

DA339A Laboration L2

Syfte

Syftet med denna laboration är att introducera hur ett program körs och hur en enskild kodrad kan ändras för att skapa en annan typ av lösning. Slutligen behandlas det sista steget i versionskontroll, det vill säga GitHub.

Följande tas upp i denna labb:

- Kompilering
- Exekvering
- Redigering av kod
- Versionshantering i GitHub

Redovisning

Laboration L2 ska inte redovisas.

Förberedelser

- Bygger på information från samtliga föreläsningar till och med F2 samt kurslitteratur
- Ladda ner Java.
 - Mac: <https://www.oracle.com/java/technologies/downloads/#jdk21-mac>
 - För äldre Mac än M1 chippet *välj x64 DMG Installer*
 - För Mac med M1 chip och senare versioner *välj ARM64 Installer*
 - Windows: <https://www.oracle.com/java/technologies/downloads/#jdk21-windows>
 - Välj x64 Installer
- Om inte redan gjort i föregående laboration bör katalog för denna laboration skapas på samma plats som tidigare labbar
 - Alla nedladdade filer tillhörande laboration läggs i katalogen för denna laboration

Installera JDK

JDK står för Java Development Kit och behövs för att kompilera och exekvera kod lokalt på en dator. Om en JDK inte redan installerats gå till länken ovan för det operativsystem som används. Var noggrann med att versionen av JDK är **Java 21** och välj en lämplig installer enligt instruktionerna ovan.

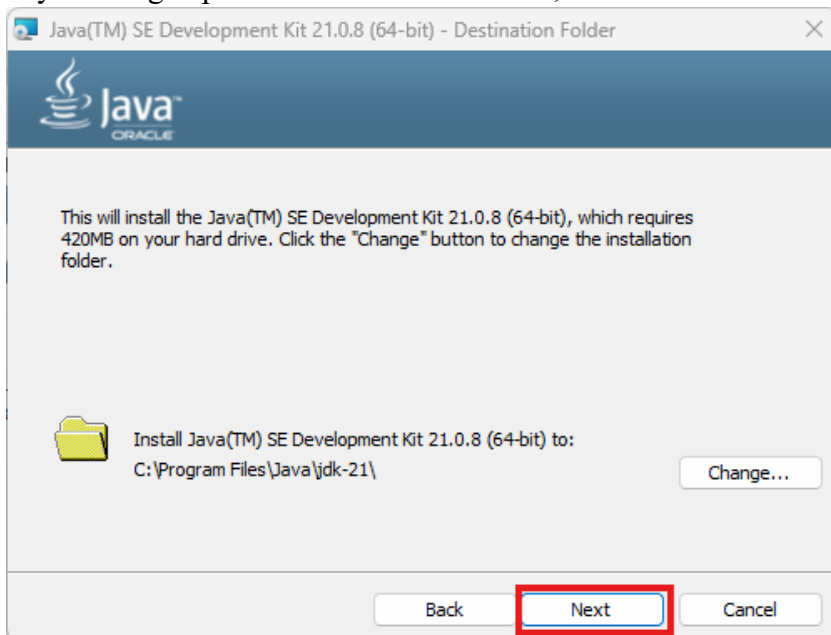
Installering för Windows

Dubbelklicka på den nedladdade installern. En ruta som kan kräva administrativa rättigheter kan dyka upp – godkänn denna dialogruta. Därefter visas en ny ruta, där du trycker på *Next*.



Figur 1: Första sidan för installation av JDK

Tryck återigen på *Next* när nästa sida visas, denna sida visar var JDK installeras.



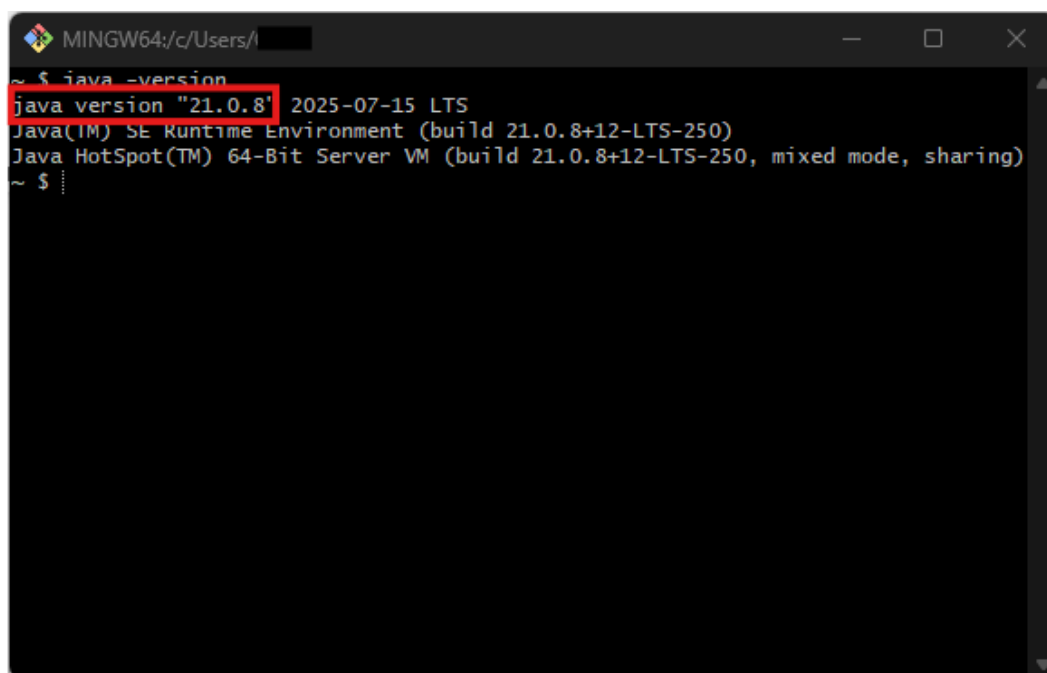
Figur 2: Installationsplats för SDK

I nästa ruta sker en kort installation, varefter den sista rutan för JDK-installationen visas. JDK är nu installerat och *Close* kan tryckas på. Om man vill kan man även klicka på *Next Steps* för att läsa mer om JDK.



Figur 3: Installation klar.

För att kontrollera att JDK har installerats korrekt, öppna Git Bash och skriv in *java -version*. Det bör nu visas text som anger att JDK 21 är installerat på datorn. Om inget visas kan det bero på att Git Bash var öppet i bakgrunden och behöver startas om.



Figur 4: Verifiering att Java 21 är i.

Nu är JDK installerat på datorn, och det är dags att gå vidare med att introducera hur en fil byggs upp och körs på datorn.

Kompilering

När en fil är färdig att köras består den fortfarande av text som människor kan läsa, så kallad källkod, men som datorn inte förstår. Kompilering översätter denna källkod till maskinkod som datorn kan exekvera. Under kompileringen kontrollerar kompilatorn att koden följer rätt syntax för det aktuella språket. Om något är fel rapporteras detta, och koden måste rättas innan den kan kompileras igen och köras. När kompileringen lyckas skapas en ny fil med ändelsen `.class`, som innehåller bytekod.

I Java används följande kommando för att kompilera en fil:

```
javac filnamn.java
```

Exekvering

Efter en lyckad kompilering skapas filen med ändelsen `.class`, som beskrivits ovan. Denna fil körs av Javas virtuella maskin (JVM), som översätter bytekoden till instruktioner som processorn kan förstå. Dessa instruktioner kallas binärkod.

I Java anges följande kommando för att exekvera en fil:

```
java filnamn
```

OBS! Filändelsen behöver inte läggas till, men däremot måste det finnas en `.class`-fil för motsvarande angivet filnamn.

GitHub

Detta bygger vidare på den kunskap som förmedlades under Laboration 1. Om denna laboration av någon anledning inte har genomförts, bör man återgå till Laboration 1 innan detta steg påbörjas.

Skapa konto

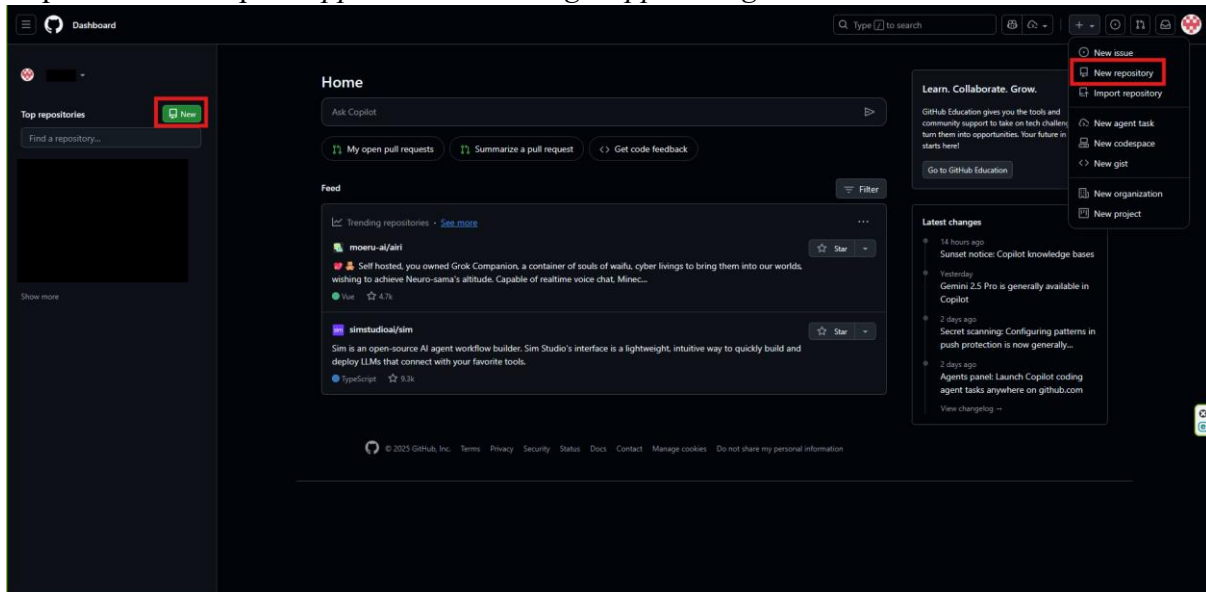
Till att börja med behöver ett konto skapas på [GitHub](https://github.com), vilket görs genom att registrera sig på webbplatsen. Registreringen är enkel och kan göras manuellt eller kopplas till ett Google-konto. Glöm inte att skapa ett lösenord som är lätt att komma ihåg men svårt att gissa. Det bästa är att använda en lösenordshanterare.

