

# Programmazione avanzata

## Lezione 2

### Tool di sviluppo avanzati

#### Controllo di versione – GIT

##### Introduzione generale

Un tool importante per lo sviluppo software è un sistema di controllo versione. Un sistema di controllo versione è un tool che permette di organizzare e tenere traccia le varie versioni dello sviluppo di un software nel tempo (vedrete i dettagli nell'esame di Ingegneria del software). Tali tools tengono traccia della storia dello sviluppo, delle modifiche (aggiunte/rimozioni) effettuate da ciascuno sviluppatore, permettendo di gestire il rilascio delle versioni al pubblico e anche l'eventuale annullamento di modifiche dannose (e.g. perché causano un bug). Tali tool sono fondamentali per gestire progetti che coinvolgono più persone perché permettono anche di gestire i cosiddetti conflitti, cioè modifiche alle stesse parti del software effettuate in contemporanea. Il codice e le sue versioni sono generalmente memorizzati in un repository, un server che contiene tutte le versioni disponibile a tutti gli sviluppatori. Quando uno sviluppatore effettua delle modifiche al software stesso, tramite il tool di controllo di versione carica le modifiche sul repository in maniera tale da renderle disponibili a tutti.

Uno dei tool di controllo versione più popolari oggi è GIT. GIT è un tool distribuito, questo vuol dire che è in grado di gestire più repository contemporaneamente per lo stesso progetto. GIT richiede almeno due repository, un repository locale (sul PC dello sviluppatore) in cui vengono caricati i cambiamenti fatti dal programmatore, e un repository generale (su un server remoto) in cui il programmatore periodicamente carica tutti i cambiamenti fatti per renderli disponibili agli altri.

Esistono diversi siti che permettono la creazione di repository GIT privati o pubblici quali ad esempio github.com o gitlab.com (noi useremo quest'ultimo).



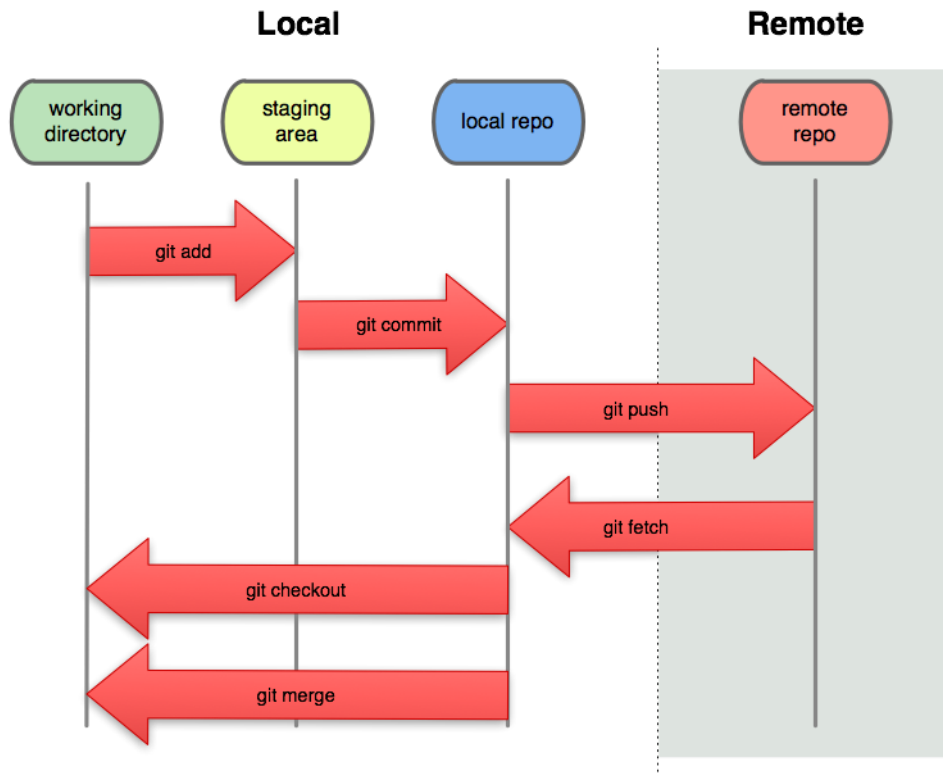
GIT lavora attraverso tre aree:

- **Working directory** - l'area che contiene tutti i files del progetto, alcuni dei quali potrebbero anche essere files temporanei e che non fanno parte del progetto stesso (e.g. files di log o files di compilazione);
- **Staging area** – è il sottoinsieme dei files della working directory che fanno parte del progetto;
- **Repository (locale o remoto)** – è il repository nel quale vengono inviati le versioni del software in maniera periodica o alla fine dell'implementazione di una funzione.



Le operazioni di base che si possono effettuare con GIT sono le seguenti:

