Scorrere il file fino al tag `context-param`
- Modificare il contenuto del tag `param-value` in modo che contenga una stringa tra quelle proposte, in particolare:
  - **/WEB-INF/spring-security.xml**: Nel caso di autenticazione standard tramite geoweb
  - **/WEB-INF/spring-security-keycloak.xml**: Nel caso di autenticazione con server Keycloack. In questo caso allo stesso path del file spring-security-keycloak.xml dovrà essere aggiunto anche un file json contenete le informazioni per l'autenticazione tramite keycloack. Lo scheletro del file è scaricabile da qui , per la configurazione si rimanda all'apposita guida.

Le modifiche apportate ai vari file dipendono dal progetto, ma anche dall'ambiente in cui il progetto deve essere utilizzato, per qualsiasi dubbio rivolgersi alla sezione sviluppo

Sincronizzare con il repository remoto

Prima di continuare, verificare di avere Fork installato sul proprio PC (procedura di installazione).

Una volta creato il repository remoto, clonato in locale e creato la struttura base delle directory, è importante assicurarsi che entrambi contengano le stesse modifiche e informazioni, cioè mantenerli sincronizzati.

Quando si sincronizza un repository remoto e locale, significa che si sta facendo in modo che entrambi abbiano gli stessi cambiamenti. Ad esempio, se si è fatto delle modifiche al repository locale, si devono inviare (push) tali modifiche al repository remoto affinché le altre persone possano vederle.

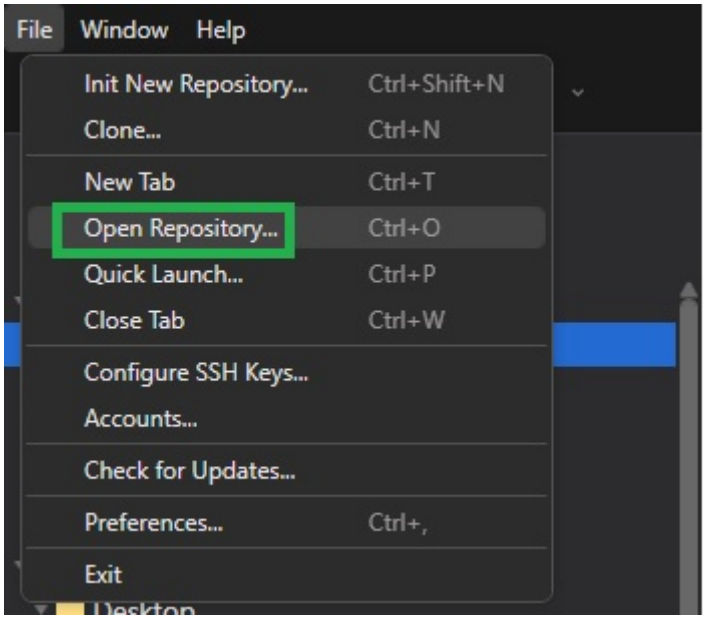
Allo stesso modo, se qualcun altro ha fatto delle modifiche nel repository remoto, dovremmo ottenere (pull) tali modifiche nel nostro repository locale per assicurarci di avere la versione più aggiornata.

In sostanza, sincronizzare un repository remoto e locale significa mantenere i due repository allineati in modo che contengano le stesse informazioni e modifiche più recenti. Ciò consente a tutti i collaboratori di lavorare con le ultime versioni dei file, mantenere la coerenza del progetto e gestire al meglio lo storico delle modifiche ai singoli file.