Figur 5: Registreringsformulär på [GitHub](https://github.com).

OBS! GitHub kan kräva användning av Microsoft Authenticator. Följ i så fall anvisningarna för att installera applikationen och påbörja användningen.

Skapa ett repository

Vid inloggning på GitHub visas den personliga dashboarden. Det finns två sätt att skapa ett nytt repository (repo) via denna dashboard: tryck antingen på den *gröna knappen New* under Top Repositories eller på *knappen med ett + längst upp till höger*.



Figur 6: GitHub dashboard.

Klicka på något av alternativen för att skapa ett repo. När ett repo skapas är det bra att överväga följande:

- Namn: Kan vara vad som helst, men välj ett namn som tydligt beskriver repots syfte. Detta används för att identifiera projektet.
- Synlighetsgrad: Public innebär att alla kan se repot, medan Private begränsar åtkomsten till dig och de du bjuder in.
- Licens: Bestämmer hur andra får använda koden. Exempelvis MIT tillåter fri användning, kopiering och modifiering.
- .gitignore: En fil som anger vilka filer Git ska ignorera, t.ex. temporära filer eller konfigurationsfiler.
- README: En textfil som ofta innehåller information om projektet, exempelvis syfte, installation och användning. I detta steg skapas den inte ännu.

Fyll i dessa detaljer och tryck på *Create repository*. Det nya repot skapas och en ny sida öppnas – den specifika dashboarden för just det nyskapade repot

Figur 7: Skapande av ett repo.

I det nyskapade repot går det just nu att se namnet, den valda licensen samt att GitHub uppmanar till att skapa en README. Detta ska dock göras senare i labben.

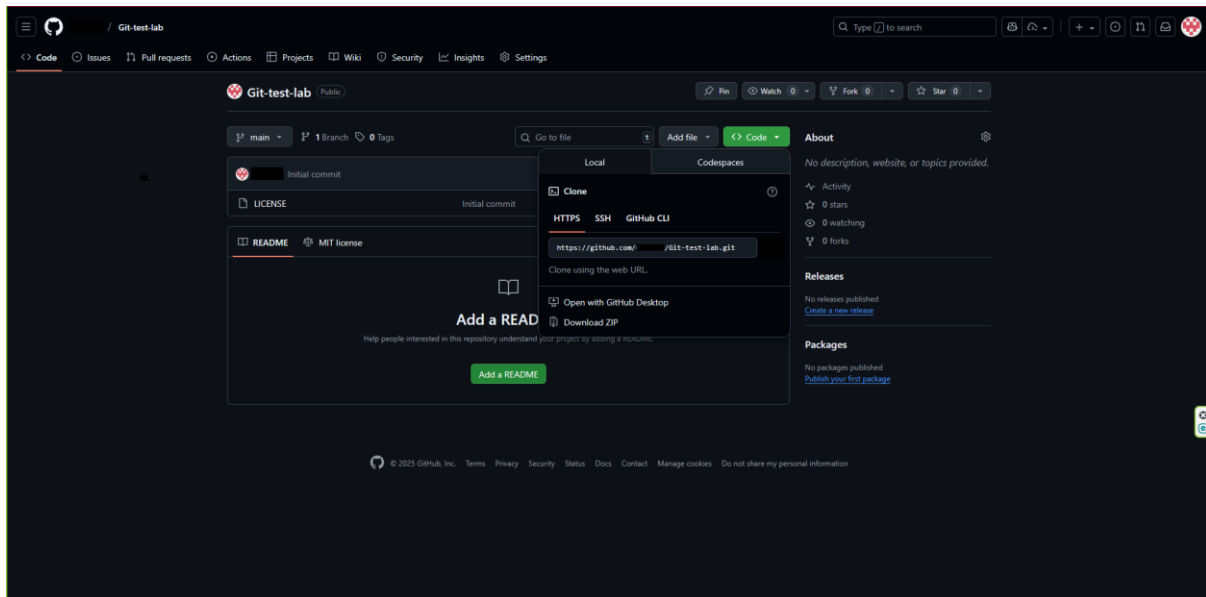
Klona ett repository

Att klona ett repo innebär i praktiken att ta en kopia av ett repo som finns på exempelvis GitHub och spara en lokal version på datorn. När detta görs får datorn rättigheter att både hämta och publicera (endast om det är ett publikt repo med rätt inställningar) mot samma repo på GitHub.

I Laboration 1 skapades ett lokalt repo av katalogen Laborationer. För att undvika problem med GitHub under denna laboration, *skapa först en katalog i DA339A som döps till GitHub*.

Det finns flera sätt att klona ett repo, men det enklaste är att göra allt via Git Bash eller Terminal, förutsatt att git är installerat på datorn. *Navigera till den nyskapade katalogen GitHub i DA339A*.

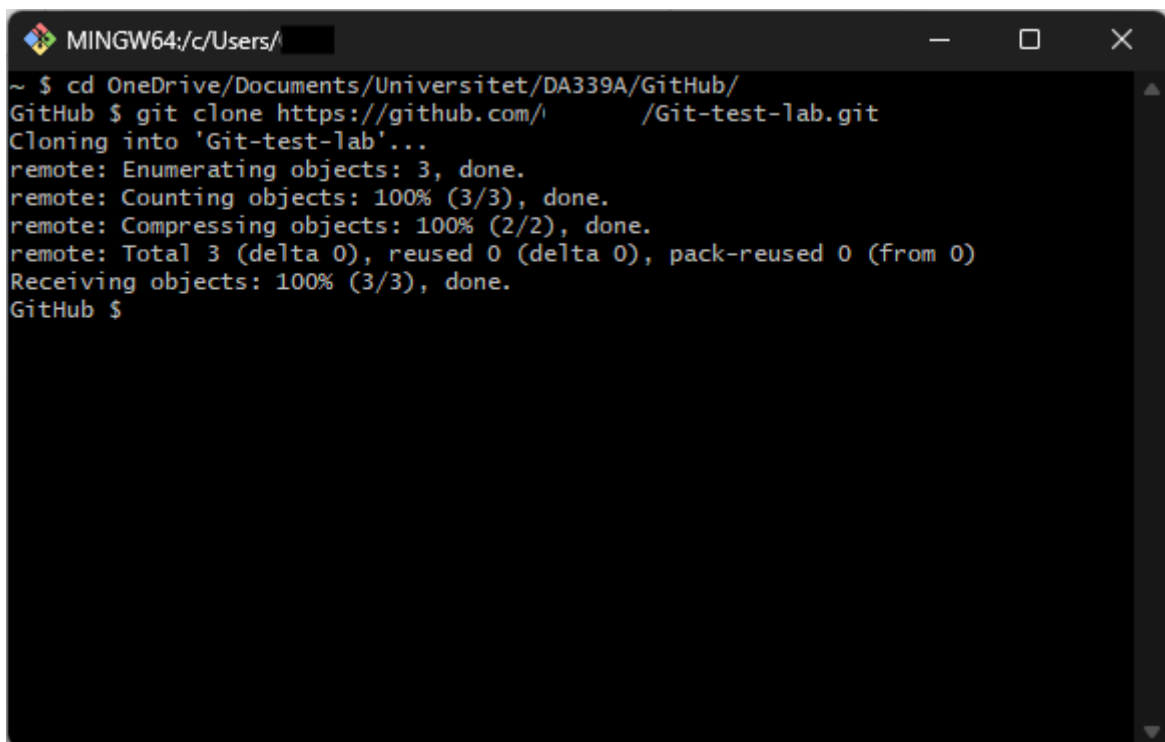
Gå sedan till ditt repo på GitHub och kopiera dess URL. Detta görs genom att trycka på knappen Code, se till att HTTPS är valt och kopiera antingen adressen manuellt eller genom att använda ikonen med två överlappande rutor.



Figur 8: Kopiering av repo-adress.