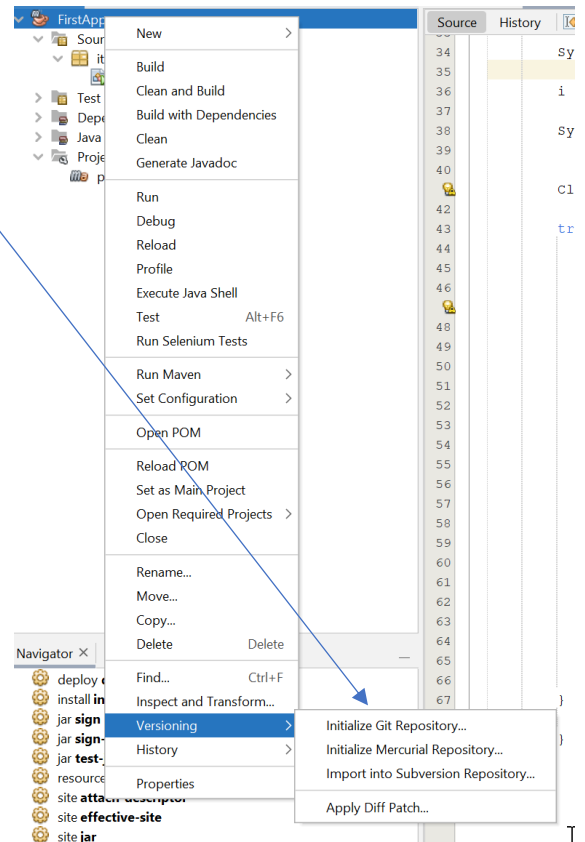
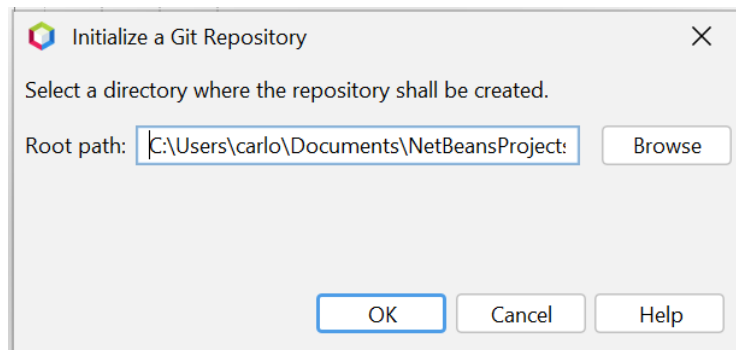
- **git add** – con questa operazione si possono aggiungere nuovi files al progetto. I files vengono aggiunti alla staging area;
- **git commit** – con questa operazione la versione corrente di tutti i files nella staging area vengono inviati al local repository. Il commit crea una nuova versione del software nel repository locale;
- **git push** – con questa operazione la versione corrente (con la storia) vengono inviati ad un repository remoto in modo da mettere a disposizione le versioni locali a tutti gli altri sviluppatori;
- **git fetch** – con questa operazione si recuperano le ultime versioni del software dal repository remoto aggiornando il repository locale con i cambiamenti prodotti dagli altri sviluppatori;
- **git checkout** – con questa operazione si ritorna ad una versione precedente;
- **git merge** – con questa operazione si risolvono conflitti nel software, solitamente a seguito di una git fetch dovuto al fatto che un collega ha modificato il software nello stesso punto;
- **git clone** – con questa operazione si crea un nuovo repository locale 'clonando' un repository remoto, cioè si scarica un software già parzialmente implementato da altri.



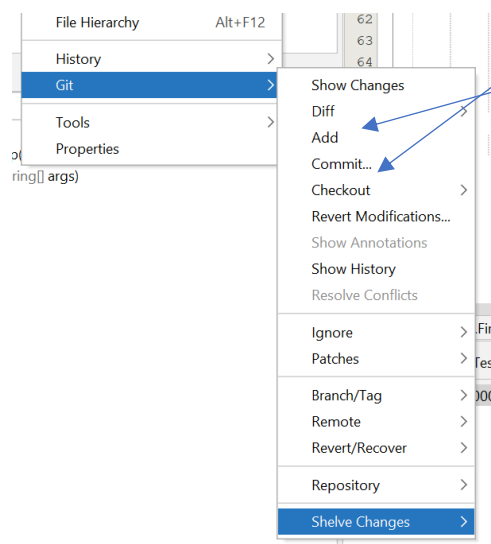
## Operazioni GIT base in NetBeans

Un nuovo repository locale può essere creato a partire da un progetto esistente attraverso la funzione “Initialize Git Repository”.

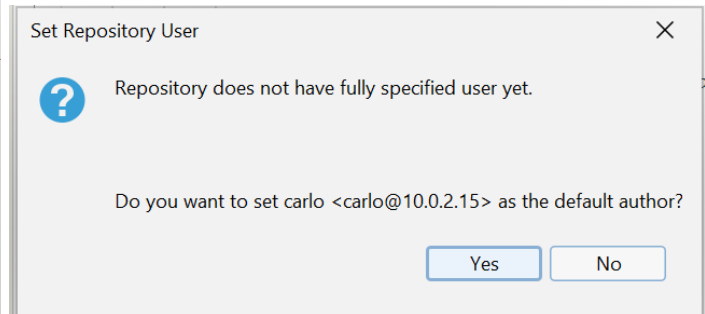
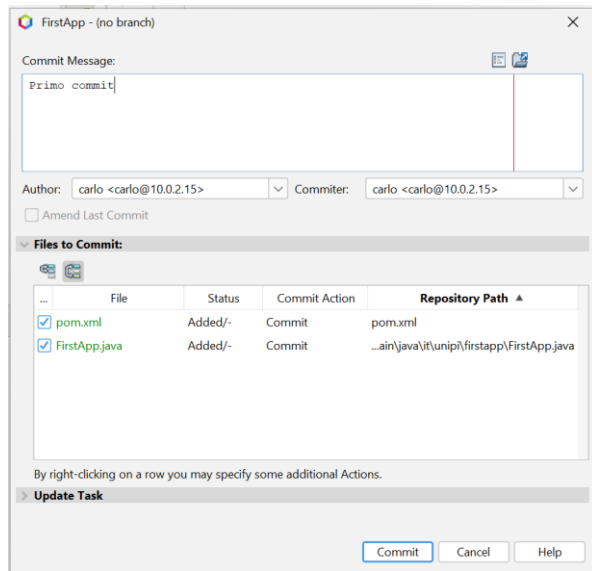
Il comando, in seguito, chiede la directory dove creare il repository locale.



Una volta creato il repository si può procedere ad aggiungere i files del progetto tramite la add e successivamente si può procedere a fare il primo commit.



L'operazione di commit apre una nuova finestra per specificare il messaggio da associare al commit, in seguito si può aprire anche una finestra per la verifica dell'identità.




## Uso di un repository remoto - GitLab

Il sito gitlab.com è uno dei siti che offre la creazione di repository remoti. A seguito dell'iscrizione si possono creare dei repository remoti privati o pubblici per la gestione di sviluppo software collaborativo o semplicemente per salvare il vostro lavoro in un server remoto.


Il primo passo è quello di creare un nuovo progetto:

### Create new project




#### Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.




#### Create from template

Create a project pre-populated with the necessary files to get you started quickly.



