Workshop: Versjonskontroll

Dann team a 3 stk.

- Les arbeidsinstruksene nøye – dette er i stor grad en «t-skje introduksjon» - om dere går glipp av en linje/ instruksjon kan det ha betydning for resten av oppgaven.

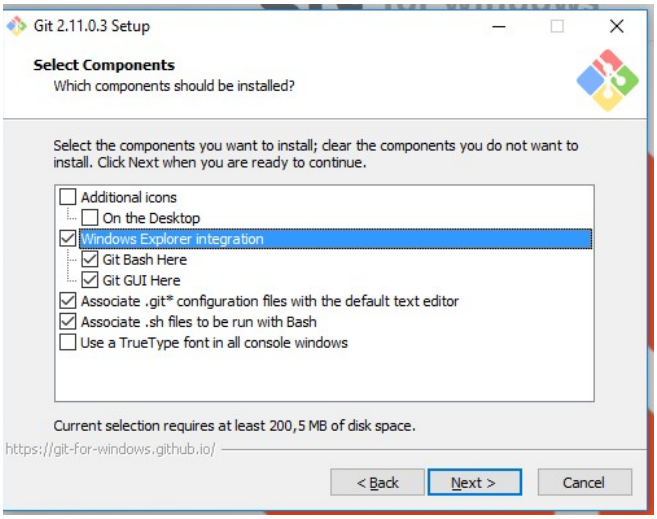
Vi skal i denne workshopen jobbe fra kommandolinja og gjerne også via IDE. Jeg foreslår at gruppa først kjører gjennom via kommandolinja og så går for IDE i runde nummer to (her må dere finne ut hvordan dere skal gjøre det selv). Evt. kan man dele seg slik at noen jobber fra kommandolinja og noen fra IDE. Sitt sammen og hjelp hverandre, slik at alle får erfaring med både konsoll og IDE. Husk også at IDEA viser konsoll-kommandoene som brukes!

1. Installer programvare og log inn på nettsiden Gitlab

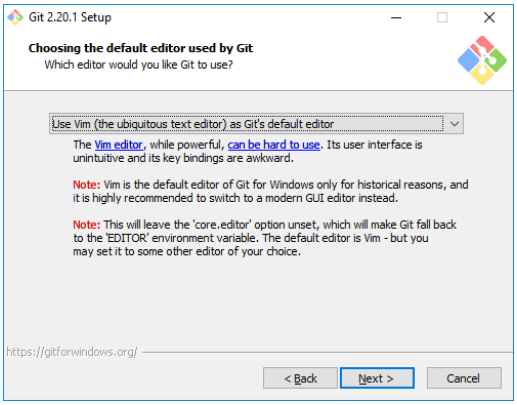
Hvis du ikke har kommandolinjeverktøy for Git installert, last dette ned og innstaller:

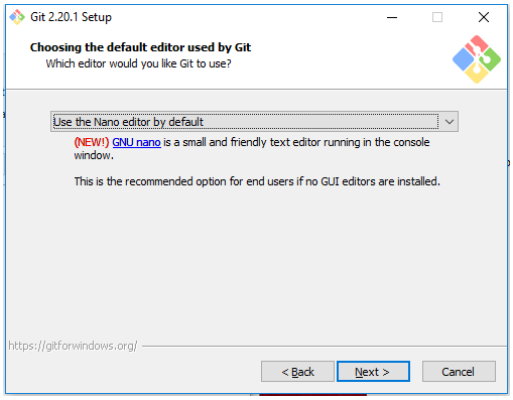
- Windows: https://git-for-windows.github.io/

Kryss av for Windows Explorer intergration om du ønsker å kunne starte Git Bash i stående folder:

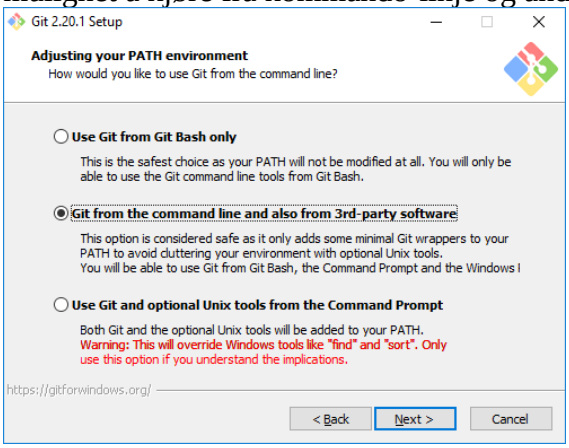


- Velg hvilken editor du vil bruke – Vim er git's default editor – men veldig lite intuitiv. Anbefalt alternativ er Nano, dere kan også velge Notepad++ om dere har denne installert.