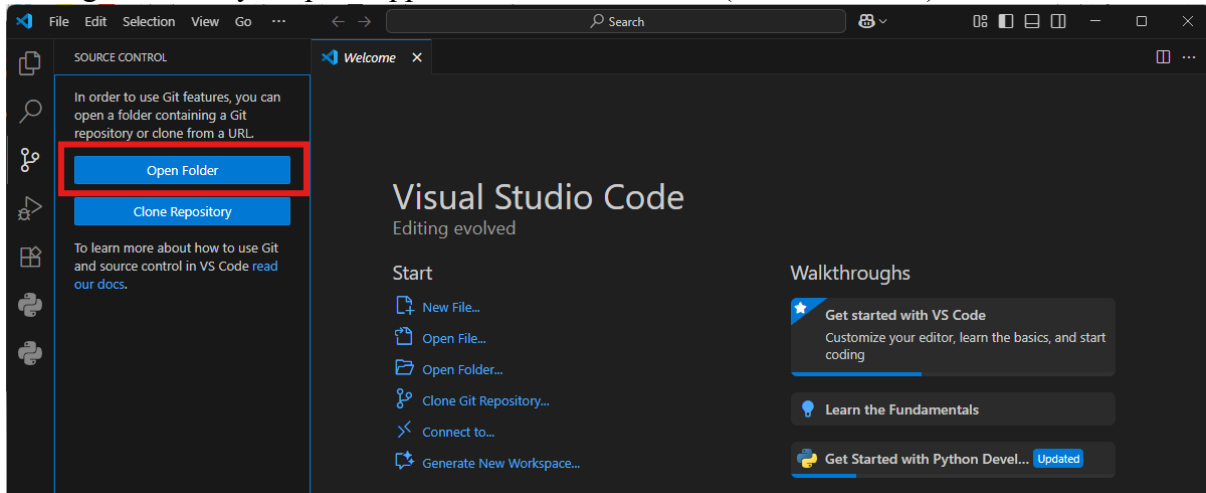
Med repo-adressen tillgänglig och Git Bash/Terminal stående i katalogen GitHub under DA339A, skriv följande kommando:

git clone "kopiera in repo-adress"



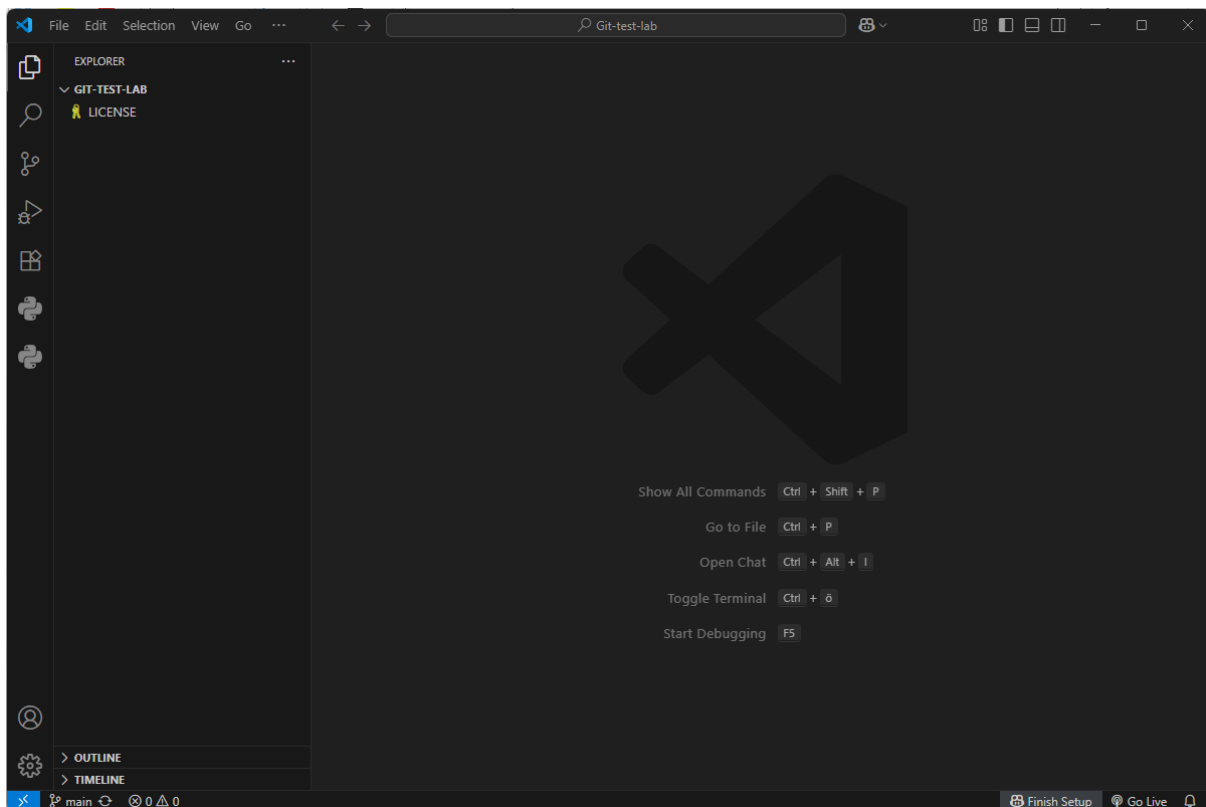
Figur 9: Kloning av GitHub repo.

När kloningen är klar bör resultatet se ut ungefär som ovan. Öppna sedan det nya repot i VS Code genom att trycka på knappen för versionskontroll (Source Control).



Figur 10: Öppna GitHub repo genom versionskontroll i VS Code.

Om allt gått bra bör det se ut som bilden nedan.

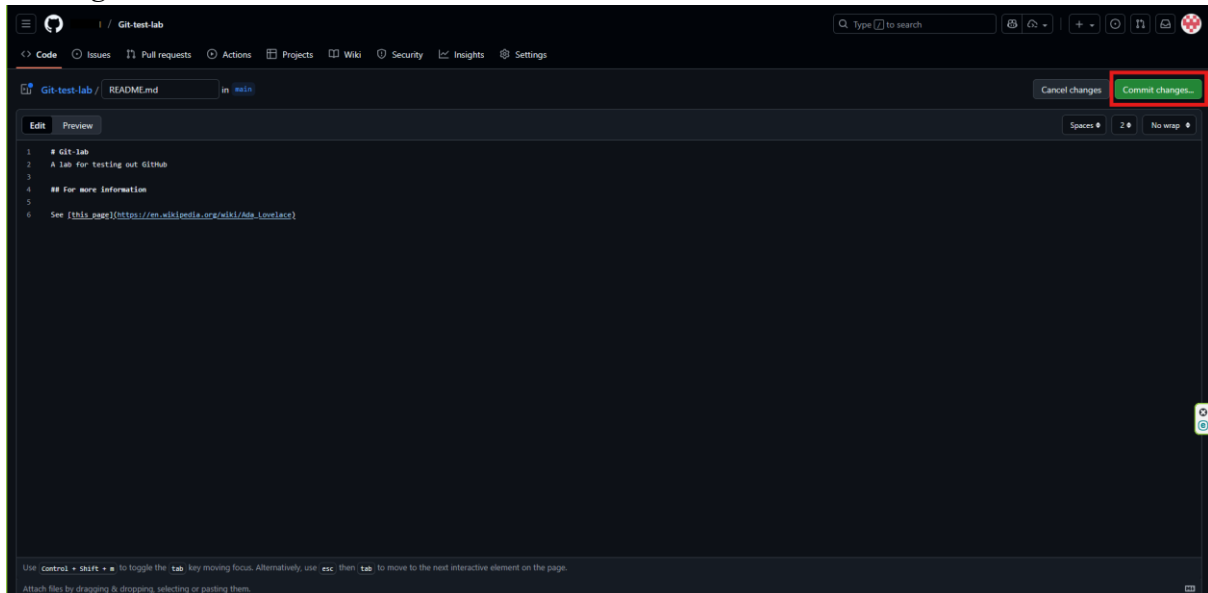


Figur 11: GitHub repo öppnat i VS Code.

Skapa fil och ladda ned den

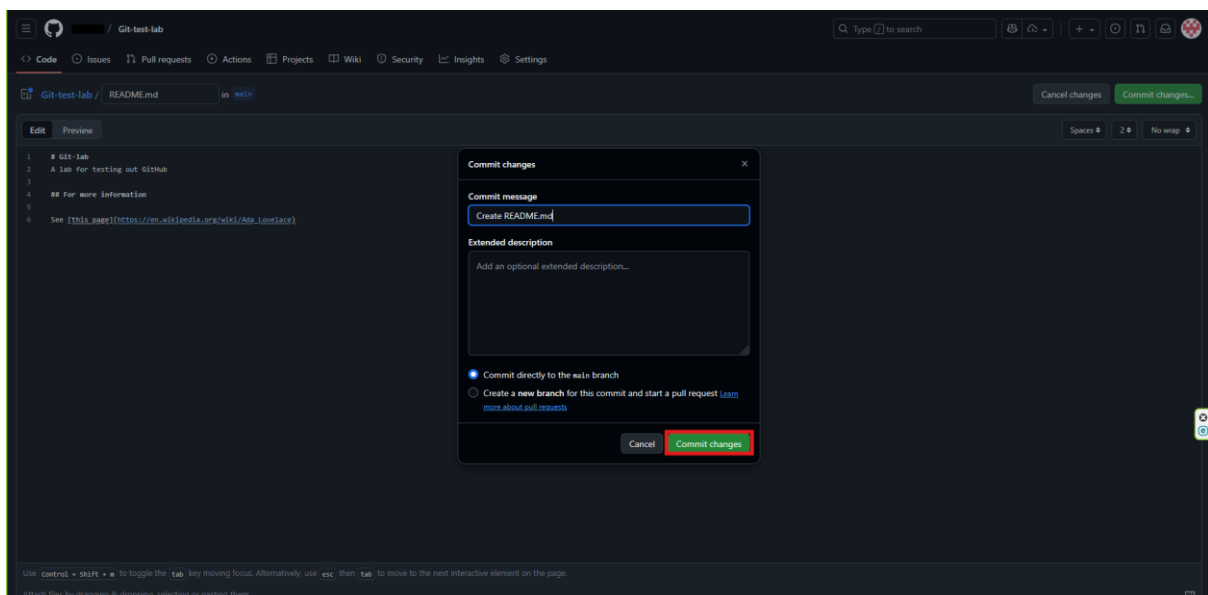
Tidigare i labben föreslog GitHub att en README borde skapas, vilket är en bra idé för alla typer av projekt. Gå nu tillbaka till GitHub och tryck på knappen *Add a README*.