#### Import project

Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.



#### Run CI/CD for external repository

Connect your external repository to GitLab CI/CD.

Specificando in seguito il nome e la visibilità del progetto (pubblica o privata).

New project > Create blank project

### Project name

My awesome project

### Project URL

https://gitlab.com/ provaapp

### Project slug

my-awesome-project

Want to organize several dependent projects under the same namespace? [Create a group.](#)

### Project deployment target (optional)

Select the deployment target

### Visibility Level ?



Private

Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.



Public

The project can be accessed without any authentication.

### Project Configuration



Initialize repository with a README

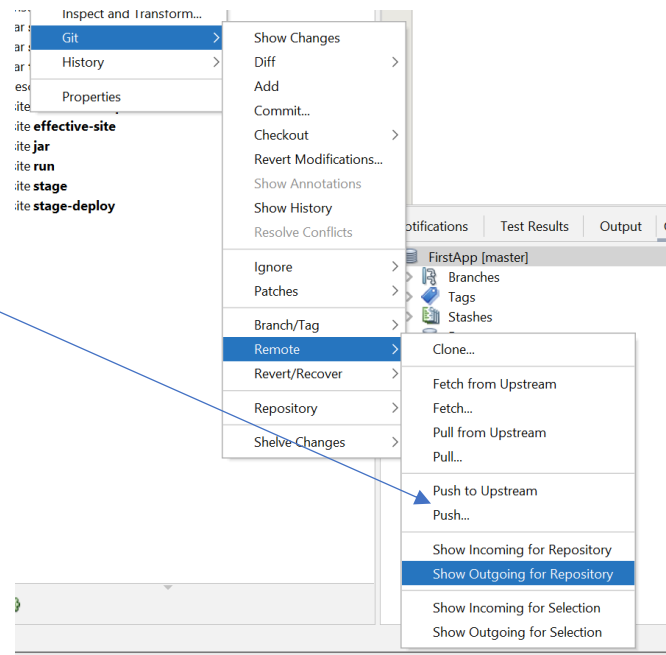
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.



Enable Static Application Security Testing (SAST)

Analyze your source code for known security vulnerabilities. [Learn more.](#)

Una volta creato il progetto si può procedere a caricare il repository locale sul repository remoto tramite la funzione netbeans  
Git->Remote->Push



In questa fase si deve specificare l'indirizzo del repository remoto e le credenziali per l'autenticazione.

The screenshot shows the 'Push to Remote Repository' dialog box with the 'Remote Repository' step selected. The 'Specify Git Repository Location' option is chosen. The 'Remote Name' is set to 'origin' and 'Persist Remote' is checked. The 'Repository URL' is 'https://gitlab.com/povaapp/applicazione-di-prova.git'. The 'User' is 'o.vallati@unipi.it' and 'Save Password' is checked. A 'Proxy Configuration...' button is at the bottom. Navigation buttons at the bottom are '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**Push to Remote Repository**

**Steps**

1. **Remote Repository**
2. Select Local Branches
3. Update Local References

**Remote Repository**

☐ Select Configured Git Repository Location:

☒ Specify Git Repository Location:

Remote Name:  ☒ Persist Remote

Repository URL:

User:  (leave blank for anonymous access)

Password:  ☒ Save Password

[Proxy Configuration...](#)

< Back Next > Finish Cancel Help

The left screenshot shows the 'Update Local References' step. The 'Remote Branches' list contains 'master -> origin/master [A]' which is selected. The right screenshot shows the 'Select Local Branches' step. The 'Local Branches' list contains 'master -> master [A]' which is selected. Both screenshots have 'Select All' and 'Select None' buttons at the bottom. Navigation buttons at the bottom are '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**Push to Remote Repository**

**Steps**

1. Remote Repository
2. Select Local Branches
3. **Update Local References**

**Update Local References**

Select remote branches you want to update under remotes in the local repository

Remote Branches

- ☒ master -> origin/master [A]

