UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

IN1010 - Objektorientert Programmering

Forkurs - Dag 3

Vår 2022

Ahmed Waseem Saeed, Jesper Dahl Norgård

11. januar 2022

Innhold

1 Forord

Velkommen til dag 3 i dette forkurset. Dagens tema er valg av editorer, samt bruk av Git og GitHub. Dagen kommer til å være todelt, der første del dreier seg om editorer (spesielt VS Code), mens andre del handler om versjonshåndtering (Git) og GitHub.

Hvorfor velger vi å bruke tid og ressurser på å lære oss disse tingene? Selv om dere mest sannsynlig er kjent med en editor i form av Atom (i IN1000), og er komfortable med den, så ønsker vi å tilrettelegge for, og vise hvordan en editor kan benyttes for å forenkle arbeidsoppgavene og arbeidsflyten deres. Vi skal i løpet av dagen vise hvordan VS Code tilbyr oss tjenester og verktøy som vi kan ha stor nytte av, og hvordan dette skiller seg fra Atom. Hvilken editor man skal bruke (eller hva som er den 'beste' editoren) er et spørsmål som oftest ikke har et klart svar, Det blir litt som å spørre: "Hvilket programmeringsspråk er det beste?". Svaret er at det spørs, og ethvert programmeringsspråk, samt editorer, kun er verktøy for den enkelte programmerer. Det er den som sitter bak som selv må anvende disse verktøyene til sin nytte.

Versjonshåndtering tillater oss å jobbe i sanntid på samme filer. De fleste har nok opplevd et behov en eller annen gang for å dele filer med hverandre. Uansett, så vil man nok møte på et behov der man trenger at flere jobber på samme filer, på én gang. En måte å håndtere dette på er at begge jobber på filene på hver sin pc, deretter sender den ene filene til den andre, og så skal vedkommende legge inn endringene manuelt på sin maskin. Dette er nok en kjent strategi for mange, men en kronglete en. For å forenkle denne prosessen, og samarbeide effektivt om de samme filene, så skal vi introdusere versjonshåndtering gjennom Git, samt GitHub for å lage delte ressurser som flere kan samarbeide om.