I en README är det viktigt att tydlig information om projektet presenteras, så ange relevant och informativ text i filen. *När informationen är färdigskriven, checka in (commit) ändringarna.*



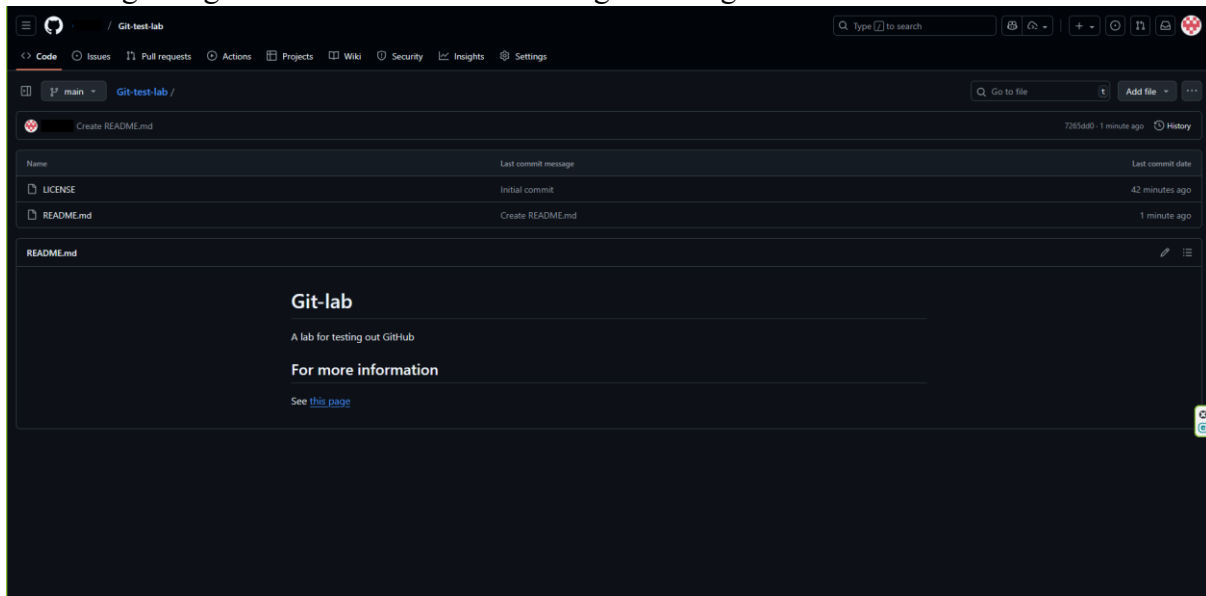
Figur 12: Skrivna README.

När något checkas in (commit) i GitHub måste alltid ett incheckningsmeddelande anges. För en README har GitHub ett förvalt meddelande, vilket gör att ingen ändring behöver göras. *Checka in ändringarna genom att trycka på Commit changes.*



Figur 13: Incheckning av README.

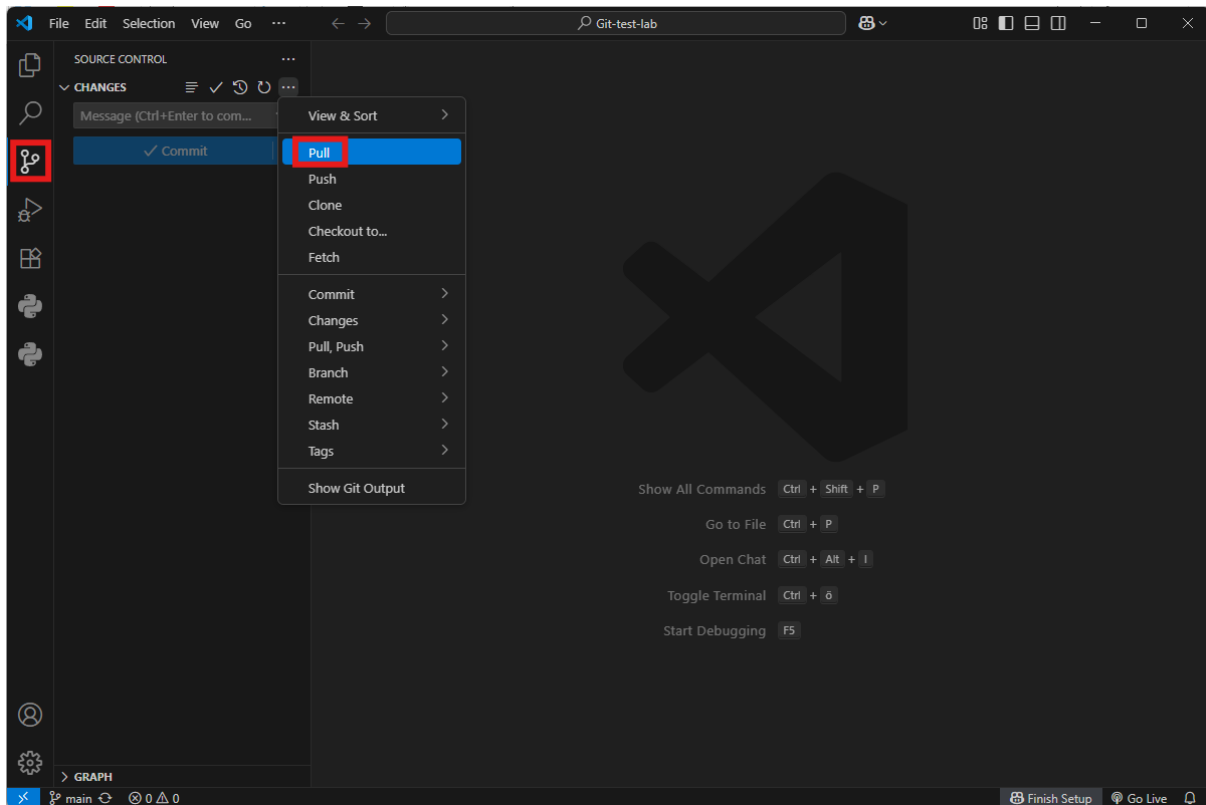
Tillbaka på dashboarden för repot syns nu den nyskapade README-filen samt en ny revidering. Längst ned finns även en förhandsgranskning av filens innehåll.



Figur 14: Uppdaterade dashboard med README.

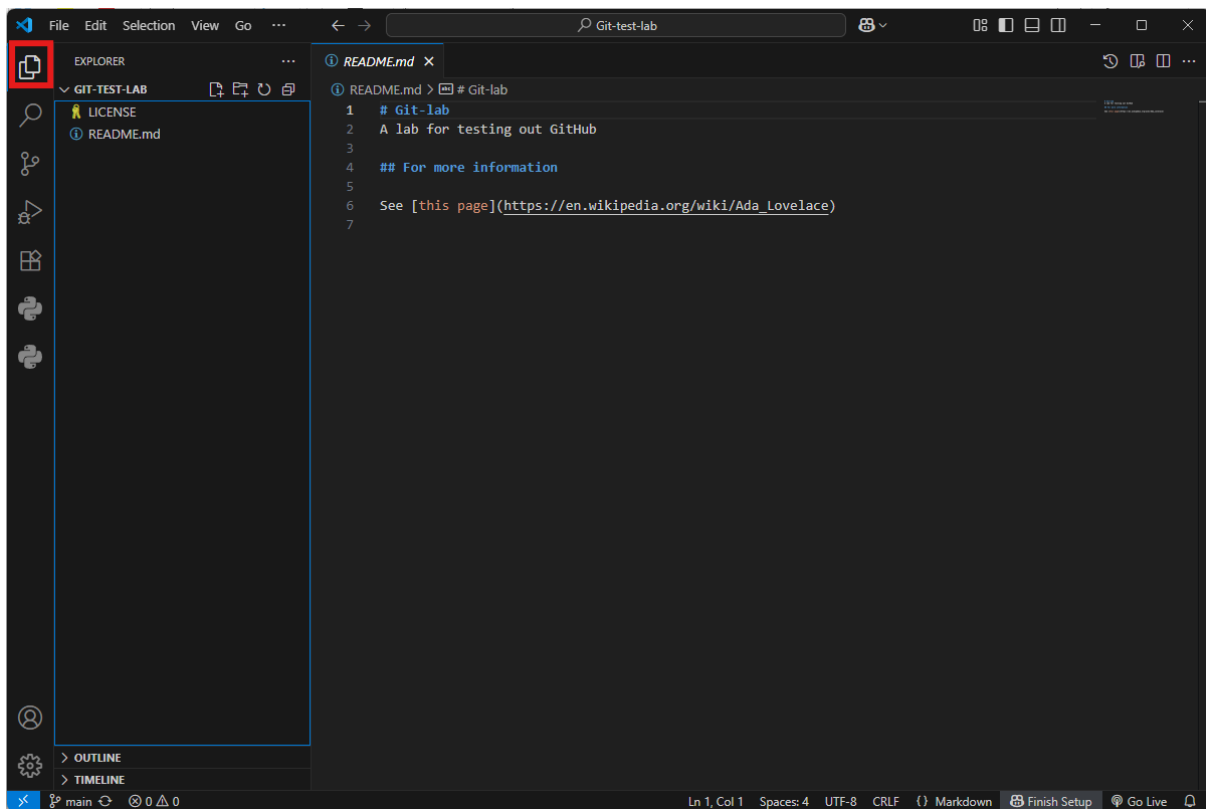
Men i VS Code finns ännu ingen README-fil, endast LICENSE syns. För att åtgärda detta används kommandot `git pull`, som finns under trepunktsmenyn i versionskontroll. När detta kommando körs kontaktar Git GitHub för att kontrollera om några nya revideringar finns som inte redan finns lokalt och hämtar sedan dessa.

OBS! Som rutin: kör alltid *git pull* innan andra kommandon i versionskontroll. Detta sparar mycket huvudvärk på lång sikt.



Figur 15: Git pull i VS Code.