Select All Select None

< Back Next > Finish Cancel Help

**Push to Remote Repository**

**Steps**

1. Remote Repository
2. **Select Local Branches**
3. Update Local References

**Select Local Branches**

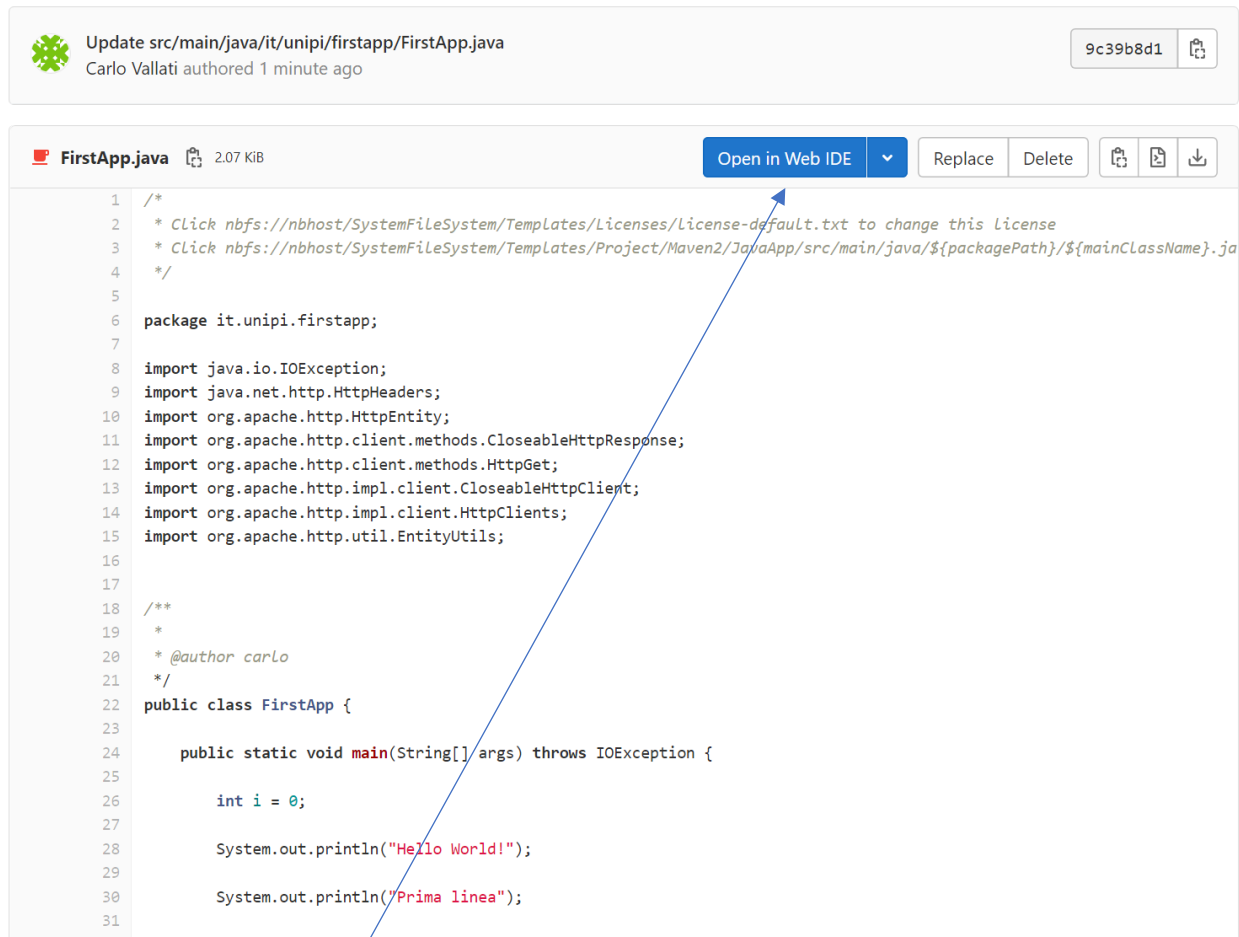
Local Branches

- ☒ master -> master [A]

Select All Select None

< Back Next > Finish Cancel Help

Una volta confermato il push il codice è disponibile anche sul repository remoto:



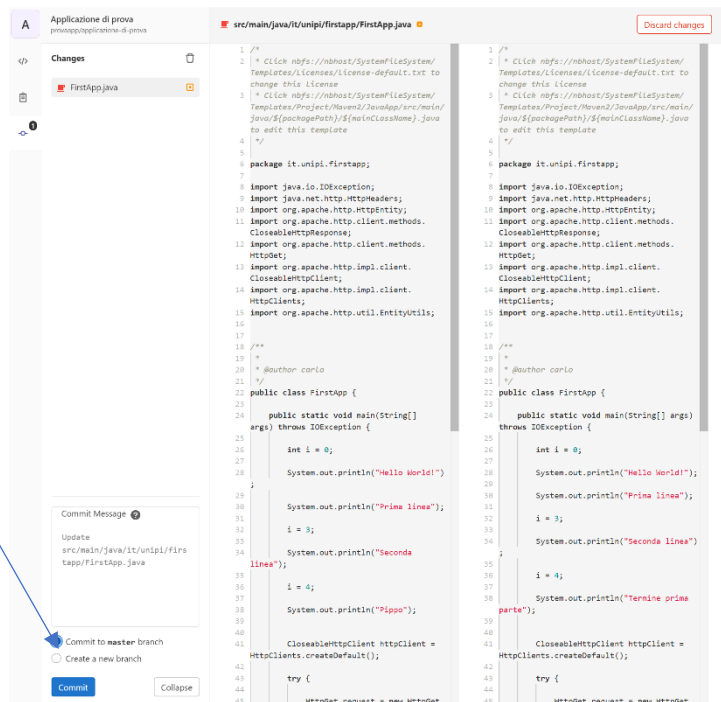
The screenshot shows a GitLab interface. At the top, a notification bar indicates an update to `src/main/java/it/unipi/firstapp/FirstApp.java` by Carlo Vallati, authored 1 minute ago. Below this, the file `FirstApp.java` is displayed with a size of 2.07 KiB. The code editor shows the following content:

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this License
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Project/Maven2/JavaApp/src/main/java/${packagePath}/${mainClassName}.java to edit this template
4   */
5
6  package it.unipi.firstapp;
7
8  import java.io.IOException;
9  import java.net.http.HttpHeaders;
10 import org.apache.http.HttpEntity;
11 import org.apache.http.client.methods.CloseableHttpResponse;
12 import org.apache.http.client.methods.HttpGet;
13 import org.apache.http.impl.client.CloseableHttpClient;
14 import org.apache.http.impl.client.HttpClients;
15 import org.apache.http.util.EntityUtils;
16
17 /**
18  *
19  * @author carlo
20  */
21
22 public class FirstApp {
23
24     public static void main(String[] args) throws IOException {
25
26         int i = 0;
27
28         System.out.println("Hello World!");
29
30         System.out.println("Prima linea");
31     }
32 }
```

Buttons for "Open in Web IDE", "Replace", and "Delete" are visible at the top right of the editor.

GitLab fornisce anche un editor web per modificare i files online ed effettuare commit. Proviamo a modificare un file e ad effettuare un commit per simulare un altro sviluppatore che modifica il codice.

Dopo aver effettuato il commit si può scaricare la nuova versione attraverso il comando `Git->Remote->Pull` da NetBeans.



The screenshot shows the GitLab web editor interface. The file `src/main/java/it/unipi/firstapp/FirstApp.java` is open. The code editor shows the following content:

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this License
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Project/Maven2/JavaApp/src/main/java/${packagePath}/${mainClassName}.java to edit this template
4   */
5
6  package it.unipi.firstapp;
7
8  import java.io.IOException;
9  import java.net.http.HttpHeaders;
10 import org.apache.http.HttpEntity;
11 import org.apache.http.client.methods.CloseableHttpResponse;
12 import org.apache.http.client.methods.HttpGet;
13 import org.apache.http.impl.client.CloseableHttpClient;
14 import org.apache.http.impl.client.HttpClients;
15 import org.apache.http.util.EntityUtils;
16
17 /**
18  *
19  * @author carlo
20  */
21
22 public class FirstApp {
23
24     public static void main(String[] args) throws IOException {
25
26         int i = 0;
27
28         System.out.println("Hello World!");
29
30         System.out.println("Prima linea");
31
32         i = 3;
33         System.out.println("Seconda linea");
34
35         i = 4;
36         System.out.println("Terza linea");
37
38         CloseableHttpClient httpClient = HttpClients.createDefault();
39         try {
40             HttpGet request = new HttpGet
41         }
42     }
43 }
```