2 VSCode og smart bruk av IDE

2.1 Hva er VS Code, og hvorfor akkurat VS Code?

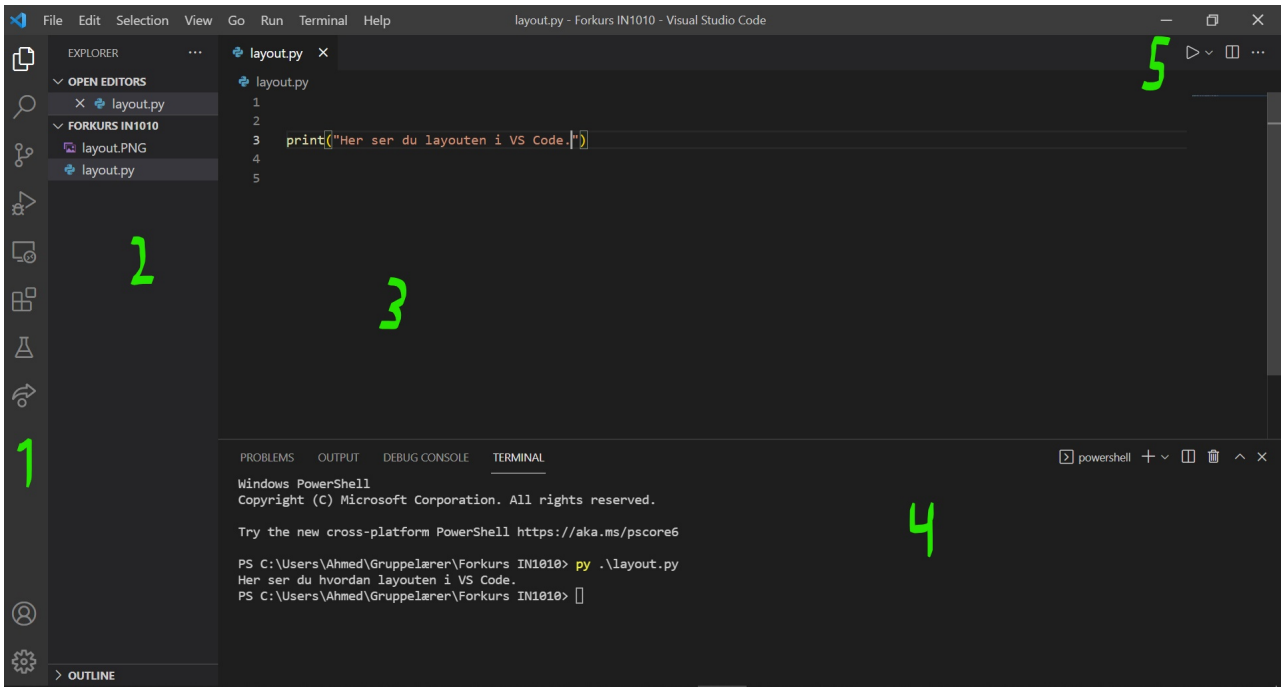
VS Code er en text-editor utviklet av Microsoft, som støtter både Linux, Windows og macOS. Innledningsvis ble det nevnt at det ikke alltid er et lett valg å velge editor. Det finnes en hel haug av editorer; IntelliJ, Eclipse, Vim, Atom, Notepad++, osv. Grunnen til at vi anbefaler VS Code, er at det er en anerkjent og god editor, som støtter mange språk og som kontinuerlig blir oppgradert og bedre (mange bidragsytere på grunn av at VS Code er open-source). Selve editoren er gratis, og kan enkelt lastes ned på følgende side:

https://code.visualstudio.com/download

Merk at det ikke er et krav om å bruke VS Vode i IN1010, Atom fungerer helt fint, men vi viser til VS Code likevel, ettersom editoren tilbyr enkelte tjenester som kan være til nytte for oss (tjenestene følger under).

2.2 Layout i VS Code

Det kan virke som det er en del ting som foregår på én gang i VS Code når man først åpner opp editoren, men vi skal nøye forklare hva hvert av de ulike tegnene betyr. Bildet nedenfor nummerer de ulike komponentene i layouten til VS Code. Her følger en forklaring på hver enkelt komponent:



1. Dette er aktivitetsbaren, som igjen har ulike deler. Den øverste (som er markert med hvit highlight) viser til filene i den aktuelle mappen, her ser du også filer som er aktive i editoren for øyeblikket. Andre tjenester her, inkluderer søkefunksjonen (tilsvarer CTRL + F på nettlesere), versjonskontroll (mer om dette senere), debugging, osv. Merk at aktivitetsbaren kan se ulik ut avhengig av hvilke tilleggsfunksjoner man laster ned. Å laste ned tilleggsfunksjoner kan gjøres gjennom å klikke på følgende symbol:

   Her er det store muligheter for å utforske, i første omgang så anbefaler vi å laste ned Extension Pack for Java, som skal inneholde alt du trenger for å programmere Java-kode i VS Code, i rammen av IN1010.

2. Her vises hvilken komponent av aktivitetsbaren som foreløpig er i bruk (markeres med hvitt). I bildet ser man explorer-funksjonen (som nevnt over viser filer i mappen)

3. Her er 'arbeidsplassen' din, og selve editoren, du kan ha flere filer åpne om gangen, både horisontalt og vertikalt (prøv å 'dra' filen ut og beveg den omkring i dette området for å teste denne funksjonen).

4. Her ser du ulike 'paneler', 'Problems' viser ulike feilmeldinger som editoren har fanget opp, Terminalfunksjonen er en innebygd tjeneste som lar deg bruke ulike terminaler på vanlig måte.