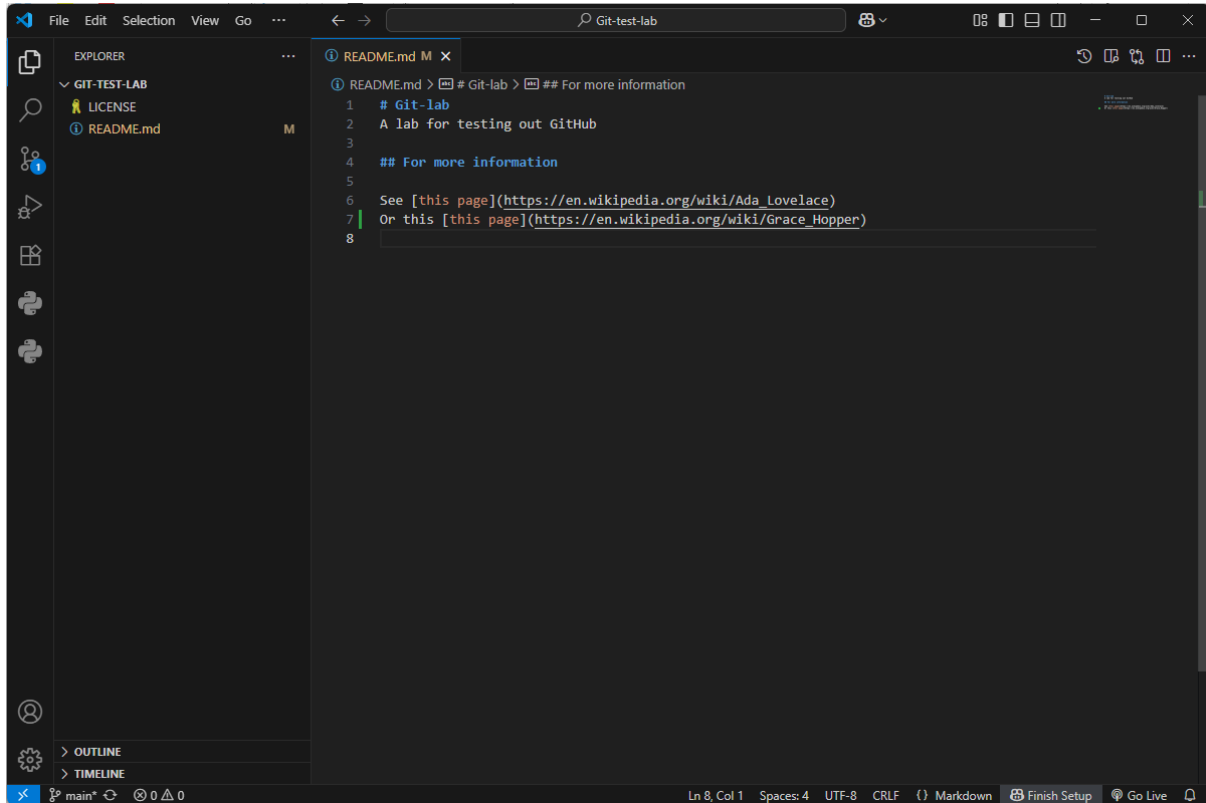
Det kan verka som att inget händer när git pull körs, men om filutforskaren öppnas genom att trycka på staplade dokumenten syns nu README-filen i det lokala repot.



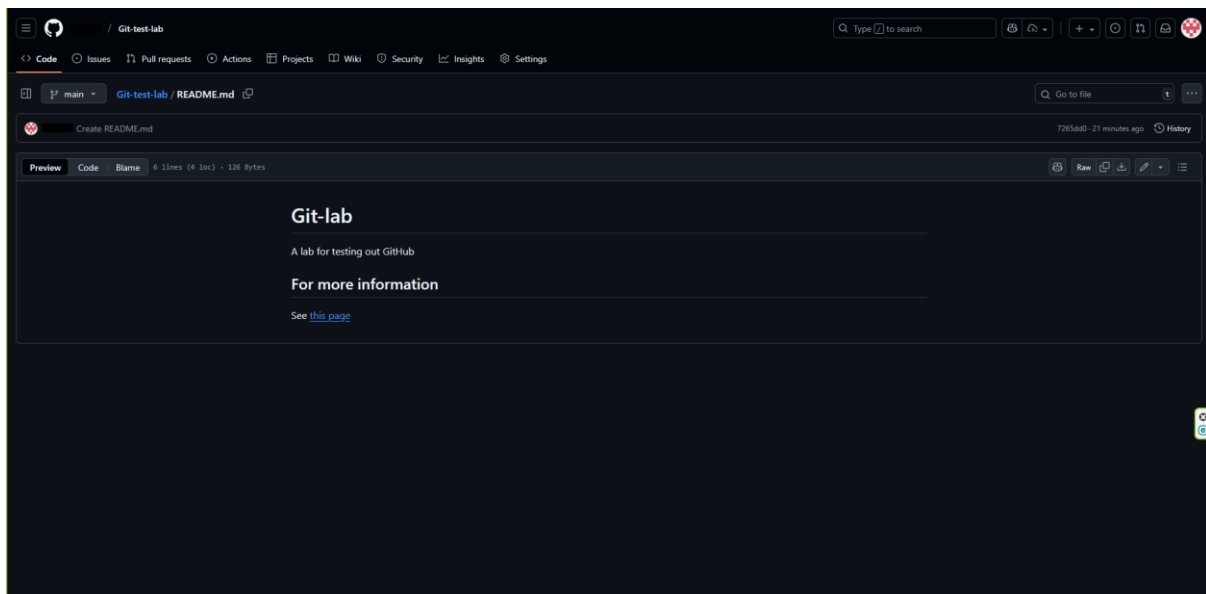
Figur 16: README-fil nedladdad till lokalt repo.

Uppdatera en fil och skicka upp den

Att ladda ner ny information från GitHub är enkelt, och samma princip gäller för att ladda upp lokala ändringar. För att kunna ladda upp krävs dock att filerna ändrats, eftersom både det lokala och det fjärrbaserade repot annars är synkade. *Börja med att öppna README-filen igen och lägg till ny information.* När filen sparas syns ändringen som ej hanterad lokalt och är ännu inte synlig på GitHub.

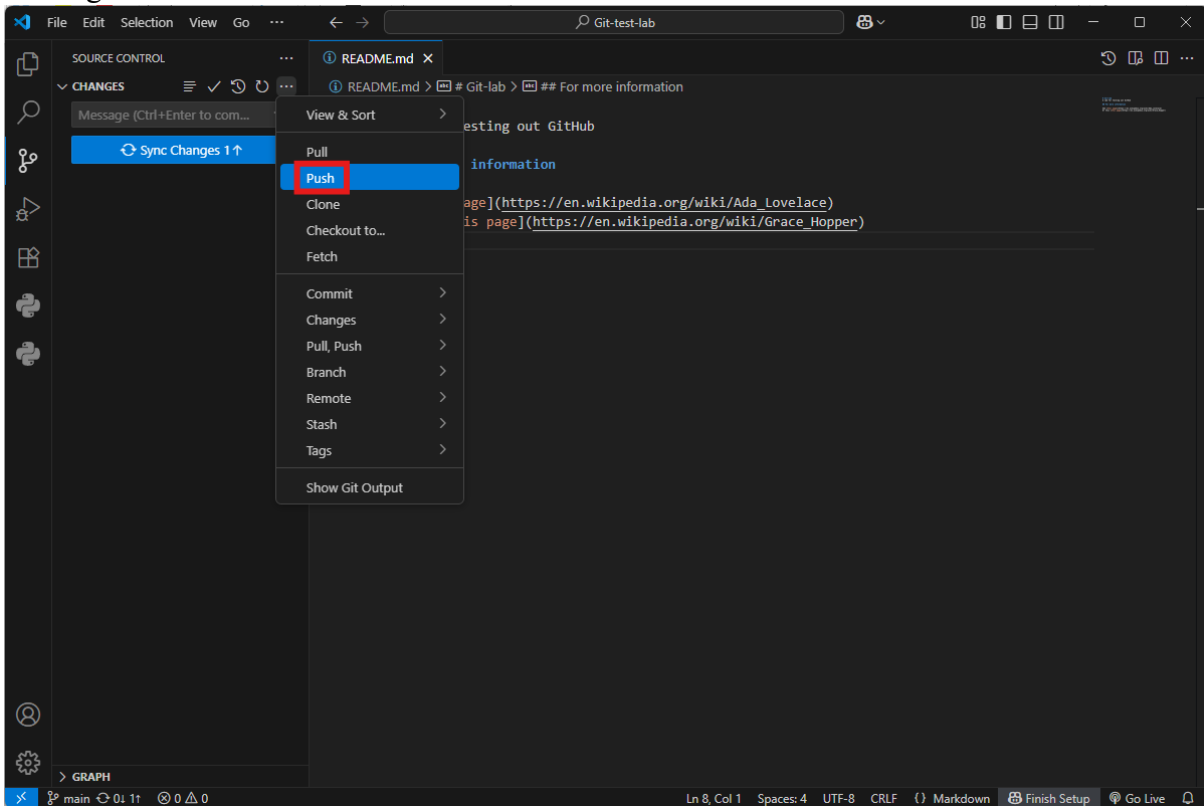


Figur 17: README med tillägg.