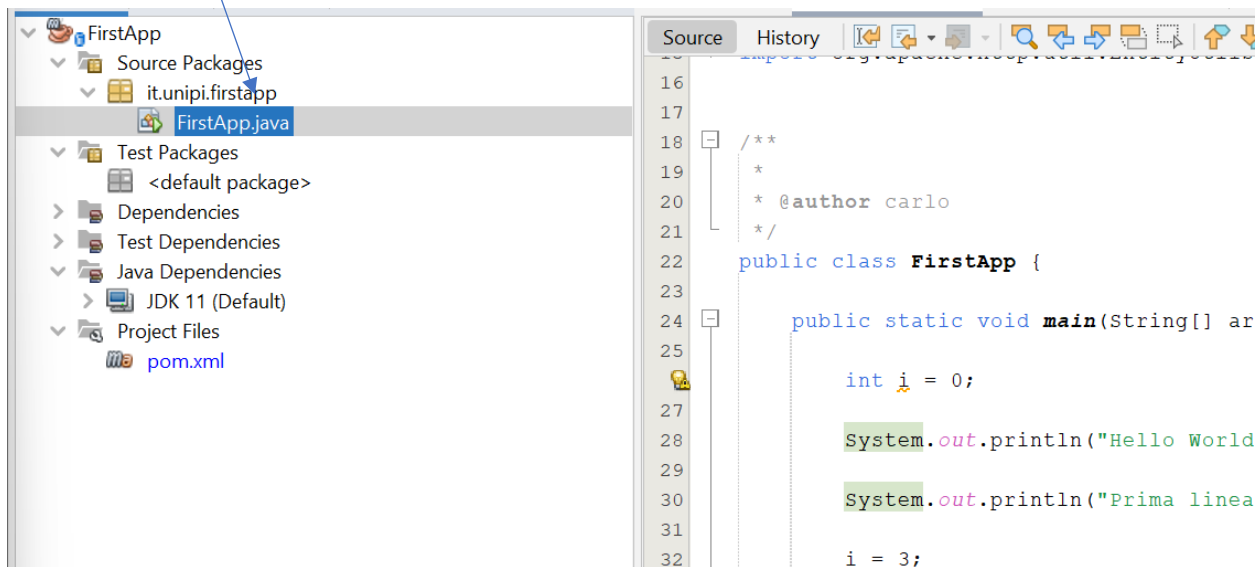
At the bottom, a "Commit Message" dialog is open, showing the message "Update src/main/java/it/unipi/firstapp/FirstApp.java". The "Commit" button is highlighted.



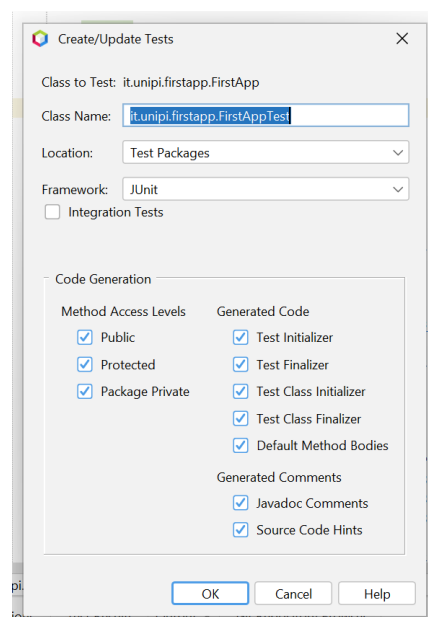
## Unit test

Lo sviluppo continuo del software da parte di piu' persone puo' portare all'introduzione di bug o malfunzionamenti nel codice esistente. Al fine di evidenziare tali problemi e contenerne gli effetti nello sviluppo software collaborativo possono essere utilizzati degli strumenti automatici per il test del software. Questi strumenti chiamati unit test sono dei codici introdotti durante lo sviluppo del software per collaudare lo stesso in maniera automatizzata durante lo sviluppo. Questi test sono chiamati Unit Test, perche' solitamente sono sviluppati in maniera sinergica con le varie unita' (o moduli) del software.

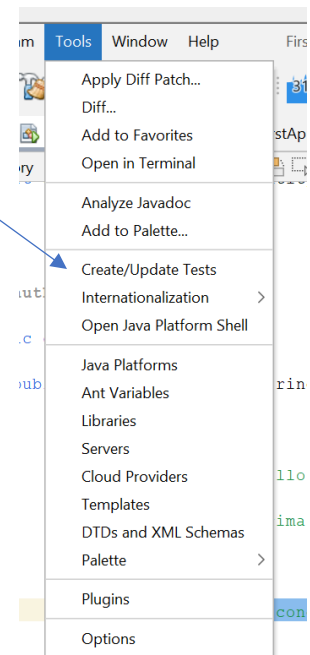
Per creare un nuovo unit test in netbeans bisogna selezionare la classe per cui si vuole creare il test dal menu a sinistra.



E successivamente usare il menu Tools-> Create/Update Tests



La finestra che si apre serve per specificare il nome del test e del package nel quale questo dovrà essere creato.



Il sistema crea automaticamente una classe di test vuota con uno scheletro di funzioni che compongono il test stesso.

Le funzioni contraddistinte dalla dicitura `@Test` sono tutte funzioni che implementano un test per una specifica funzionalità della classe. Queste funzioni verranno invocate automaticamente da Maven al momento della compilazione per testare che una modifica al codice non abbia compromesso alcune delle funzionalità esistenti dell'applicazione.