- Velg om du vil kjøre git kun fra programmet git bash, eller om du også vil ha mulighet å kjøre fra kommando-linje og andre..



Aksepter de resterende valgene

- Windows: https://git-for-windows.github.io/
- Mac: http://sourceforge.net/projects/git-osx-installer/files/
- Linux: apt-get install git-core

Logg på http://gitlab.stud.iie.ntnu.no/ Bruk feide-innloggings credentials (ditt brukernavn og passord på NTNU)

2. Oppgave 1: Opprett ett felles prosjekt for teamet

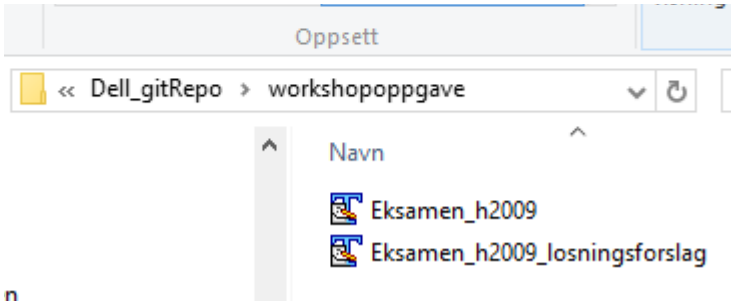
Ett (1) av medlemmene på teamet følger framgangsmåten som oppgitt under og får opprettet et nytt prosjekt som inneholder de to java-filene på sin egen pc og på gitlab. Etter at en på teamet har opprettet sentralt repo som inneholder aktuelle filer skal resten av teamet kopiere (clone) prosjektet og lage et eget lokalt repo på sine maskiner.

I blackboard finner dere et delvis ferdig løsningsforslag til en eksamen i Programmering grunnkurs H2009 og et fullstendig.. Løsningen består av 3 klasser, alle lokalisert på samme fil: Eksamen_h2009.java. Dere skal kopiere denne fila og jobbe litt med disse klassene.

Framgangsmåte – Opprette nytt prosjekt:

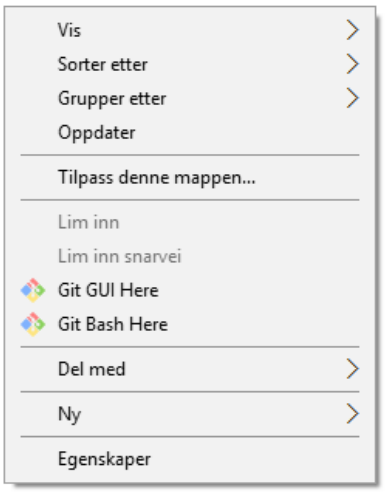
Steg 1: Opprette Lokalt Repo på egen PC

Lag en mappe på egen pc i explorer som skal være ditt lokale workingspace og repo. Legg inn de to filene vi skal jobbe med i mappen. Legg de evt. i en egen mappe.



Høyreklikk mens du står i denne mappen og velg Git Bash for å starte kommandolinjeverktøyet (forutsetter at du valgte Windows Explorer Integration under installasjon).

TIPS til det som har valgt å bruke VIM-editor ved installasjon av git: Generelt når du bruker GitBash: dersom du får opp VIM-editor, avslutter du denne ved å skrive inn :q!

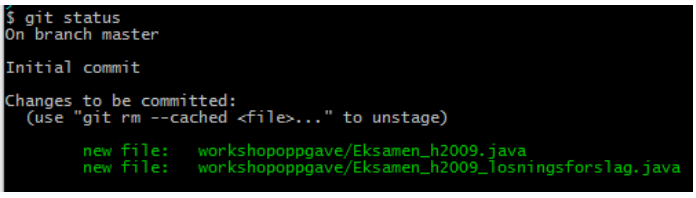


Macbrukere: start terminal og naviger til ønsket mappe.

Før vi starter skal dere legge inn egen identitet (brukernavn og epost-adresse). Denne informasjon brukes hver gang du foretar en commit (sjekker kode inn/ ut el foretar endringer) slik at alle kan se hvem som har lagt inn endringen. Du trenger kun å gjøre dette en gang.