5. Spill av-knappen tillater deg å kjøre koden din. Knappen til høyre for den, lar deg dele editoren din i flere deler og ha flere filer åpne samtidig (likt det som ble forklart under punkt 3).

Merk: dersom du ikke liker fargetemaet i editoren (fargen på koden) så kan dette enkelt endres. Utfør følgende steg for å komme deg til oversikten over fargetemaer:

File → Preferences → Color Theme

Evt. benytte hurtigtasten: CTRL + K + T.

Dersom ingen av fargetemaene faller i smak, så kan du utforske en god del flere temaer ved å klikke på 'Browse Additional Color Themes...'. Dersom savnet fra Atom blir for stort, så kan det være en idé å kikke ut temaet 'Atom One Dark'.

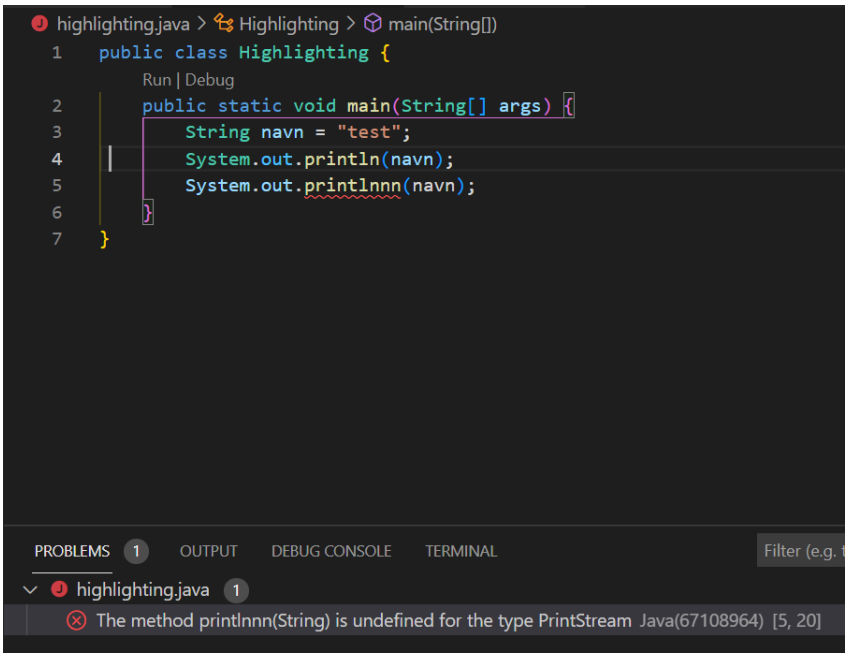
Nå som du er kjent med layouten i VS Code, kan vi vise til de ulike tjenestene som tilbys av editoren.

2.3 Ulike tjenester i VS Code

2.3.1 Syntaks-highlighting og kontinuerlige feilmeldinger

Syntaks-higlighting er mekanismen at ulike deler av koden markeres i forhold til innholdet. Eksempelvis kan funksjonskall, eller nøkkelord som def, for og if, markeres gjennom ulike farger for å vise til hva de gjør. Merk at dette også er en funksjon i Atom.

En av styrkene til VS Code, kontra Atom derimot, er kontinuerlige feilmeldinger som kan fanges før kompilering, og i sanntid. Figuren under illustrerer både dette og syntaks-highlighting (Java-kode):



De ulike fargene viser til at ulike deler av koden har forskjellige formål. Merk forskjellen på linje 4 og 5. Linje 4 er syntaktisk korrekt, mens linje 5 har en skrivefeil. Måten VS Code sier ifra om dette, er gjennom en krøllete understrek på den aktuelle koden. I tillegg dukker det opp et 1-tall i panelet under 'Problems'. Dette er et viktig hjelpemiddel for oss, ettersom editoren sier ifra hvor feilen skjer, samt hva slags feil den oppdager. På denne måten kan vi oppdage feil mens vi programmerer, fremfor å få drøssevis av feilmeldinger under kompilering. Helst bør man unngå å ha varsler i 'Problems', men enkelte varsler (gule varsler), tillater kompilering uten feil. Merk at gule varsler vil også gi gule understreker.