Ogni test nel caso di fallimento deve invocare la funzione `fail` che indica al sistema il fallimento.

```
/**
 *
 * @author carlo
 */
public class FirstAppTest {

    public FirstAppTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    /**
     * Test of main method, of class FirstApp.
     */
    @Test
    public void testMain() throws Exception {
        System.out.println("main");
        String[] args = null;
        FirstApp.main(args);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }
}
```

Proviamo a creare una funzione nella classe principale e a scrivere il test corrispondente a lanciare poi Maven e vedere se il test viene eseguito.

```
@Test
public void testSomma() {
    if (FirstApp.somma(1, 2) != 3) {
        System.out.println("Fallito");
        fail("Test somma fallito");
    } else {
        System.out.println("OK");
    }
}

//
public class FirstApp {

    public static int somma(int a, int b) {
        return a+b;
    }
}
```

#### Esercizio

Creare un nuovo repository su gitlab e caricare il codice della prima applicazione. Includere nell'applicazione anche un test.

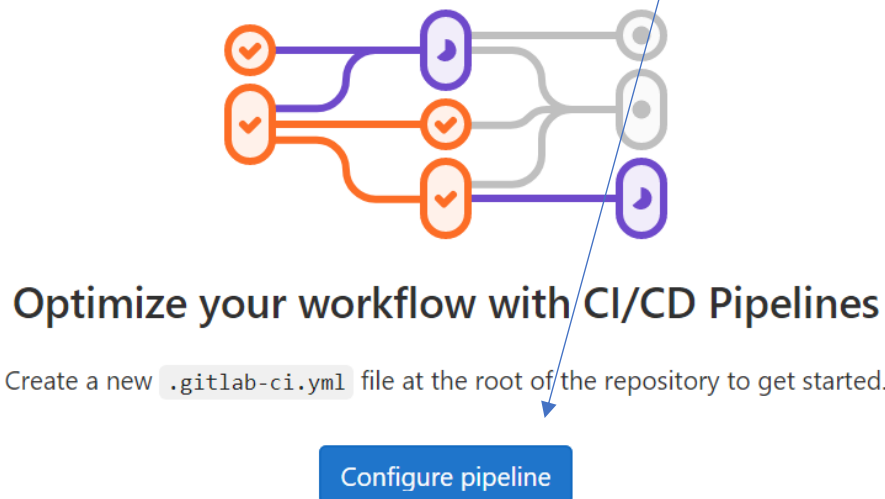
## Test automatizzati in GitLab

I repository centralizzati possono essere utilizzati non solo per memorizzare le versioni ma anche per effettuare i test automatici al fine di verificare che il nuovo codice prodotto dagli sviluppatori non comprometta alcune delle funzionalità esistenti. In questo ultimo caso il sistema può anche rifiutare l'invio del codice.

Questa pratica fa parte di una nuova pratica di sviluppare software chiamata Continuous Integrazion/Continuous Delivery (CI/CD) che è un approccio per lo sviluppo di software, focalizzato sull'automazione delle procedure che portano il codice dallo sviluppo all'integrazione, dal test alla distribuzione e deployment finale.

Vediamo come si può utilizzare GitLab per l'esecuzione automatica dei test ogni volta che delle nuove versioni del software vengono caricate. Carichiamo il nuovo codice fatto con il relativo test.

Creiamo in gitlab un nuovo task CI/CD e creiamo una nuova 'pipeline'<sup>1</sup>, cioè una serie di operazioni di test che la piattaforma deve effettuare ogni volta che uno sviluppatore invia il codice.



<sup>1</sup> Per attivare questa funzionalità GitLab vuole l'inserimento di una carta di credito, questo perché la funzionalità CD/CI implica un utilizzo di risorse per l'esecuzione dei test e la funzionalità è gratuita solo sotto una determinata soglia.

Il sistema automaticamente crea una pipeline vuota. Ogni pipeline di esecuzione dei test è composta da 3 fasi:

- build – il codice sorgente viene compilato e il binario viene costruito
- test – i test di unità vengono eseguiti
- deploy – se i test eseguiti sono tutti positivi una nuova release del software può essere rilasciata automaticamente

```
18
19 stages:           # List of stages for jobs, and their order of execution
20   - build
21   - test
22   - deploy
23
24 build-job:         # This job runs in the build stage, which runs first.
25   stage: build
26   script:
27     - echo "Compiling the code..."
28     - echo "Compile complete."
29
30 unit-test-job:     # This job runs in the test stage.
31   stage: test      # It only starts when the job in the build stage completes successfully.
32   script:
33     - echo "Running unit tests... This will take about 60 seconds."
34     - sleep 60
35     - echo "Code coverage is 90%"
36
37 lint-test-job:     # This job also runs in the test stage.
38   stage: test      # It can run at the same time as unit-test-job (in parallel).
39   script:
40     - echo "Linting code... This will take about 10 seconds."
41     - sleep 10
42     - echo "No lint issues found."
```

Nel nostro caso modifichiamo la pipeline per includere nello stage build il comando 'mvn compile' che serve per produrre il JAR del nostro programma, mentre nella fase test inseriamo il comando 'mvn test' che serve per lanciare solo i test. La fase di deploy può essere lasciata vuota.

```
image: maven:latest

stages:           # List of stages for jobs, and their order of execution
  - build
  - test
  - deploy

build-job:         # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - mvn compile
    - echo "Compile complete."

unit-test-job:     # This job runs in the test stage.
  stage: test      # It only starts when the job in the build stage completes successfully.
  script:
    - echo "Running unit tests... This will take about 60 seconds."
    - mvn test
    - sleep 60
    - echo "Code coverage is 90%"

deploy-job:        # This job runs in the deploy stage.
  stage: deploy    # It only runs when *both* jobs in the test stage complete successfully.
  environment: production
```

Una volta modificata la pipeline il test parte automaticamente (la creazione della pipeline stessa rappresenta un commit). Il test che abbiamo creato include un test due test uno dei quali fallisce, il risultato è quindi una segnalazione di fallimento del test all'interno della piattaforma.