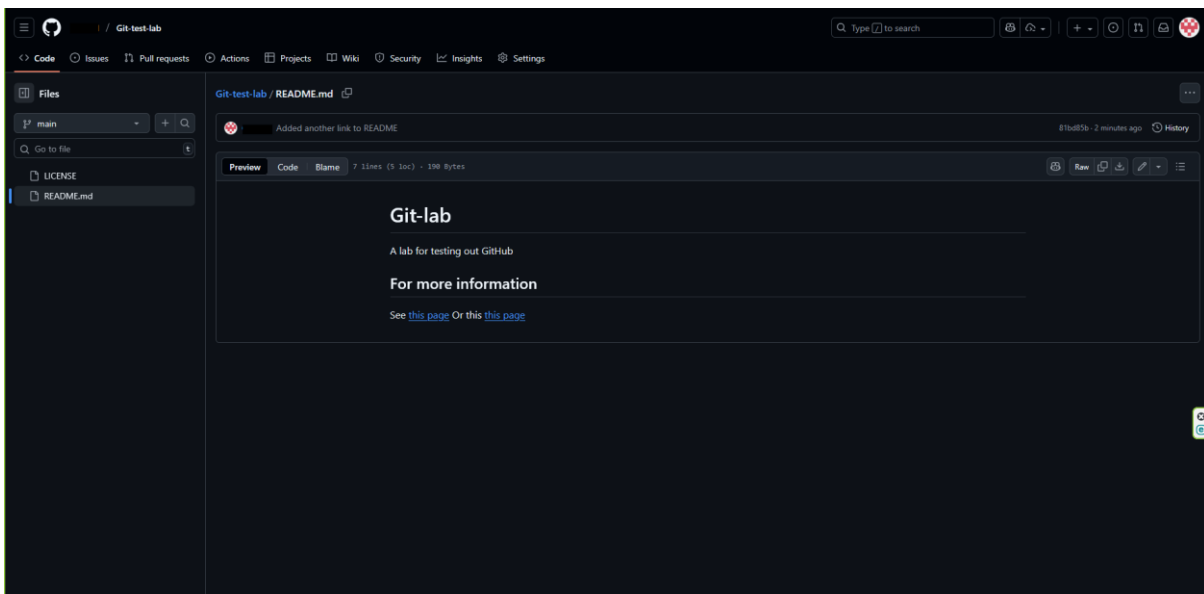
Figur 18: Ingen ändring för README-fil på GitHub.

Gå till versionskontrollen, *kör först git pull* för att säkerställa att inga ändringar gjorts på GitHub under arbetet med README. *Checka sedan in ändringen med ett lämpligt commit-meddelande*. Använd därefter trepunktsmenyn för att *köra git push*, vilket laddar upp de lokala ändringarna till GitHub.



Figur 19: Git push till GitHub utförs.

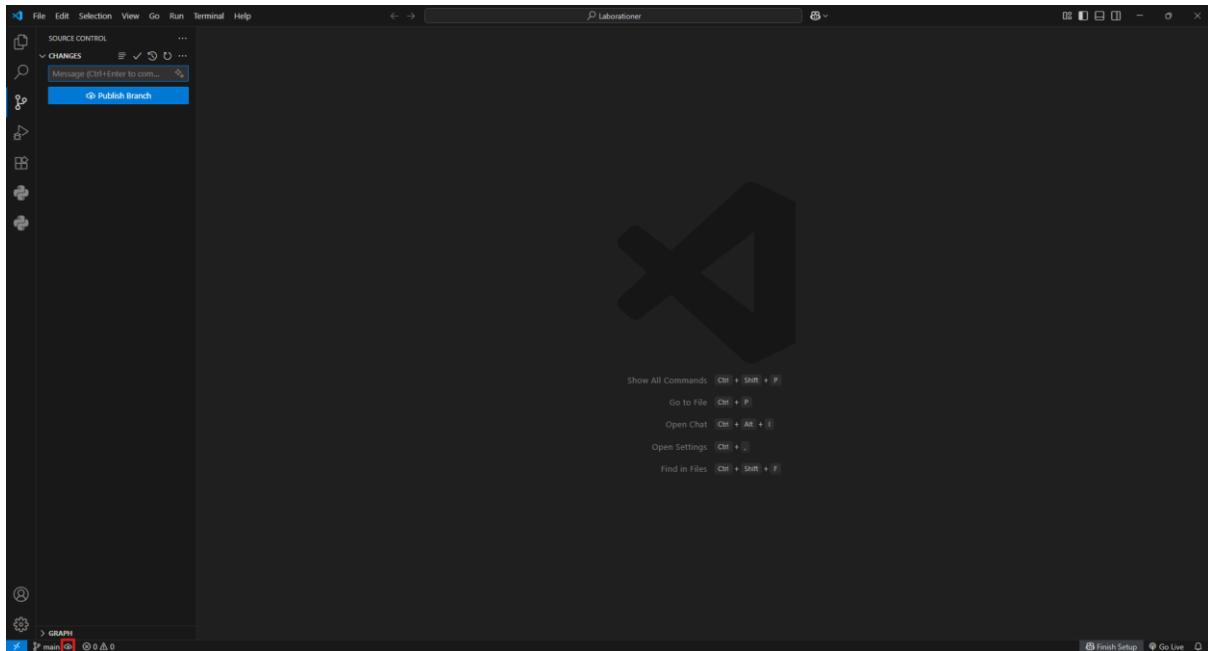
Ändringarna i README-filen är nu synliga även på GitHub, vilket visar att **git push** lyckades och repot är uppdaterat.



Figur 20: Lokalt repo och GitHub repo i synk.

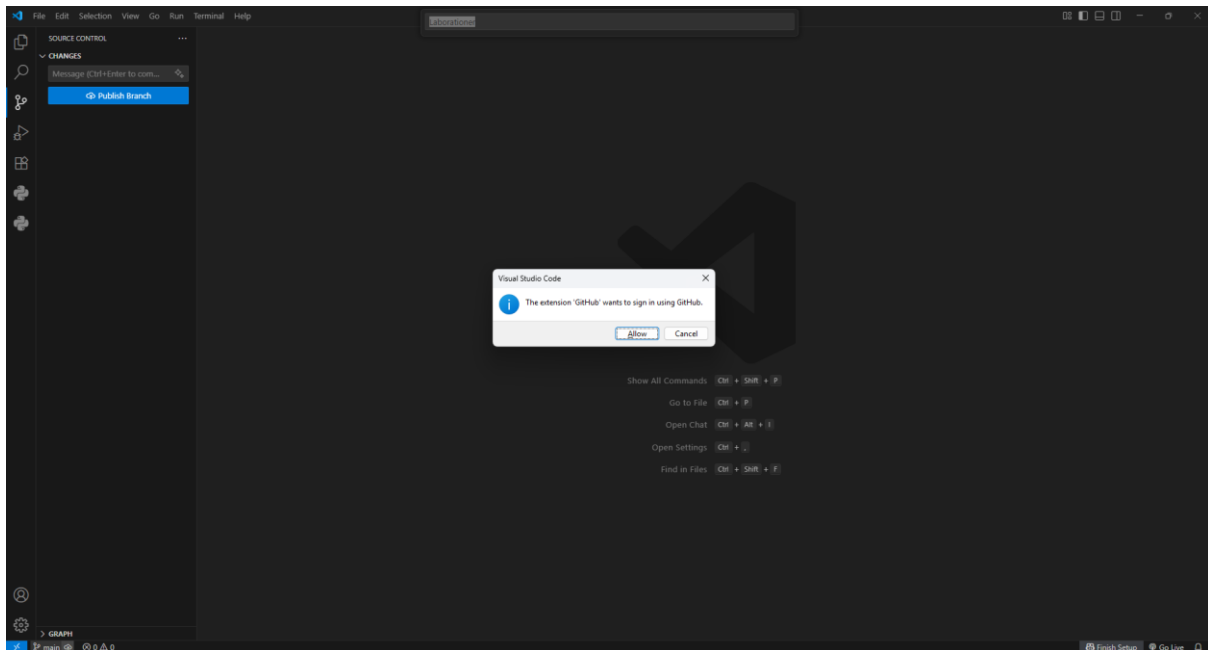
Publicera lokalt repo till GitHub

Det sista steget är att publicera det lokala Git-repot från Laboration 1 på GitHub. *Öppna katalogen Laborationer i VS Code, vilket öppnar repot som skapades under Laboration 1. Navigera till versionskontrollen och klicka på molnikonen bredvid main/master för att publicera repot.*



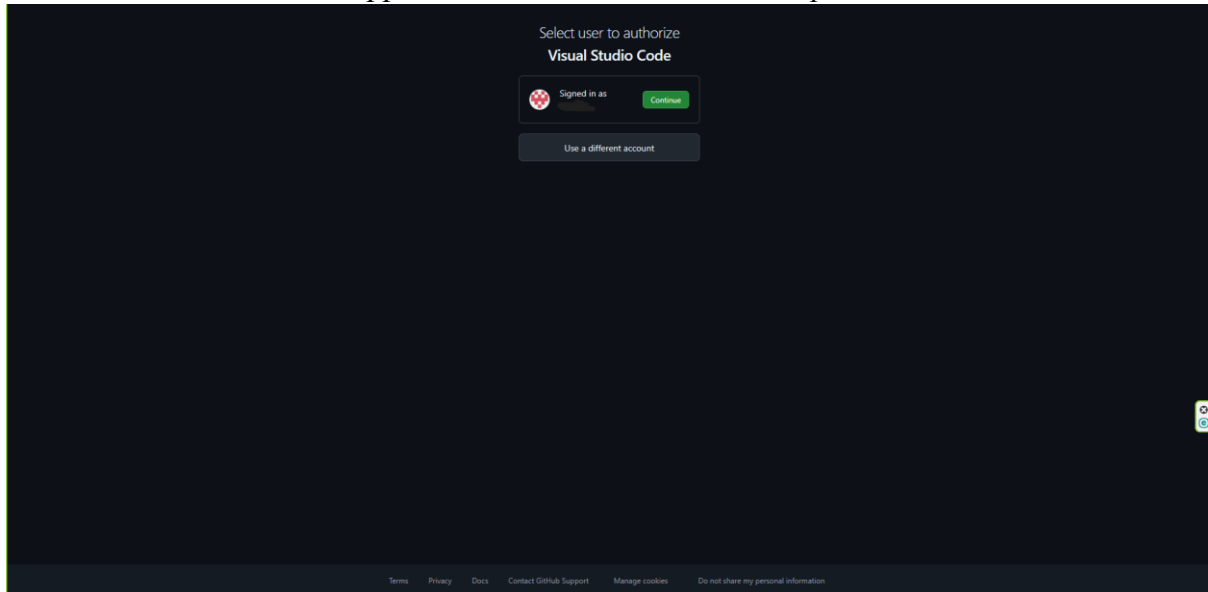
Figur 21: Synka lokalt repo till GitHub genom molnikonen.