$ git config --global user.name "brukernavn"
$ git config --global user.email epost@adresse.no

Gi inn følgende kommandoer:

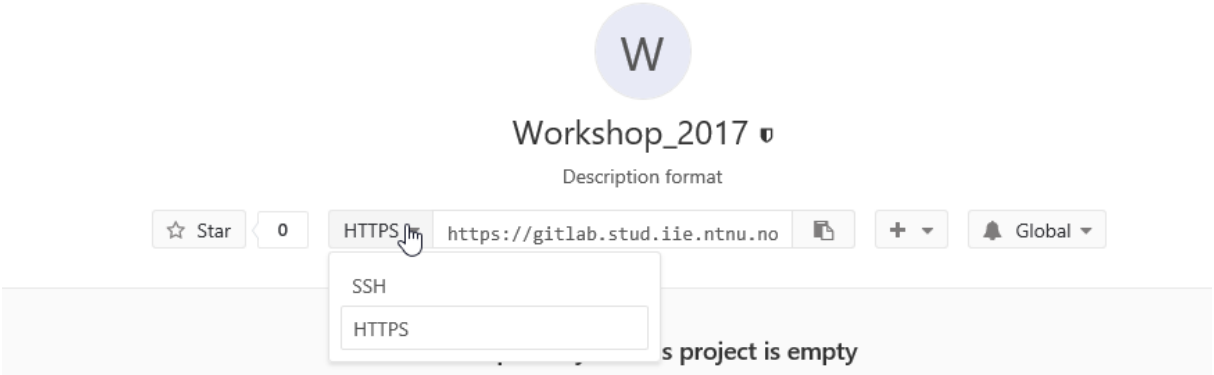
Opprett lokalt repo:	git init
Legg de to filene som ligger i mappen til git repoet (eksamen_h2009.java og Eksamen_h2009_losningsforslag.java): Kommandoen add . legger til alle filer/ endringer i stående katalog.	git add .
Ta en sjekk på hvordan dere ligger an ved å bruke kommandoen:  Her ser vi at 2 filer er lagt til "stage" – de er altså ikke lagret i ditt lokale repo enda, men ligger på vent.	git status
Lagre endringene i ditt locale repoet	git commit –m «Første commit»

Steg 2: Opprette Sentralt Repo på gitlab – felles for hele teamet

Da skal vi opprette et sentralt repo som skal være felles for alle på teamet.

Åpne/ logg deg inn på http://gitlab.stud.iie.ntnu.no og lag et nytt prosjekt – velg type Internal – da er det tilgjengelig for alle registrerte brukere (kun iie-studenter/lærere) Husk å gi prosjektet ett navn før du trykker på Create project.

Du vil da få opp en liste over hvilke kommandoer du skal kjøre på lokal pc for å koble deg opp mot det sentrale repoet for hhv HTTPS-protokoll og SSH-protokoll. Merk forskjellen på ssh og https:



I kommandolinjeverktøyet (gitbash) – gi følgende kommando for å opprette kontakt med det sentrale repoet. Velg enten 1. HTTPS eller 2. SSH. *På nyere GitLab finner du valget/stien til SSH og HTTS under Clone-knappen.*

***OBS:** Du vil ikke kunne bruke push og pull på koden via SSH uten å opprette SSH-key for profilen din. – Se siste kapittel i denne veiledninga for mer detaljer rundt oppretting SSH-nøkler:*