2.3.2 Debugging

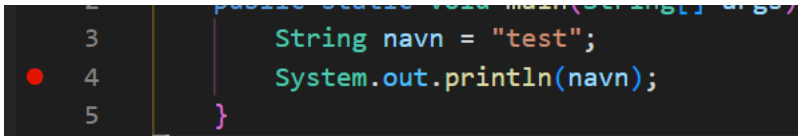
Debugging-funksjonen i VS Code er et meget nyttig verktøy for å analysere koden og se hvordan programmet 'oppfører' seg i ulike stadier av koden.

Debug-funksjonen kan aktiveres ved å klikke på følgende symbol:

Alternativt kan man klikke på 'Debug' over main-metoden i Java-programmer. Dette starter opp debug-funksjonen automatisk.

En viktig del av debuggingen, er å legge inn breakpoints. Dette er områder som angir når programmet skal stoppe underveis i kjøringen, slik at du som programmerer kan analysere koden fra dette ståstedet og evaluere om koden kjører som den skal. Dette er meget nyttig, ettersom det hjelper deg å kartlegge hvor i programmet det potensielt har oppstått feil.

Breakpoints i VS Code legges inn ved å klikke til venstre for linjenummeret du ønsker å markere. Dersom det dukker opp en rød sirkel til venstre for nummeret, så har man markert et breakpoint. Se figuren under.



For å se hvordan programmet ser ut på det nåværende punktet, så kan du sjekke ut debug-panelet, hvor du ser verdiene til ulike variabler under kjøring.

Vi anbefaler å bli kjent med debugging-funksjonen gjennom å prøve seg frem med noen funksjonkall og løkker, for å se hvordan variabler og data endrer seg underveis i program-eksekveringen.

2.3.3 Auto-completion

Auto-completion handler om hvordan editoren fremmer ulike forslag basert på hva du begynner å taste inn. Dette er en funksjon som mange editorer støtter, inkludert både Atom og VS Code. Selv om du kanskje har opplevd Atom sin auto-completion, så vil du fort legge merke til at VS Code har en rekke flere forslag basert på det du skriver. I tillegg kan VS Code fullføre og gi forslag på funksjoner og variabler fra biblioteker som du importerer (gjennom import). Under følger et eksempel på hvordan det å skrive inn 'sys', fremmer et forslag om 'sysout'. Ved å trykke Enter eller Tab nå, så vil editoren skrive 'System.out.println()' for meg.

Helt til høyre følger en forklaring på hvilken metode snarveien fører til og hva metoden gjør. I dette tilfellet så vil 'sysout' → 'System.out.println()', som tilsvarer print()-setningen i Python.

2.4 Hva nå?

Dette er kun en introduksjon til hvordan VS Code fungerer, det finnes mye mer man kan utforske og prøve ut, som ikke dekkes i kurset her. Den beste måten å lære på, er ved å prøve seg frem, utforske editoren og se hva den tilbyr. Nå kan du både lære deg Java, samtidig som du utforsker editoren og lærer deg mekanismer i VS Code.

Dersom du er interessert eller nysjerrig, så anbefaler vi å ta en titt på dokumentasjonen til VS Code, som inneholder en del ytterligere informasjon om de ulike tjenstene editoren tilbyr. Dokumentasjonen kan du finne på følgende side: https://code.visualstudio.com/docs.

En annen stor styrke med VS Code er hvordan den kan integreres med versjonshåndtering og GitHub, men før vi viser dette ønsker vi å introdusere Git og GitHub litt nærmere.