Efter att molnikonen klickats öppnas en ruta som frågar om tillägget får ansluta till GitHub. Välj *Allow*.



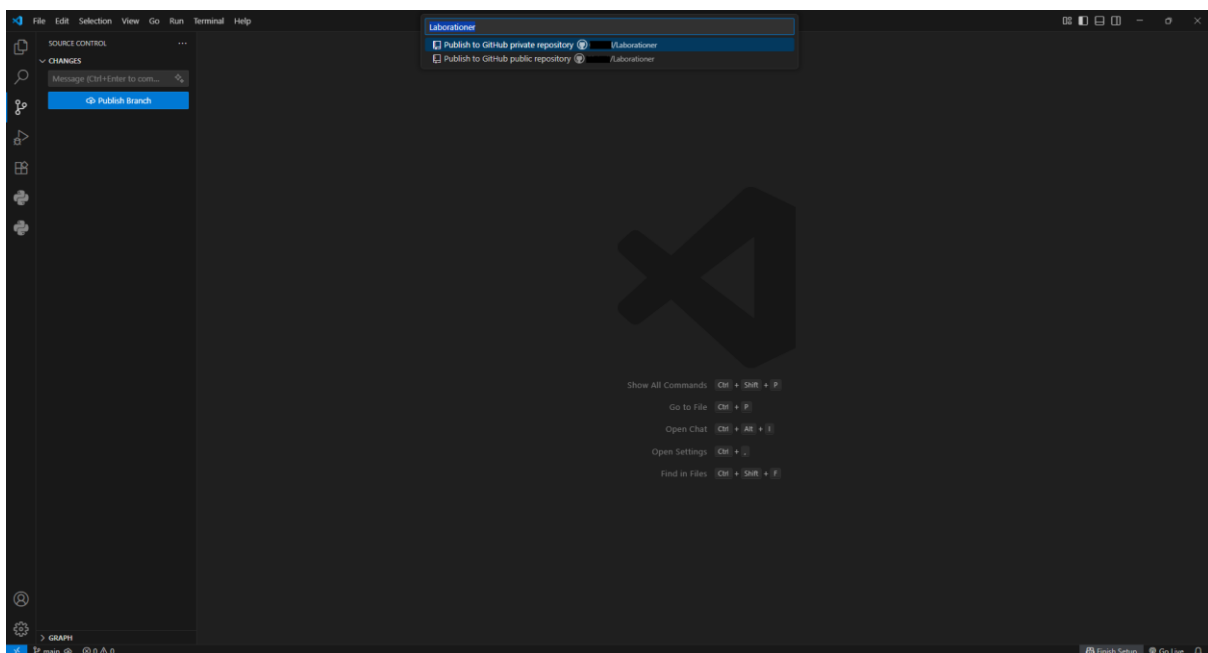
Figur 22: Tillåt tillägget att logga in till GitHub.

Den förvalda webbläsaren öppnas och nedan sida visas; *klicka på Continue*.

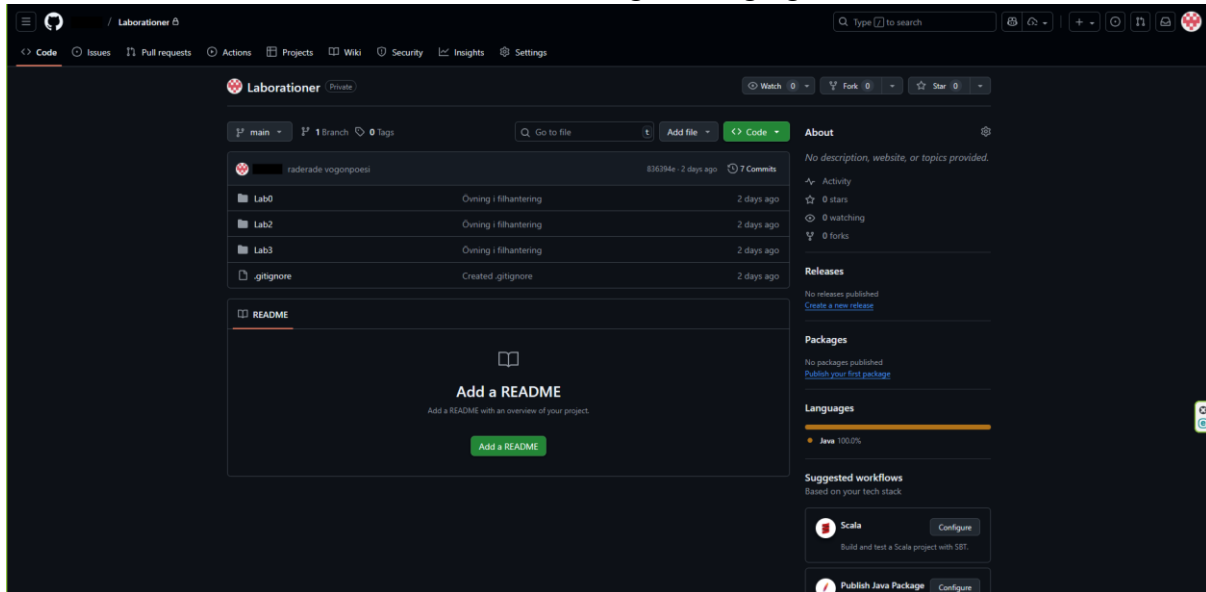


Figur 23: GitHub vill veta vilket konto som ska användas.

Efter att Continue valts i webbläsaren visas två alternativ i VS Code: public eller private repository. Om alternativen inte syns, klicka i sökfältet högst upp i VS Code. *Välj Private för repot.*



Efter detta kan repot hittas under Repositories på GitHub. När repot öppnas går det att se information om dess innehåll, och varje katalog kan öppnas för att visa filer. Lab1 syns inte eftersom den enda filen där är en lokal fil som lagts till i .gitignore.



Figur 24: Översikt av repot Laborationer.

Så länge versionshantering används korrekt kommer alla filer som skapas och ändras under labbar automatiskt att finnas tillgängliga på GitHub.

Uppgifter

Följande uppgifter bygger på det som introducerats ovan samt kunskap från tidigare labbar. Försök alltid att lösa uppgifterna själv först, fråga sedan en annan student eller labbassistent vid behov. Som sista alternativ kan facit konsulteras, men observera att facit visar en lösning och att flera olika lösningar ofta är möjliga.

Kompilering

Till denna labb följde det med ett flertal filer, navigera till katalogen där dessa filer sparades via *Git Bash* eller *Terminalen*. Den fil som ska kompileras heter *Lab2Excercise2*. Om kompilering lyckas ges ingen information i *Git Bash* eller *Terminalen*, men vad skapades efter en lyckad kompilering?

Exekvering

Filen är nu kompilerad och en .class-fil har skapats, detta betyder att filen nu kan exekveras. *Utför en exekvering på filen*, vad händer i *Git Bash* eller *Terminalen*?

Redigering

Bland labbfilerna finns *Lab2Excercise2Fel.java*, som innehåller ett eller flera fel. *Kompilera filen och läs noggrant igenom felmeddelandena*. Öppna sedan filen i *Visual Studio Code* och åtgärda felen enligt instruktionerna i meddelandena. När filen kompilerar korrekt ska den fungera på samma sätt som *Lab2Excercise2*.

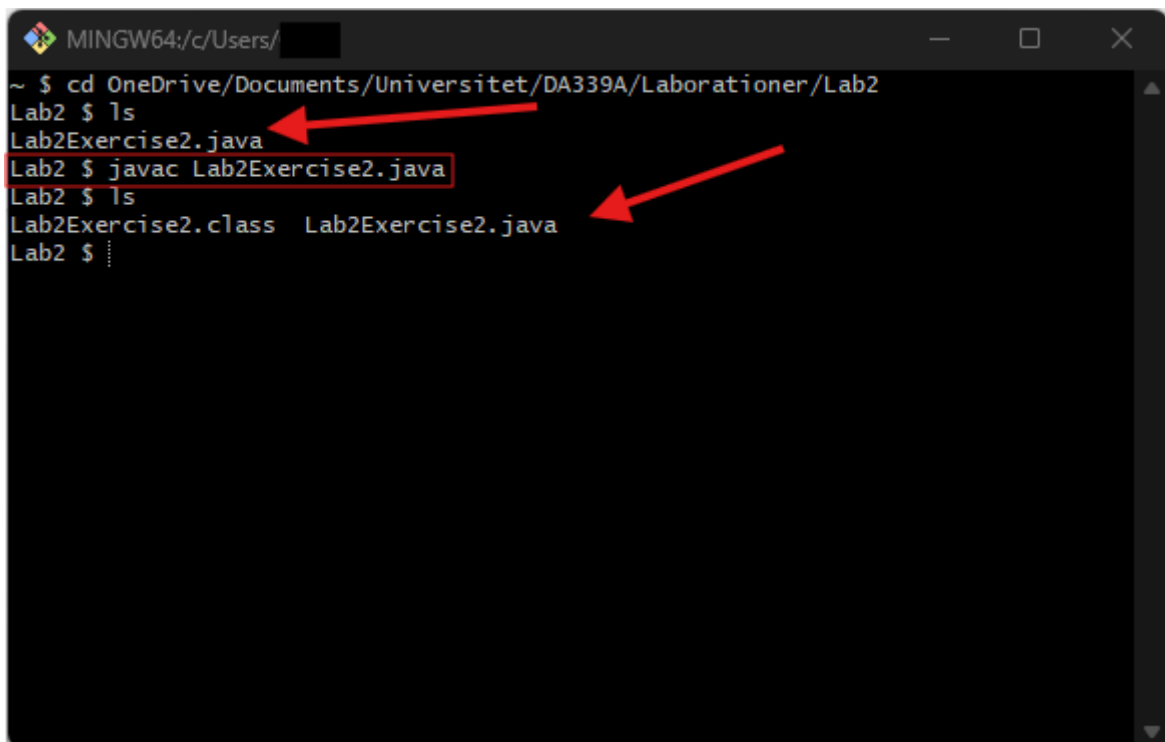
Skapa repo för inlämningar.