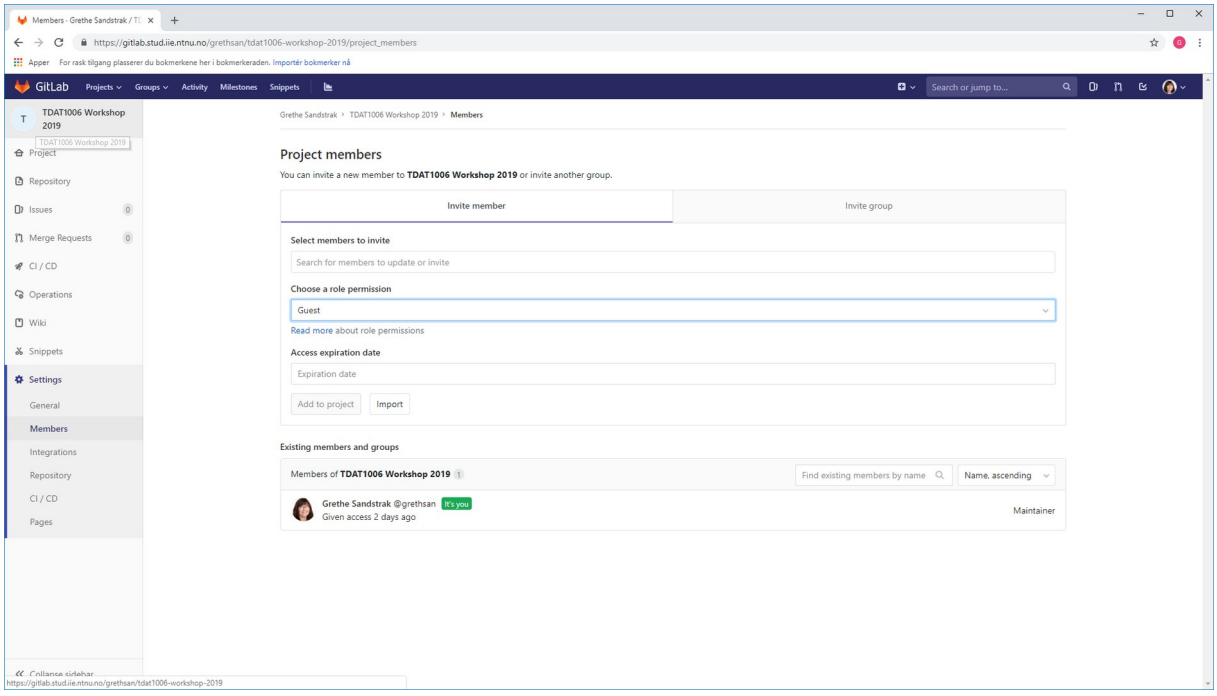
Enten HTTPS, kobler til det sentrale repoet via en web-tjener. (http-protokollen)	git remote add origin https://gitlab.stud.iie.ntnu.no/grethsan/Workshop_2019.git
Eller SSH – går ikke via web-tjeneren, men kobler direkte til det sentrale repoet. (via en ssh-protokoll)	git remote add origin git@gitlab.stud.iie.ntnu.no:grethsan/Workshop_2019.git

Husk å opprette ssh-keys..	
Overfør filene fra det lokale repoet til det sentrale (det kan være du må oppgi brukernavn og passord for gitlab) Legg merke til –u i kommandoen. Dette gjør at Git husker parameterne slik at du neste gang kun trenger skrive git push.	git push –u origin master

Gå tilbake til http://gitlab.stud.iie.ntnu.no og se på prosjektet dere nå har opprettet og dets filer der.

Steg 3: Legg til de andre medlemmene til prosjektet og gi rettigheter

Personen som opprettet prosjektet går inn på gitlab og prosjektet. Der velges Settings-Members. Legg til alle – enten en og en, alternativ opprett en egen gruppe for teamet-deres og inviter hele gruppa. For at medlemmene skal kunne gjøre endringer må de har rettigheter som Developer eller Maintainer.



Steg 4: Clone prosjektet: Få inn resten av gruppa på samme prosjekt

Alle medlemmene i gruppen kan nå klone det nye prosjektet slik at alle får en kopi av repoet på deres egen maskin.

Start kommandolinjeverktøyet Git Bash og skriv inn kommandoen under – legg inn ditt eget brukernavn og legg inn riktig prosjektnavn (macbrukere: start terminal og gå til ønsket mappe) enten vha HTTPS eller SSH (forutsetter at ssh-nøkler er lagt inn)

HTTPS
$ git clone http://gitlab.stud.iie.ntnu.no/brukernavn/prosjektnavn.git
Brukernavn er navnet på den brukeren som opprettet prosjektet

SSH
$ git clone git@gitlab.stud.iie.ntnu.no:brukernavn/prosjektavn.git

Ikke gå videre før alle er klar – kollektivt ansvar for at alle får ting til å virke.