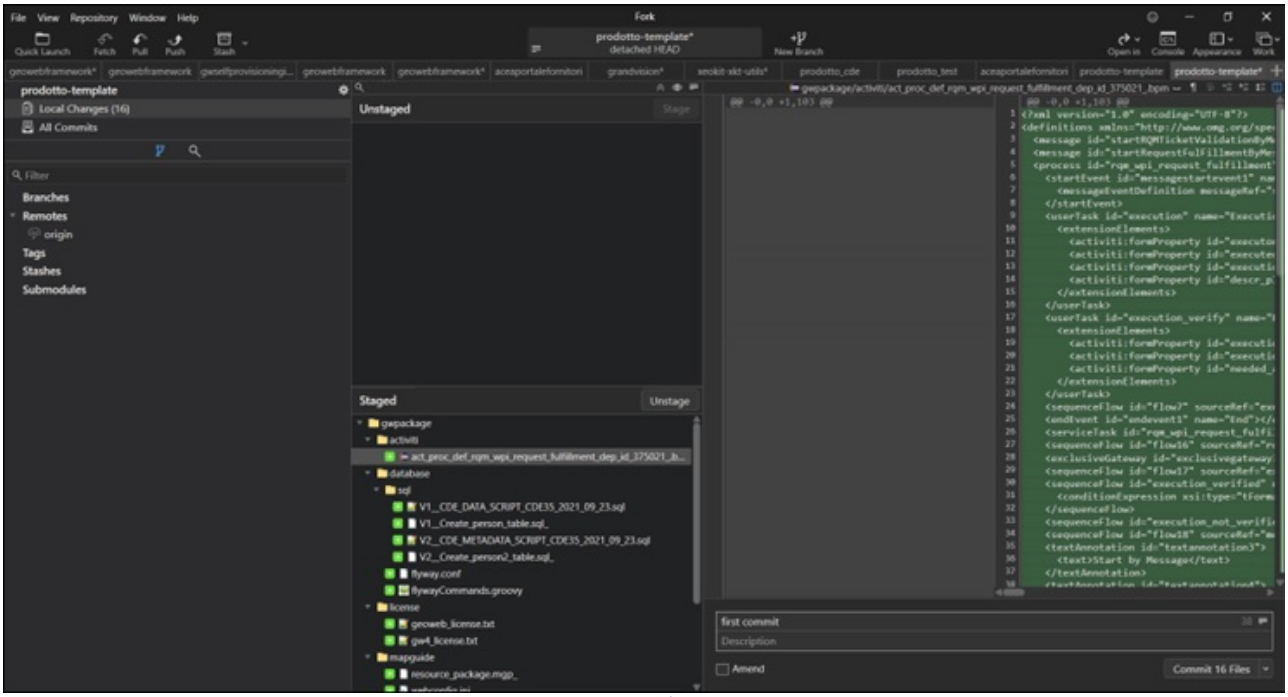
Alcuni dei comandi principali di Fork sono descritti qui.

Primo commit

1. Aprire **Fork**
2. Selezionare **File ⇒ Open Repository**



3. Scegliere la cartella con il repository GIT da importare
4. Fare click su **Seleziona cartella**
5. Selezionare sulla sinistra in alto **Local Changes** (a destra verrà mostrato un numero tra due parentesi tonde che sta ad indicare il numero di modifiche e il numero di nuovi file e cartelle create o modificati nella cartella del repository locale)
6. Nella sezione **Unstaged** selezionare in alto l'icona  per **stage all** (saranno selezionati TUTTI i file oggetto di modifica. Ad eccezione di quelli esclusi dal file .gitignore)
7. Scrivere un opportuno messaggio di commit. Normalmente il primo commit contiene per buona pratica il testo “first commit”
8. Cliccare su **Commit**