3 Versjonshåndtering og deling med Git og GitHub

3.1 Hva er Git og GitHub?

Git er et gratis system for versjonshåndtering, og er ett av de viktigste verktøyene en programutvikler møter i sin hverdag. Git baserer seg på øyeblikksbilder av hvordan programkoden ser ut, og lagrer referanser til disse øyeblikksbildene. Git lagrer ikke den faktiske koden, men heller en peker til hvordan den så ut på ulike punkter. Da er det lett å for eksempel dele kode, programmere uten å være redd for å ødelegge noe viktig, og holde orden på endringer underveis. Det gjør også Git ekstremt smidig og raskt. Ved å laste ned Git kan du enkelt sette opp et prosjekt på din lokale maskin, og benytte deg av funksjonene for versjonshåntering for personlige prosjekter. For å enkelt kunne bruke Git med flere for samarbeid om utvikling, benytter vi oss av tjenesten GitHub.

GitHub er ikke det samme som Git, men er en online-tjeneste som integrerer Git. Her kan man sette opp prosjekter (repositories) som kan holdes private, deles med noen utvalgte medutviklere, eller være åpne (såkalt Open-Source). Vi skal først se på de grunnleggende funksjonalitetene til Git før vi går over på deling med GitHub.

3.2 Installere Git

Det første som må gjøres er å laste ned Git til din maskin. Ønsker du å gjøre dette ved bruk av installasjonsprogrammer, finner du lenke til dette her: https://git-scm.com/downloads. Vær nøye med å velge riktig OS. Dersom du heller vil, kan du installere via konsollen på Linux og MacOS. Fremangsmåte for ulike operativsystemer finner du på neste side.

Linux

Linux kommer i form av en rekke ulike distribusjoner, alle med litt ulik syntaks for installasjon. Dersom du benytter deg av Linux som et subsystem på Windows (eks Ubuntu) kan du bruke:

```
$ sudo apt install git-all
```

For Fedora (eller lignende distribusjoner) kan du bruke:

```
$ sudo dnf install git-all
```

For alle distribusjoner, se denne lenken: https://git-scm.com/download/linux

MacOS

På Mac kan åpne terminalen og forsøke å sjekke versjonen på git-installasjonen. Dersom det ikke finnes på maskinen, vil du få mulighet til å laste det ned direkte:

```
$ git --version
```

For Mac er det også lett å benytte seg av en installasjonsprogram som finnes lenket til på forrige side.

3.3 Komme i gang med Git

Git tilbyr en stor verktøykasse med funksjonalitet for effektiv versjonshåndtering, og vi skal se på de viktigste. En komplett liste med kommandoer for Git finnes her: https://git-scm.com/docs. Vi skal nå gå gjennom steg for steg den grunnleggende funskjonaliteten til Git for arbeid med prosjekter

3.3.1 Git lokalt

Steg 1: Sette opp et repository

Et repository (forkortet *repo*) er Git's referanse til ditt prosjekt, og det som Git ser på din maskin. Et repo kan enten opprettes lokalt, eller klones fra en ekstern kilde. Det kommer vi tilbake til senere.

Vi begynner med å navigere til stedet der vi vil at repoet skal ligge, og opprette en mappe der. Alle følgende kommandoer vil være terminalkode. Her antar vi at du kan det grunnleggende.

```
$ mkdir prosjekt
$ cd prosjekt
```

Når vi er inne i mappen vi vil ha som rot i repoet kan benytte `$ git init` for å opprette et repo. Det vil da se noe slik ut i terminalen:

```
$ git init
Initialized empty Git repository in /Users/user/Skrivebord/Prosjekt/.git
```

Steg 2: Opprette filer

Nå som vi har et repo kan vi legge til filer. For eksempelets skyld oppretter vi en tom .txt-fil med `touch`

```
$ touch nyFil.txt
$ ls
nyFil.txt
```

Med `$ git status` kan vi enkelt sjekke hvilke filer Git vet om.

```
$ git status
On branch master

No commits yet

Utracked files:
    (use "git add <file>..." to include in what will be committed)
        nyFil.txt

nothing added to commit but untracked files present (use "git add" to
   track)
```