Följ instruktionerna för GitHub och skapa ett nytt repo med namnet Assignments. Detta repo används för att synkronisera alla individuella inlämningar.

Facit

Kompilering

Genom att köra kommandot `javac Lab2Exercise2.java` kompileras filen till en `.class`-fil. Efter kompileringen kan detta bekräftas genom att köra kommandot `ls`, där den nya `.class`-filen nu syns i katalogen.



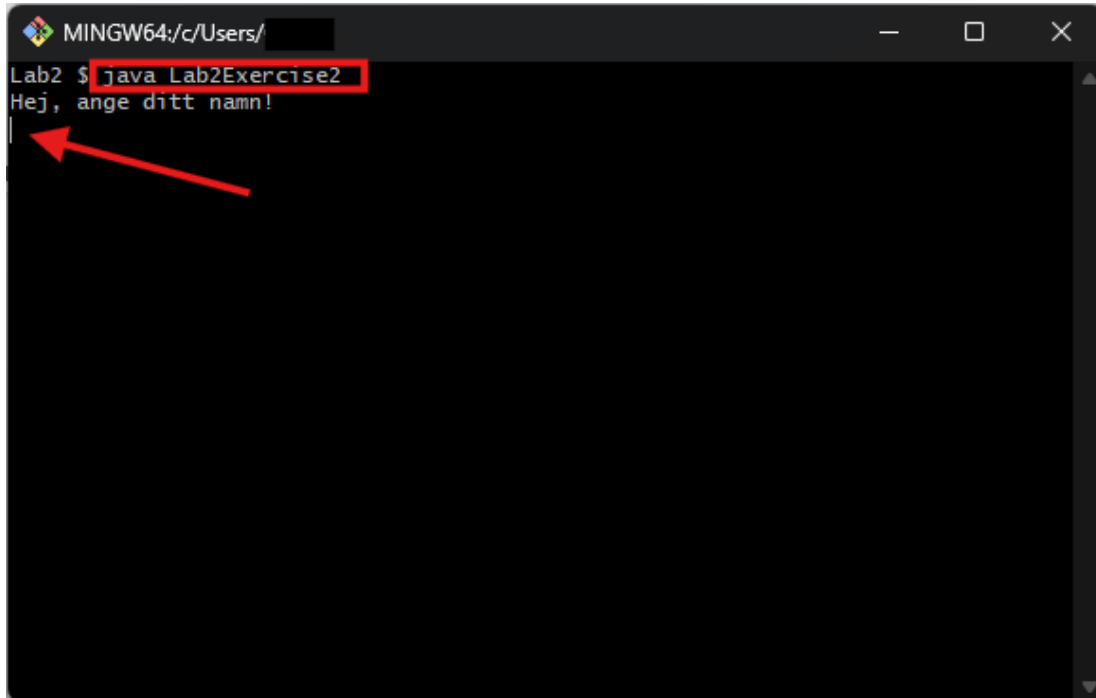
```
MINGW64:/c/Users/[redacted]
~ $ cd OneDrive/Documents/Universitet/DA339A/Laborationer/Lab2
Lab2 $ ls
Lab2Exercise2.java
Lab2 $ javac Lab2Exercise2.java
Lab2 $ ls
Lab2Exercise2.class  Lab2Exercise2.java
Lab2 $
```

The screenshot shows a terminal window with the following commands and output:
1. `cd OneDrive/Documents/Universitet/DA339A/Laborationer/Lab2`
2. `ls` shows `Lab2Exercise2.java`.
3. `javac Lab2Exercise2.java` is executed.
4. `ls` is run again, showing both `Lab2Exercise2.class` and `Lab2Exercise2.java`.
Red arrows point from the text in the preceding paragraph to the corresponding lines in the terminal output.

Figur 25: `.class`-fil genererad efter kompilering.

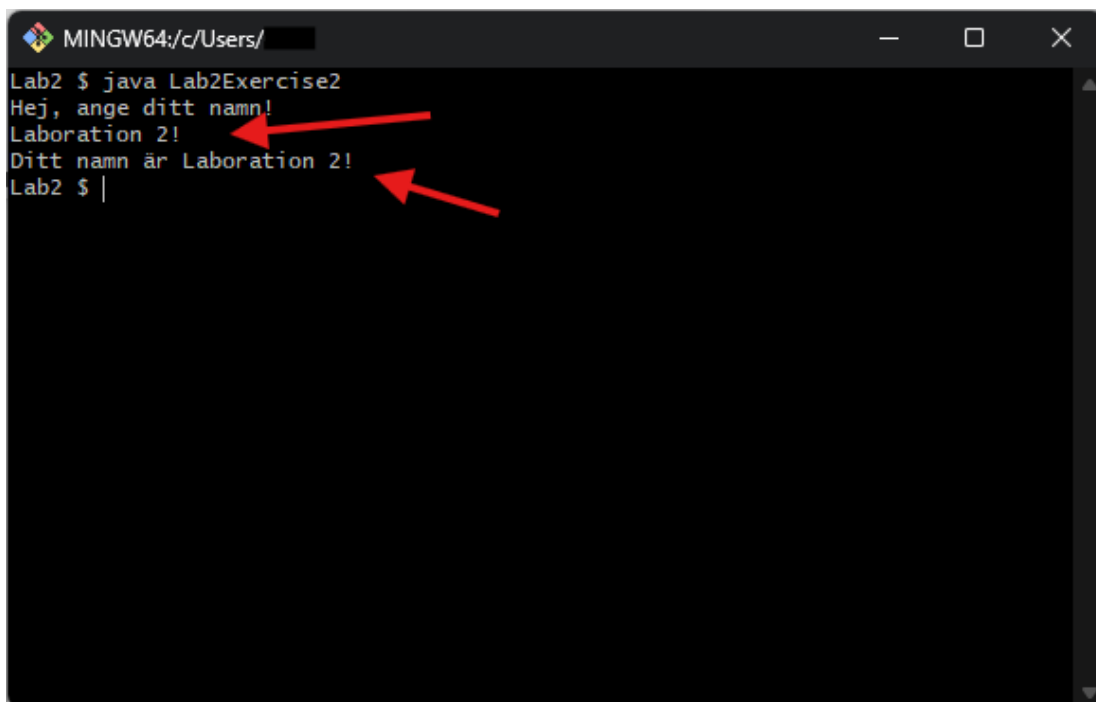
Exekvering

Genom att köra kommandot `java` följt av filnamnet exekveras programmet. Den blinkande markören, som pilen nedan pekar på, indikerar att Git Bash/Terminalen väntar på användarens input.



Figur 26: Exekverad fil väntar på input från användare.

Genom att skriva något på raden med den blinkande markören och trycka på "enter" skickas det namn som användaren har angett. Den övre pilen visar detta, medan den nedre pilen visar hur programmet sedan returnerar det valda namnet till användaren.



Figur 27: Program tar in och använder input för att modifiera en output.

Redigering

Vid kompilering av Lab2Exercise2Fel.java uppstår flera fel enligt felmeddelandet på bilden nedan.

```

MINGW64:/c/Users/
Lab2 $ javac Lab2Exercise2Fel.java
Lab2Exercise2Fel.java:14: error: ';' expected
    System.out.println("Hej, ange ditt namn!")
                                ^
Lab2Exercise2Fel.java:15: error: illegal start of expression
    String name = scanner.nextLine(
                                ^
Lab2Exercise2Fel.java:16: error: ')' or ',' expected
    System.out.println("Ditt namn är " name);
                                ^
Lab2Exercise2Fel.java:16: error: not a statement
    System.out.println("Ditt namn är " name);
                                ^
Lab2Exercise2Fel.java:16: error: ';' expected
    System.out.println("Ditt namn är " name);
                                ^
Lab2Exercise2Fel.java:17: error: reached end of file while parsing
    }
    ^
6 errors
Lab2 $ |
  
```

Figur 28: Hittade fel vid kompilering.

Observera de tre inringande blocken, i fallande ordning är de 1 till 4.

1. Saknat semikolon

- Raden `System.out.println("Hej, ange ditt namn!")` saknar ett semikolon i slutet. I Java måste de flesta kodrader avslutas med `;` för att visa att instruktionen är färdig. Om semikolonet saknas vet inte kompilatorn var instruktionen tar slut, och det blir ett fel

2. Obalanserade parenteser vid metodskapande

- Raden `String name = scanner.nextLine(;` saknar en högerparentes `)`. När man anropar en metod (t.ex. `nextLine()`) måste alltid varje öppnande vänsterparentes (ha en motsvarande stängande högerparentes `)`. Om de inte är i balans kan inte kompilatorn tolka anropet korrekt

3. Fel vid strängkonkatenering

- Raden `System.out.println("Ditt namn är " name)` försöker skriva ut både text och innehållet i variabeln `name`, men de är inte ihopkopplade på rätt sätt. I Java används `+` för att slå ihop text med variabler eller annan text. Det ska vara: `System.out.println("Ditt namn är " + name);`

4. Obalanserade klammerparenteser

- a. En klass eller metod i Java omges av klammerparenteser { och } som visar var den börjar och slutar. I filen finns fler öppnande vänsterklammerparenteser { än stängande högerklammerparenteser }, vilket gör att klassen aldrig avslutas. Detta måste rättas genom att lägga till en avslutande } längst ner i filen.

Nedan finns en fil vilket innehåller korrekt syntax för att kunna köra filen.

```
public class Lab2Exercise2Fel {  
    public static void main(String args[]){  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Hej, ange ditt namn!");  
        String name = scanner.nextLine();  
        System.out.println("Ditt namn är " + name);  
    }  
}
```

Figur 29: Rättade fel i koden.

Skapa repo för inlämningar

Se genomgången av GitHub tidigare i laborationen.