9. Verrà automaticamente generato il branch **main** [1)]

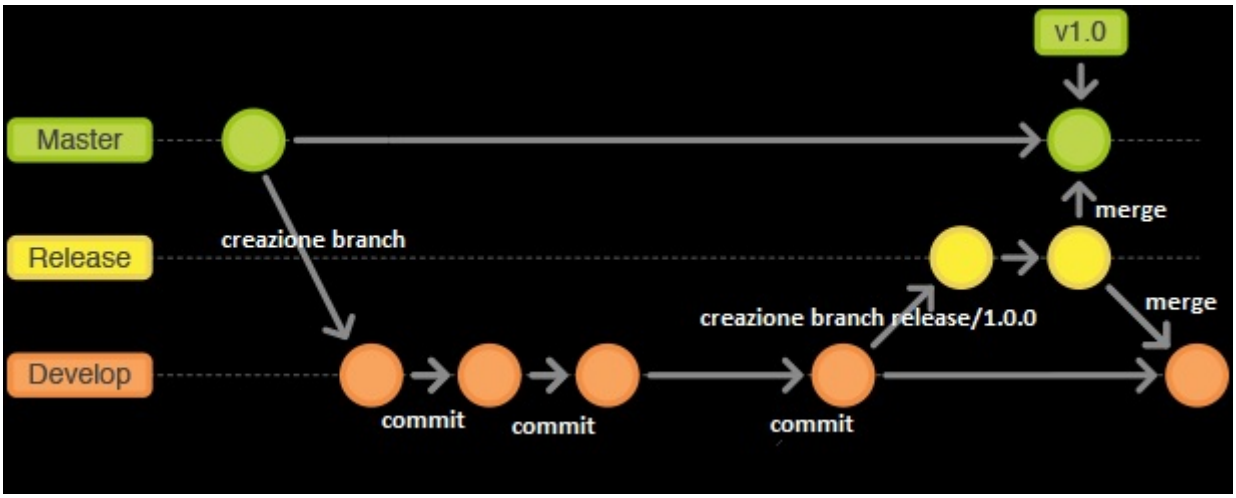
GitFlow del repository di progetto

Gitflow è un modello di branching e workflow per la gestione dei repository GIT, che offre una struttura ben definita per organizzare il flusso di lavoro e gestire le modifiche nel repository.

Il Gitflow si basa sulla creazione di diversi branch (rami di lavoro) con scopi specifici. Ecco una panoramica dei principali branch utilizzati nel Gitflow che adotteremo nel nostro specifico flusso di lavoro:

1. **Branch "main" o "master"**: Rappresenta il branch principale del repository, contenente le versioni rilasciate.
1. **Branch "develop"**: È il branch di sviluppo principale, dove vengono integrate le nuove funzionalità e le modifiche non ancora pronte per il rilascio.
1. **Branch "release"**: Quando il lavoro sul branch "develop" raggiunge uno stato stabile e pronto per il rilascio, viene creato un branch "release/X.X.X" essendo X.X.X la versione in corso di rilascio. In questo branch, vengono effettuati i test e le preparazioni per la distribuzione del progetto. Una volta completato, il branch "release/X.X.X" viene integrato sia nel branch "main" (o "master") che nel branch "develop" attraverso la procedura di **merge**.

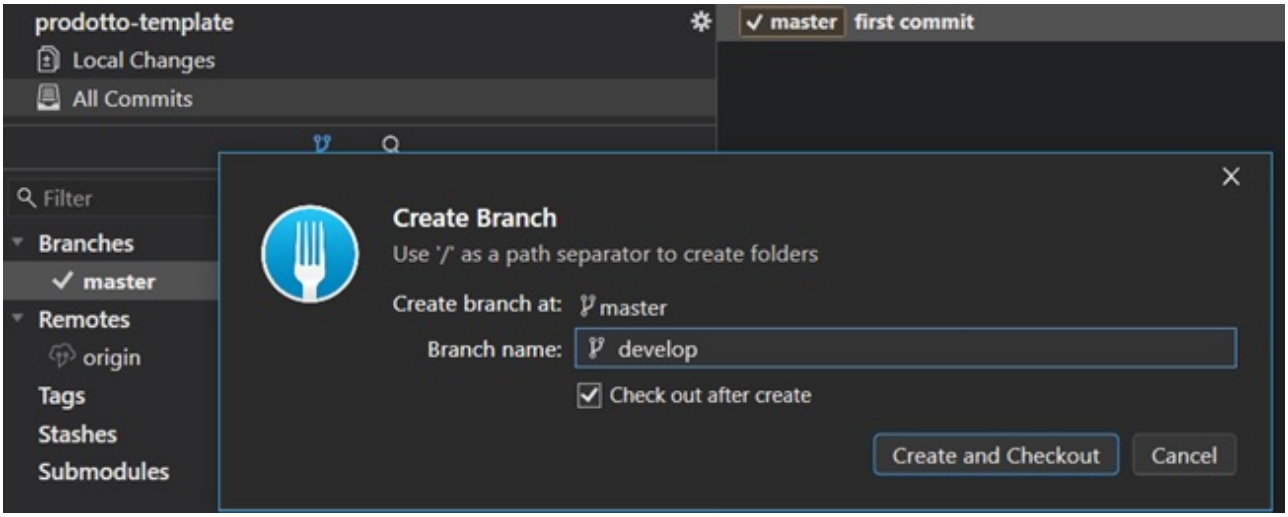
Il Gitflow definisce regole chiare su come e quando integrare i branch e promuovere le modifiche attraverso il flusso di lavoro. Ciò aiuta a mantenere la stabilità del codice, a facilitare la collaborazione tra i membri del team, a fornire una struttura organizzativa chiara nel processo di sviluppo del software gestendo al meglio e puntualmente le modifiche oggetto di ogni commit.