Output over forteller at Git vet om den nye filen, men at den ikke har gjort noe med den enda. Vi må nå fortelle Git at vi skal behandle nyFil.txt. Her er det to faser, *staging* og *committing*. Tenk på staging som klargjøring av filen vi vil ta 'bilde' av, og committing som å ta bildet og lagre det. Vi bruker `$ git add nyFil.txt` for å klargjøre filen vår. Vi kan se at alt har gått bra ved å bruke `$ git status` igjen:

```
$ git status
On branch master

No commits yet

Changes to be committed:
    (use "git rm -cached <file>..." to unstage)
        new file:   nyFil.txt
```

Deretter er vi klare til å utføre vår første commit! Når vi sender bilde av filen til Git, kan vi legge ved en melding som beskriver hva som har endret seg. En 'commit message' bør være noe som enkelt kan forståes av deg selv og andre senere!

```
$ git commit -m "En beskrivende melding"
[master (root-commit) fcb1bd4] En beskrivende meldingl
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 nyFil.txt
```

(Før denne går gjennom kan det hende du må spesifisere et brukernavn og epost. Dette er kun for å tilknytte en identitet til hver commit)

3.3.2 Kort om branching

Branching er et viktig og kraftig verktøy i Git's arsenal, men også et som fort blir komplisert og kronglete når man ikke har gjort det før. Kort fortalt dreier det seg om å dele opp versjonene til en trestruktur slik at de kan videreutvikles parallellt med hverandre, og slås sammen igjen når alt er klart.

Når du har et prosjekt med flere branches kan du bevege deg mellom ulike stadier av prosjektet som mellom grener i et tre.

For å opprette en branch bruker vi kommandoen `$ git checkout -b <branch navn>`. Når denne kjøres vil git opprette en ny branch, og flytte deg ('checkout') til den, vekk fra hovedbranchen!

```
$ git checkout -b newBranch
Switched to new branch 'newBranch'
```

Vi kan undersøke hvilke branches som finnes med `$ git branch`. Navnet som har en '*' indikerer hvor du befinner deg!

```
$ git branch
    master
 *  newBranch
```

På denne branchen kan vi nå gjøre commits, som **KUN** gjelder for denne branchen. Koden vi skriver her, vil være helt adskilt fra 'master'-branchen helt til vi setter dem sammen igjen!

Når du er ferdig med å utvikle nye, flotte features på den nye branchen kan du gå tilbake til 'master'-branchen med `$ git checkout master` og deretter bruke **merge** for å speile kode fra den nye grenen, over på hovedgrenen.

```
$ git merge newBranch
    ...
```

Og voila! Du har nå skrevet features på en egen gren, trygg fra å ødelegge hovedgrenen, og flettet dem sammen igjen!

PS: Vil du bli rå på branching? Sjekk ut denne siden: https://learngitbranching.js.org/

3.3.3 Git og GitHub

Dersom du kun ønsker å benytte deg av Git lokalt, er det nå bare å sette igang! Men Git er best når man jobber sammen! Senere i dette emnet (IN1010) vil det være mulighet for å gjennomføre en obligatorisk innlevering som en gruppe. Det beste verktøyet for samarbeid under utvikling er GitHub. Vi skal nå gå gjennom stegene for å opprette og klone et repo.