2. Gjøre endringer i koden

Før dere starter å gjøre endringer må vi si fra til vårt lokale repo om hvordan vi ønsker å oppdatere det sentrale repoet. Ønsker vi å overføre/ merge samtlige brancher (grener) eller kun den aktive? Fram til nå jobber vi bare mot en branch (master), men vi skal senere opprette flere brancher og da kan det være lurt å velge siste metode:

1. Pushe alle brancher (ikke bruk denne) git config --global push.default matching

2. Push kun aktiv branch (velg denne) git config –-global push.default simple

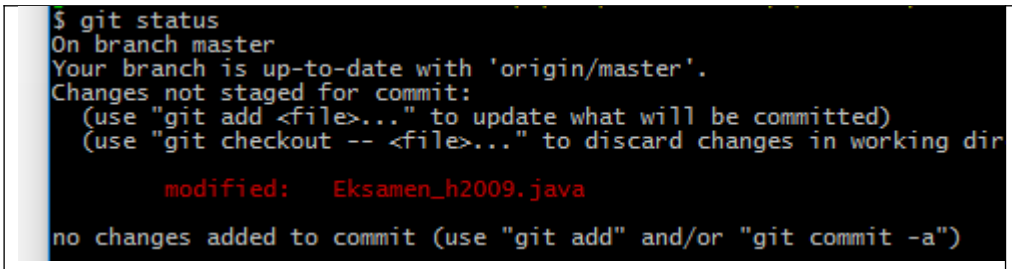
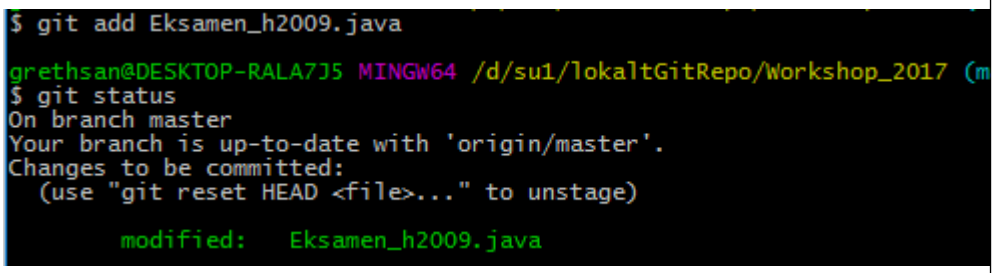
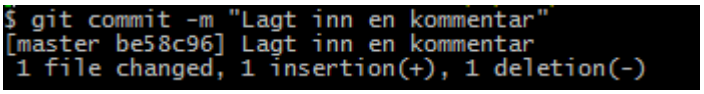
Nå skal dere "jobbe" litt på prosjektet. Start med å legge inn hver deres linje med kommentar i koden – på forskjellige steder slik at det ikke oppstår konflikter.- bli enige om hvor dere legger disse inn på forhånd. ‘

Lagre endringene du har gjort før du går videre.

Da kan endringene overføres til det lokale repoet og deretter til det sentrale repoet. Vi starter med å inspisere de endringene som er gjort lokal ved å kjøre en diff før vi går videre:

Vise alle endringer vi har gjort på koden
 	
(figuren under viser at vi har lagt til en ny kommentar i java-fila.
 	
$ git diff
diff --git a/Eksamen_h2009.java b/Eksamen_h2009.java
index ba9c167..23e287a 100644
--- a/Eksamen_h2009.java
+++ b/Eksamen_h2009.java
@@ -17,7 +17,7 @@ import static javax.swing.JOptionPane.*;
class Spor{
String navn;
String artist;
- double lengde;
+ double lengde; // Ny kommentar oppgis i minutter
 	
public Spor(String navn, String artist, double lengde){
this.navn = navn;
	$ git diff

Vi sjekker tilstanden på repoet vårt:	$ git status

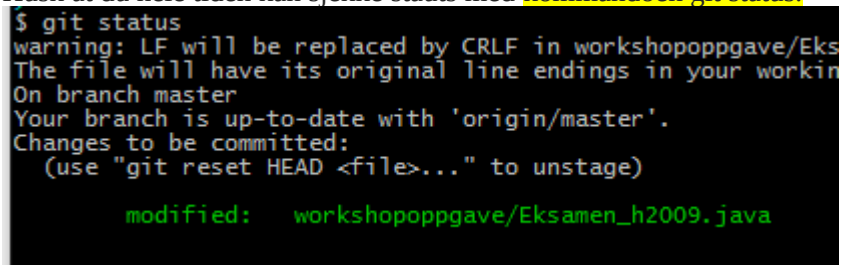
	
Legge filene klar for commit til ditt lokale repo	$ git add filnavn
Vi har alltid mulighet for å sjekke tilstanden på repoet vårt: 	$ git status
Vi commiter endringene til vårt lokale repo 	$ git commit –m «fornuftig melding som forklarer hva som er endret»

Da er oppdateringa lastet opp til det lokale repoet. Neste trinn er å laste det opp til det sentrale repoet på gitlab slik at de andre kan få tilgang til endringen.