✖ failed

Update .gitlab-ci.yml file

#635897712 master 433bab22

00:01:23  
6 minutes ago

🔄 ⬇️

## Update .gitlab-ci.yml file

🕒 3 jobs for master in 1 minute and 23 seconds (queued for 1 second)

🔗 433bab22

🔗 No related merge requests found.

Pipeline

Needs

Jobs 3

Failed Jobs 1

Tests 0

build

test

deploy

✓ build-job

✖ unit-test-job

» deploy-job

```
583 Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 1.805 sec <<< FAILURE!  
584 it.unipi.firstapp.FirstAppTest.testMain() Time elapsed: 1.781 sec <<< FAILURE!  
585 org.opentest4j.AssertionFailedError: The test case is a prototype.  
586     at org.junit.jupiter.api.AssertionsUtils.fail(AssertionsUtils.java:39)  
587     at org.junit.jupiter.api.Assertions.fail(Assertions.java:109)  
588     at it.unipi.firstapp.FirstAppTest.testMain(FirstAppTest.java:58)  
589 Results :  
590 Failed tests: it.unipi.firstapp.FirstAppTest.testMain(): The test case is a prototype.  
591 Tests run: 2, Failures: 1, Errors: 0, Skipped: 0  
592 [INFO] -----  
593 [INFO] BUILD FAILURE  
594 [INFO] -----  
595 [INFO] Total time: 14.995 s  
596 [INFO] Finished at: 2022-09-09T14:20:23Z  
597 [INFO] -----  
598 [ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.1  
599 2.4:test (default-test) on project FirstApp: There are test failures.  
600 [ERROR]  
601 [ERROR] Please refer to /builds/provaapp/applicazione-di-prova/target/surefire-rep  
602 orts for the individual test results.  
603 [ERROR] -> [Help 1]  
604 [ERROR]  
605 [ERROR] To see the full stack trace of the errors, re-run Maven with the -e switc  
606 h.  
607 [ERROR] Re-run Maven using the -X switch to enable full debug logging.  
608 [ERROR]  
609 [ERROR] For more information about the errors and possible solutions, please read  
610 the following articles:  
611 [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureExcept  
612 ion  
613 Cleaning up project directory and file based variables  
614 ERROR: Job failed: exit code 1
```

Per risolvere il problema commentiamo la funzione fail nel testMain e poi effettuiamo il push sul nostro repository.

```
@Test
public void testMain() throws Exception {
    System.out.println("main");
    String[] args = null;
    FirstApp.main(args);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}
```

passed 00:02:51 1 minute ago Merge origin/master #635901571 master ffe13416 latest

## Merge origin/master

3 jobs for master in 2 minutes and 51 seconds

latest

ffe13416

No related merge requests found.

Pipeline Needs Jobs 3 Tests 0

build	test	deploy
build-job	unit-test-job	deploy-job

```
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.426 sec
Results :
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.686 s
[INFO] Finished at: 2022-09-09T14:24:20Z
[INFO] -----
$ sleep 60
$ echo "Code coverage is 90%"
Code coverage is 90%
Cleaning up project directory and file based variables
Job succeeded
```

## LOG4J

Il log fatto attraverso delle stampe a video non e' la soluzione migliore nei grandi progetti, che hanno bisogno di un sistema di log scalabile e altamente configurabile a seconda delle esigenze e delle situazioni, e.g. log su file o a video, log dettagliato o solo superficiale.

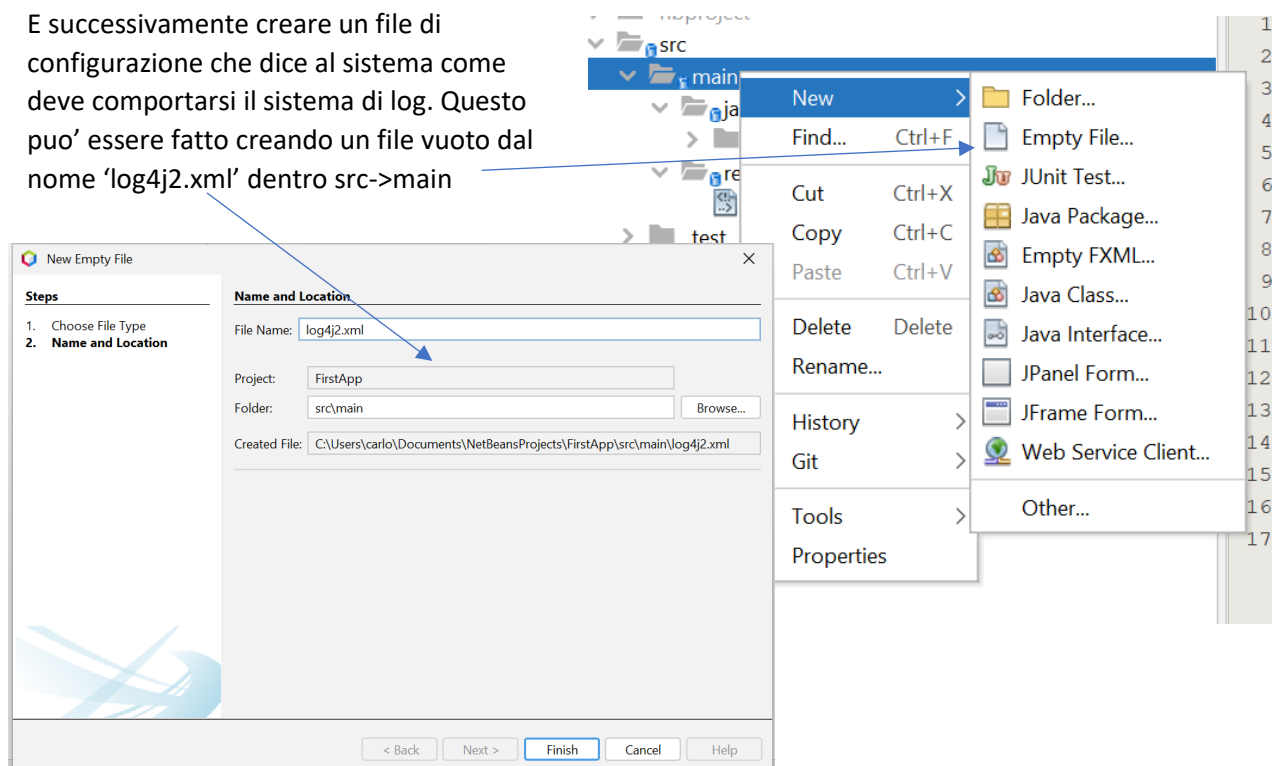
Una delle liberie di log piu' usate in java e' Log4J <https://logging.apache.org/log4j/2.x/>