Steg 1: Lag en bruker

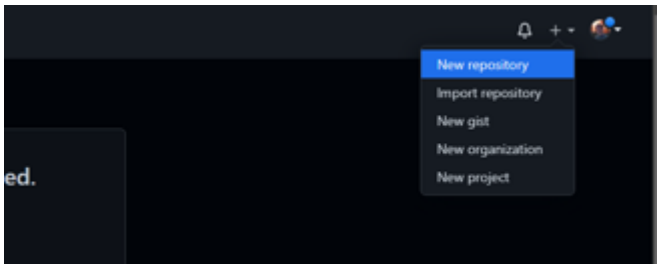
UiO har sin egen GitHub som vi anbefaler å bruke til prosjekter relatert til dine studier.

github.uio.no

Gå inn på lenken over og opprett en bruker med din UiO-epost dersom du ikke har en fra før

Steg 2: Opprett ett nytt repository via GitHub

Her er det flere muligheter, men kan enkelt gjøres ved å trykke på 'New Repository' under '+' ved siden av brukernavnet ditt:

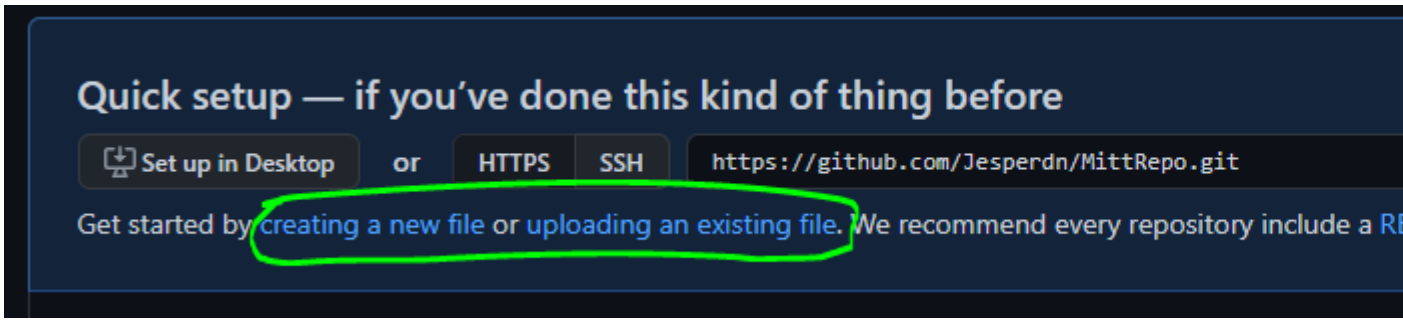


Du vil deretter bli bedt om å opprette et navn på repoet, samt en kort beskrivelse. Du får også mulighet til å velge om repoet skal være privat eller offentlig. For obligatoriske innleveringer **SKAL** repoet være privat med unntak av dine medutviklere på prosjektet.

Du vil også få tilbud om å legge til en README, en .gitignore og en lisens. Disse er valgfrie, og vil ikke bli gjennomgått grundig i dette kurset.

Når alt er klart er det bare å trykke på **'Create Repository'**!

Du har nå opprettet et repository som er klart til bruk. Det finnes imidlertid ingen filer der, disse må legges til manuelt.



Det letteste er å bruke et av valgene over til å enten opprette en ny fil i GitHubs kode-editor, eller å laste opp en fil som du har på din lokale maskin. **NB! Dette er ikke nødvendig å gjøre umiddelbart!**

Du er nå klar til å dele dette repoet med andre (hvis du vil). Mens du er inne på repoet kan du under `settings > Manage access` sette medutviklere til ditt prosjekt. Hvis dine medutviklere har en GitHub-konto kan de legges til her. De vil da få opp ditt repo blant sine egne!

3.3.4 Opp og ned med Git(hub)

Gjennom terminal med repo fra før

Dersom du har et repo fra før du ønsker å jobbe med på din lokale maskin kan du benytte deg av `$ git clone <url>`. Naviger til stedet du ønsker å lagre repoet og bruk kommanden:

```
$ git clone https://github.uio.no/user/repo
    ...
```

Her er det viktig å ha med 'https://'

Du vil nå bli bedt om å identifisere deg. Benytt deg av brukernavn og passord for din GitHub-bruker, eller en *Personal Access Token* (GitHub foretrekker disse for sikkerhet). Dette kan du gjøre ved å gå til:

`Settings>Developer Settings>Personal access tokens`.