Før vi laster opp våre endringer til repoet er det være lurt å sjekke om det er gjort noen endringer sentralt som vi ikke har lastet ned før vi laster opp egen endring:

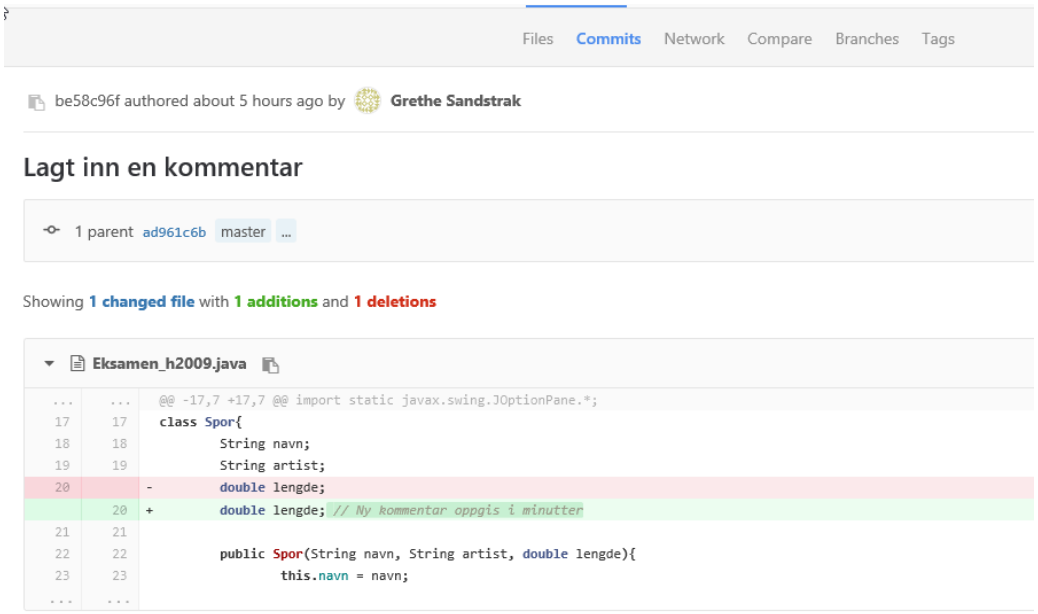
$ git pull - last ned eventuelle endringer og merge denne sammen med koden har i eget lokalt repo. Du får tilbakemelding på om endringer blir foretatt eller ikke:

Husk at du hele tiden kan sjekke stauts med kommandoen git status.



$ git push –u origin master – laster opp endringen til sentralt repo

Vi tar en titt på prosjektsiden på gitlab.stud.iie.ntnu.no:



3. Branching

Før dere går videre, forsikre dere om at alle har siste versjon av prosjektet ved å kjøre kommondoen:

```
$ git pull
```

Fordel nå de tre oppgavene (a-c) under mellom dere og lag egne brancher for Spor og Album-klassen. Lag branchene først (se lenger ned hvordan dette gjøres) og gjør deretter endringer i koden.

a) Klassen Spor – endringer utføres i branch Spor

- Legg inn get og set-metoder for alle objektvariabler
- toString-metode – fyll inn det som mangler
- Equals-metode - fyll inn det som mangler

b) Klassen Album – endringer utføres i branch Album

- Konstruktør mangler innhold
- Metoen getSpilleTid() mangler innhold i for-løkka
- Metoden re
- gNyttSpor() mangler innehold i else-delen
- toString() metoden mangler innhold

c) Klientprogrammet Eksamen_h2009 – endringer utføres i branch master

- Endre fra switch-uttrykk til if-else.
- Legg til et ekstra valg i menyen: Overraskelse. Du velger selv hva som er overraskelsen
-

Lag en ny branch for hver av klassene Spor og Album :

git branch <navn på branch>

For å jobbe med en branch må vi først «logge på» denne vha kommandoen checkout:
git checkout <navn på branch>

Gjør ønsket endring i filene og lagre endringen i lokalt repo:
git add filnavn
git commit –m “Fornuftig beskrivelse av hva som er gjort”

Push endringene og den nye branch'en til sentralt repo på GitLab (husk å sjekke om det er gjort endringer i det sentrale repoet før dere pusher):
git push -u origin <navn på branch>

Når dere har pushet endringene se på GitLab-nettsiden og sjekk "Graph"- for å se en grafisk representasjon av repoet.