Creazione del branch «develop»

Per la creazione del branch **develop** seguire i passi seguenti:

1. In **Fork** selezionare con il pulsande destro del mouse il **branch main** (o **master**)
2. Cliccare su **New Branch…**
3. In corrispondenza di **Branch name** digitare **develop**
4. selezionare **Create and Checkout**

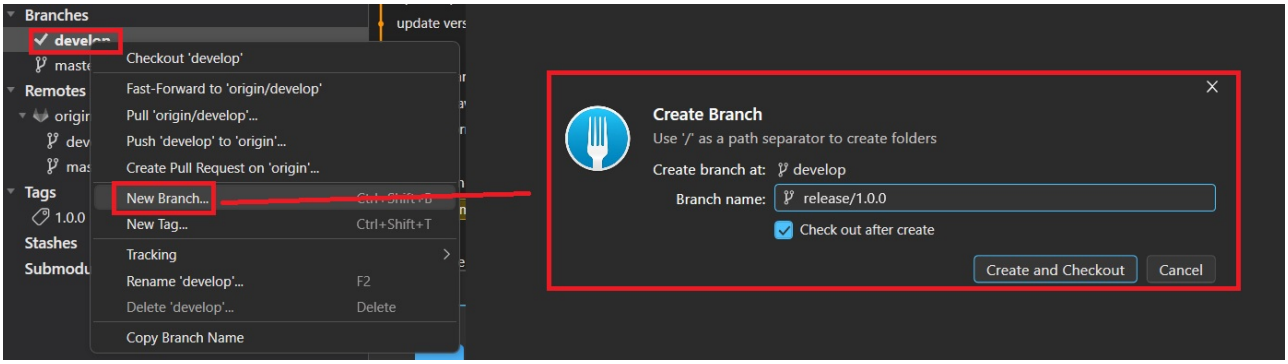


A questo punto il branch corrente è **develop** (un segno di spunta ✓ comparirà accanto al nome del branch), il repository remoto e locale sono allineati, perciò si può procedere con gli sviluppi ed eseguire tutti i commit necessari dal branch corrente, fino al termine dello sviluppo. Una volta terminato si è pronti per fare il primo rilascio.

Creazione del branch «release»

Quando il lavoro sul branch “develop” raggiunge uno stato stabile e pronto per il rilascio, creare un branch **release** consente di focalizzarsi sulla stabilizzazione delle funzionalità. In questo modo, è possibile effettuare gli ultimi test e preparazioni per il rilascio senza interferire con lo sviluppo delle nuove funzionalità in corso.