Trykk på 'generate token' og huk av for tilgang til 'repo'. Legg til et notat som forteller

hva tokenen gjelder, og generer den. Kopier den eller skriv den ned på et midlertidig notat. **NB! Denne får du kun se én gang!**

Lim den så inn i feltet, du vil nå ha klonet repoet med alt innhold, og kan behandle det etter ønske.

Når du har gjort en endring du ønsker å oppdatere til GitHub sender du den ut med `$ git push`

```
$ git push -u origin <branch>
    ...
```

For å laste ned endringer til ditt lokale repo kan du benytte `$ git pull`

```
$ git pull -u origin <branch>
    ...
```

Gjennom terminal uten repo fra før

Dersom det ikke ligger et repo på github, må vi lage det. Gjør det samme som du gjorde på side 14, men denne gangen tar du heller bare vare på lenken som blir laget. Deretter kan du utføre følgende kommando fra terminalen:

```
$ git remote add origin <url>
git branch -M main
git push -u origin main
    ...
```

Du kan nå gå inn på GitHub og se at repoet du laget er fyllt med hva enn innhold du hadde i ditt lokale repo! Benytt deg av 'push' og 'pull' som vanlig for å laste opp og hente ned oppdatert innhold :D

GitHub Desktop

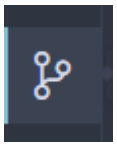
GitHub Desktop tilbyr et svært brukervennlig GUI for bruk av GitHub. Her kan alle repoer behandles på vanlig måte, med grafisk fremstilling av hva som er endret på hver commit. Dette er et godt sted å starte med Git, og kan anbefales til arbeid med IN1010. Gå via lenken under for å laste ned:

https://desktop.github.com/

GitHub Desktop er svært intuitivt og her anbefaler vi at du eksperimenterer med funksjonalitetene for å lære det. Lek deg med å opprette et nytt repo eller klone et fra GitHub og gjør noen commits for å se endringene

3.4 Git og VSCode

Med VSCode kan du du enkelt sette opp og håndtere repositories på GitHub. Se etter dette symbolet på venstre side:



Ved å logge inn og autorisere VSCode overfor GitHub kan du enkelt sette opp et repository når som helst via VSCode. Dette er svært enkelt, så her er det bare å præve seg fram! Følg guiden underveis til hvordan du kan autorisere og logge inn med GitHub!

Dersom mappen du er inne i er et repo fra før, vil du automatisk få tilgang til git-funksjonalitet i VSCode!

4 Avsluttende ord

I løpet av dette forkurset har vi først sett på installering og oppsett av Java, VMware Horizon og Webdrive. Deretter dykket vi ned i terminalbruk, og hvordan man kan benytte terminalen for å fjerninnlogge og overføre filer mellom ulike plattformer. Og i løpet av dag 3 så har vi introdusert ulike tjenester i VS Code, samt versjonshåndtering med Git og GitHub.

Ofte er det ikke nok å kun lese et notat for å mestre de ulike verktøyene, vi anbefaler derfor igjen å prøve seg frem, og virkelig bli kjent med stoffet gjennom å teste og utforske seg frem. Vi håper at disse verktøyene kan gi deg stor nytte i IN1010, og ønsker deg lykke til med semesteret.

Temaene i forkurset er ofte temaer som ikke blir fanget opp i den vanlige undervisningen i ulike emner, men ofte noe man må lære om på egenhånd, Dersom du har en interesse for å lære mer om slike temaer, så kan vi vise til en ressurs utviklet av tre doktorgradsstudenter fra MIT; The Missing Semester. Her kan du se forelesninger og notater om temaer som er akutelle, men som nødvendigvis ikke blir tillagt mye fokus gjennom en grad. Merk at de fleste av disse temaene er mer avanserte enn det vi har gått gjennom. Du kan sjekke ut ressursen på følgende side: https://missing.csail.mit.edu/