Legg inn en merge-konflikt

Før dere går vider til neste kapittel og merger alle branchene til en –samarbeid med de andre på teamet og prøv å endre på samme kodelinje i to ulike brancher. Dette for å skape en konflikt når vi *senere* skal flette sammen branchene. Etter dere har endret koden kjører dere igjen:

git add .
git commit –m “Fornuftig beskrivelse av hva som er gjort”
git push

(merk – nå kan dere kjøre kortvarianten av git push – da dere satte sammenheng mellom lokal og sentral første gang)

4. Merging av branch’er

Tilslutt skal vi merge branch'en med master. Sørg for at du står i Master-branch (git checkout master) før du merger.

git merge <navn på branch>

OBS dersom du får opp VIM-editor, avslutter du denne ved å skrive inn **:q!**
For de som bruker MAC eller Linux kan dere bruke nano som standard editor i stedet for VIM. Dette kan gjøres ved kjøre kommandoen: $ git config –global core.editor nano. For de so mbruker windows kan dere også bruke nano eller for eksempel notepad++.

Får å ferdigstille mergen og samkjøre med sentralt repo på gitlab:
git push -u origin master

Observer at branch'en har kommet tilbake til master på "Network"-fanen i GitLab.

Dersom det oppstår en konflikt – dere har redigert på samme linje i to ulike brancher (spor/album/master) så oppdages det ved merging. De kodelinjene vil bli uthevet i den aktuelle fila i ditt lokale repo (på din maskin). Åpne aktuell fil og rediger denne manuelt – fiks den slik du vil den skal være å kjør så ny commit før du prøver å merge igjen.

OBS: Den som merger sist vil få beskjed om å kjøre en pull før merge kan gjennomføres fordi det er endringer på master-branch som ikke er lastet ned. Typisk melding som presenteres: «Your branch is

ahead of ‘origin/master’ by 1 commits».

Eks på ei fil der konflikt er oppdaget under merging;

```
<<<<<<< HEAD
/* Her mangler get/set - metoder, legger inn kommentar for å skape konflikt med
branchen Spor før merging*/
=======
/* Her mangelr get/set - metoder  - det skal være mulig å endre alle
objektvariabler*/
>>>>>>> Spor
```

Her må dere inn i koden og rette opp: <<<<HEAD angir starten på der konflikten er, >>> angir slutten. ===== angir skille mellom versjon 1 og versjon 2 av disse 2 kodelinjene. Fjern det som ikke skal være med – inkludert <<<head, == og >>> slik at kun for eksempel dette gjenstår:

```
/* Her mangler get/set - metoder, legger inn kommentar for å skape konflikt med
branchen Spor før merging*/
```

Lagre endringer, kjør add, commit, pull og push igjen.

Når en branch er ferdig merget med master-branch er det vanlig å slette den. Vanligvis ikke før koden er testet (junit) i henhold til krav til testing/ kvalitetssikring og klarert for merging med master.

Før branchen slettes sjekk om den er merged
$ git branch --merged

```
grethsan@DESKTOP-RALA7J5 MINGW64 /d/su1/lokaltGitRepo/Workshop_2017 (master)
$ git branch --merged
  album
* master
  spor
```

Slett branch’en lokalt – den vil fortsatt eksistere sentralt.
$ git branch --d navn_på_branch

For å liste alle branch’er bruker vi git branch –a. Under har vi listet alle branch’er, så slettes branch spor og til sist listet alle branch’er igjen. Vi ser her at branch spor fremdeles er med på sentralt repo (remote/origin/spor).

```
grethsan@DESKTOP-RALA7J5 MINGW64 /d/su1/lokaltGitRepo/Workshop_2017 (master)
$ git branch -a
  album
* master
  spor
  remotes/origin/album
  remotes/origin/master
  remotes/origin/spor

grethsan@DESKTOP-RALA7J5 MINGW64 /d/su1/lokaltGitRepo/Workshop_2017 (master)
$ git branch -d spor
Deleted branch spor (was cfe73bc).

grethsan@DESKTOP-RALA7J5 MINGW64 /d/su1/lokaltGitRepo/Workshop_2017 (master)
$ git branch -a
  album
* master
  remotes/origin/album
  remotes/origin/master
  remotes/origin/spor
```