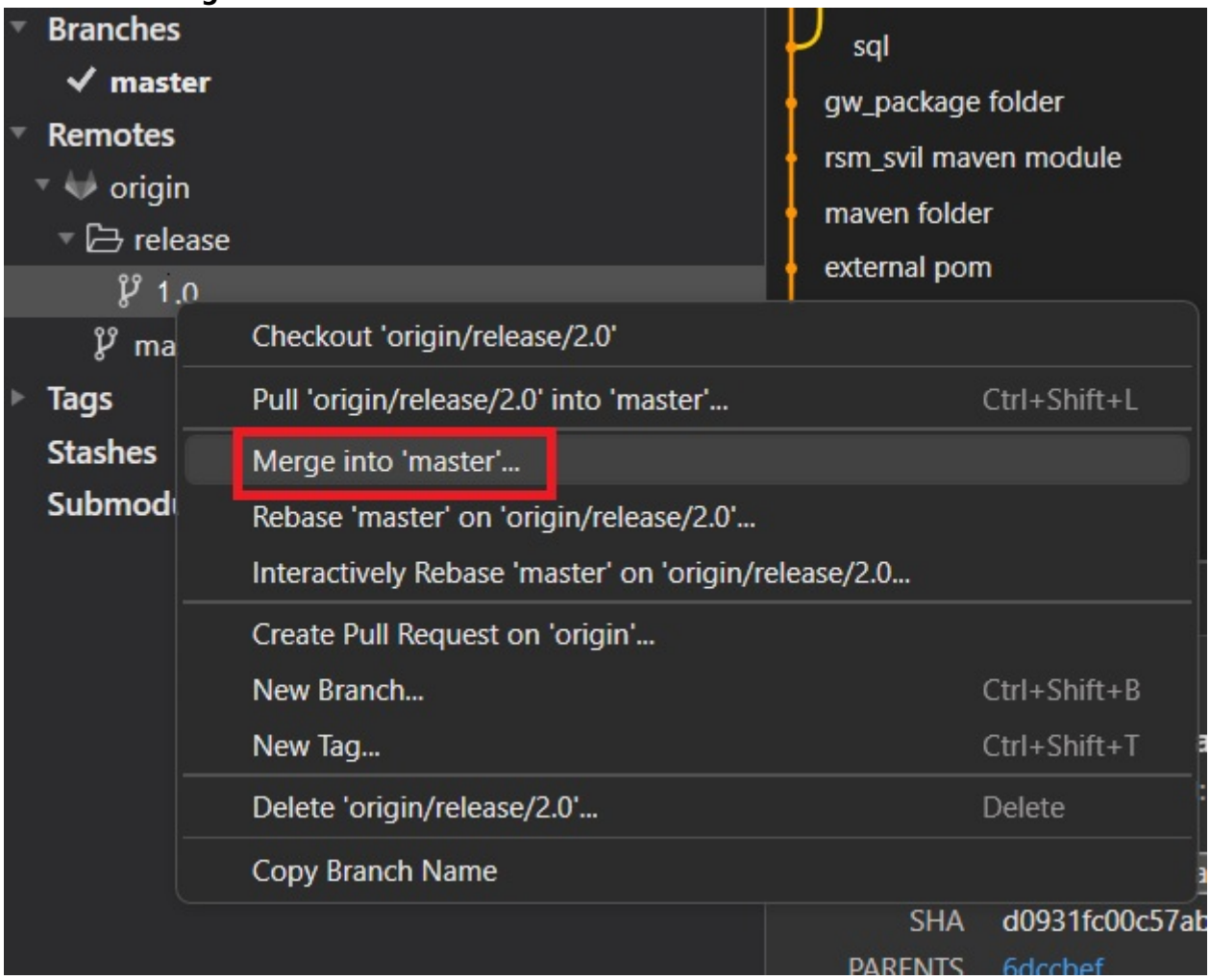
1. In Fork accertarsi di essere posizionati sul branch develop (a sinistra del nome develop c’è un segno di spunta ✓)
2. Assicurarsi di aver committato TUTTE le modifiche in locale (a sinistra Local Changes non presenta numeri tra parentesi tonde)
3. Selezionare con il pulsande destro del mouse il **branch develop**
4. Cliccare su **New Branch…**
5. In corrispondenza di **Branch name** digitare **release/1.0.0** dove 1.0.0 dovrà ogni volta essere sostituito dalla versione in corso di rilascio
6. Selezionare **Create and Checkout**