Tale libreria permette di organizzare il log su piu' livelli di interesse: DEBUG, INFO, WARN, ERROR, FATAL<sup>2</sup>

Per usare tale libreria nel progetto bisogna aggiungere tale dipendenza nel progetto:

```
1. <dependency>
2.   <groupId>org.apache.logging.log4j</groupId>
3.   <artifactId>log4j-api</artifactId>
4.   <version>2.16.0</version>
5. </dependency>
6. <dependency>
7.   <groupId>org.apache.logging.log4j</groupId>
8.   <artifactId>log4j-core</artifactId>
9.   <version>2.16.0</version>
10. </dependency>
11.
```

E successivamente creare un file di configurazione che dice al sistema come deve comportarsi il sistema di log. Questo puo' essere fatto creando un file vuoto dal nome 'log4j2.xml' dentro src->main



<sup>2</sup> Per maggiori dettagli sui livelli: <https://javapapers.com/log4j/log4j-levels/>

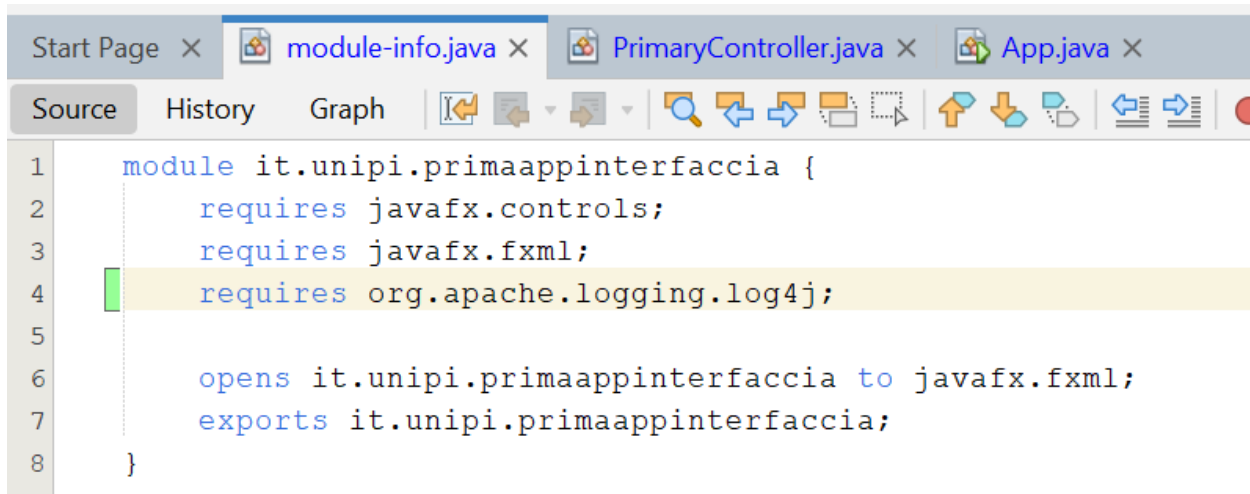


Un file di configurazione Log4j è il seguente:

```
1. <Configuration status="INFO">
2.   <Appenders>
3.     <Console name="ConsoleAppender" target="SYSTEM_OUT">
4.       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
5.     </Console>
6.     <File name="FileAppender" fileName="application-${date:yyyyMMdd}.log"
7.       immediateFlush="false" append="true">
8.       <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} -
9.       %msg%n"/>
10.    </File>
11.  </Appenders>
12.  <Loggers>
13.    <Root level="debug">
14.      <AppenderRef ref="ConsoleAppender" />
15.      <AppenderRef ref="FileAppender"/>
16.    </Root>
17.  </Loggers>
18. </Configuration>
```

In questo caso il log viene sia mostrato a video che salvato su file ad ogni esecuzione. Il file di configurazione permette di specificare il nome del file e il livello da mostrare.

In alcuni progetti c'è bisogno dove presente di modificare il file "module-info.java" presente nella cartella main/java/it aggiungendo la seguente riga con la dipendenza a Log4j:



The screenshot shows an IDE with three tabs: "Start Page", "module-info.java", and "PrimaryController.java". The "module-info.java" tab is active, showing the following code:

```
1 module it.unipi.primaappinterfaccia {
2     requires javafx.controls;
3     requires javafx.fxml;
4     requires org.apache.logging.log4j;
5
6     opens it.unipi.primaappinterfaccia to javafx.fxml;
7     exports it.unipi.primaappinterfaccia;
8 }
```

Una volta che la libreria è configurata si può usare all'interno del programma. In ogni classe bisogna recuperare un puntatore al logger (riga 1) che poi può essere utilizzato per stampare messaggi di livello differente (righe 6-11).

```
1.     private static final Logger logger = LogManager.getLogger(FirstApp.class);
2.
3.     public static void main(String[] args) throws IOException {
4.
5.         logger.trace("We've just greeted the user!");
6.         logger.debug("We've just greeted the user!");
7.         logger.info("We've just greeted the user!");
8.         logger.warn("We've just greeted the user!");
9.         logger.error("We've just greeted the user!");
10.        logger.fatal("We've just greeted the user!");
11.
12.    }
```

Il logger deve essere creato in ogni classe. Per facilitare la creazione si può usare la funzione "Insert Code->Logger" di NetBeans

### Esercizio

Modificare il programma in modo da stampare utilizzando la libreria Log4J.