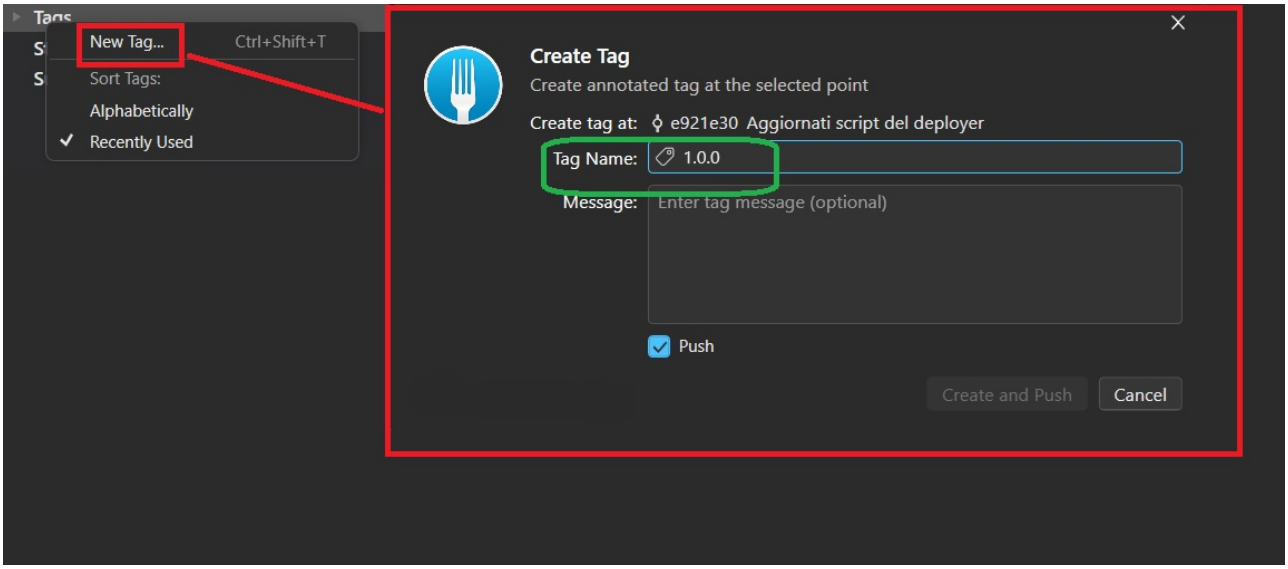
Rilascio versione e Tag

Una volta eseguito il collaudo e ultimati i test, solo dopo aver ricevuto l'ok, è possibile rilasciare la versione e creare il tag nel repository GIT.

1. In fork, fare doppo click sul **branch main o master** (aspettare che il simbolo di check ✓ si posizioni a sinistra di master)
2. Click con il pulsante destro del mouse sul branch di release che si vuole rilasciare (*Esempio: release/1.0.0*)
3. Selezionare **Merge into 'master'...**



4. Andare su tags (in basso a sinistra), fare click con il pulsante destro del mouse e scegliere **New tag**
5. In corrispondenza di **Tag Name** digitare il numero di versione rilasciata
6. Assicurarsi di aver selezionato **Push** (serve per pubblicare il tag anche sul repository remoto)
7. Cliccare su **Create and Push**



8. Fare doppio click sul branch **develop**
9. Click con il pulsante destro del mouse sul branch di release appena rilasciata (*Esempio: release/1.0.0*)
10. Selezionare **Merge into 'develop'…**
11. Cliccare **Push**

Ad ogni rilascio deve corrispondere un tag corrispondente alla versione rilasciata. Il versionamento deve seguire le regole consuete del versionamento semantico

Regole base di versionamento

Le regole di versionamento dei rilasci possono variare a seconda del contesto e delle preferenze del progetto. Tuttavia, esistono alcune convenzioni comuni che possono essere seguite per gestire il versionamento dei rilasci del progetto. In particolare si è scelto di adottare il sistema di versionamento semantico (Semantic Versioning), che prevede il formato **"MAJOR.MINOR.PATCH"**.

Le regole di incremento sono le seguenti:

1. Incremento del numero **MAJOR** quando vengono apportate modifiche incompatibili con la versione precedente.
2. Incremento del numero **MINOR** quando vengono aggiunte nuove funzionalità in modo retrocompatibile.
3. Incremento del numero **PATCH** quando vengono effettuate correzioni di bug retrocompatibili.

1)

Attenzione: dal 2022 il branch di riferimento di GIT è il branch **main** e non più il branch **master**. È solo un cambio di nome perchè "master" non è un termine "inclusivo"… Per i vecchi progetti può continuare ad essere utilizzato "master"