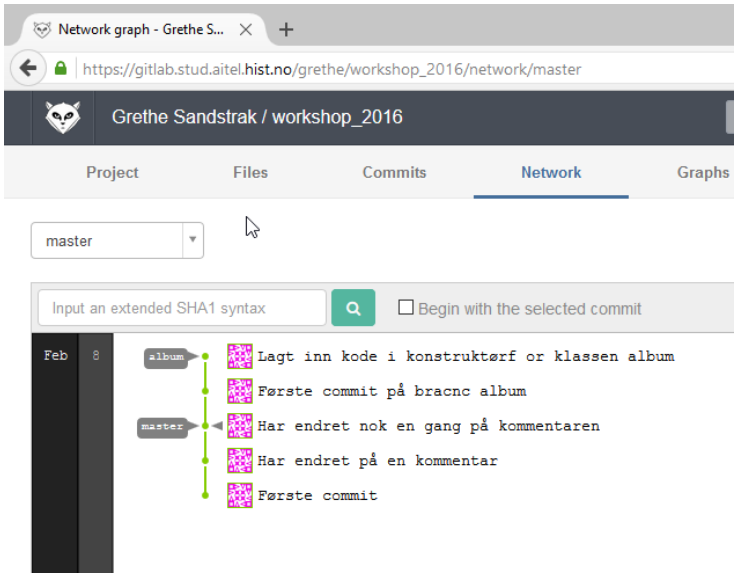
For å slette en branch sentralt bruker vi kommandoen:
$ git push origin –delete navn_på_branch

Vi ser at når vi så lister opp alle branch'er lokalt og sentralt så er nå branch spor borte.

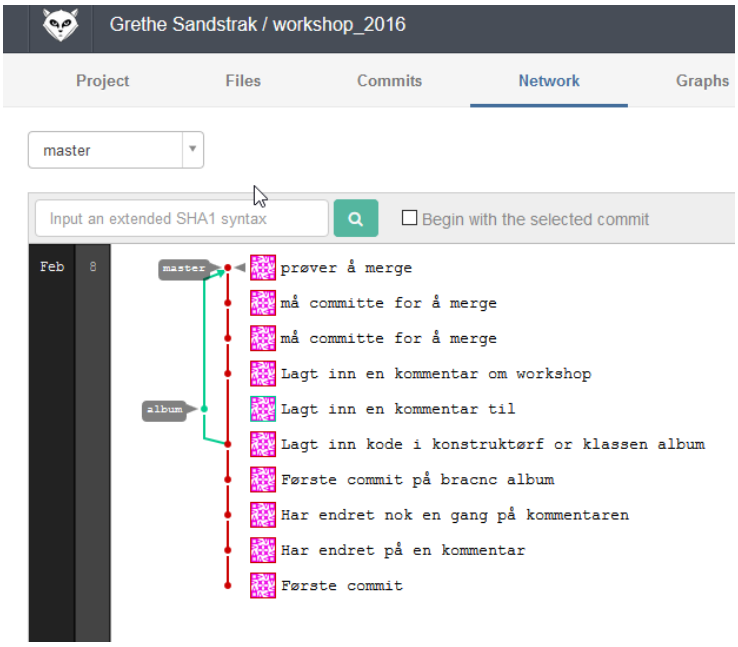
```
grethsan@DESKTOP-RALA7J5 MINGW64 /d/su1/lokaltGitRepo/Workshop_2017 (master)
$ git push origin --delete spor
To gitlab.stud.iie.ntnu.no:grethsan/Workshop_2017.git
 - [deleted]         spor

grethsan@DESKTOP-RALA7J5 MINGW64 /d/su1/lokaltGitRepo/Workshop_2017 (master)
$ git branch -a
  album
* master
  remotes/origin/album
  remotes/origin/master
```

Visualisering av prosjektet i gitlab.stud.iie.ntnu.no.

Eksemplet viser at det er opprettet en ny branch album og gjort en endring som er pushet til sentral-repoet:



Jobber videre i to brancher – foretar 3 endringer på master branch før de to branchene merges sammen:



6. Opprette SSH-nøkler

– dersom dere ikke skal bruke HTTPS-protokollen

Før du kan push'e endringer til en GitLab-server trenger du en sikker kommunikasjonskanal for å dele informasjon. GitLab bruker Public-key eller asymetrisk kryptering som krypterer kommunikasjonskanalen ved å låse den med din private nøkkel og tillater «trusted parties» å låse opp med din offentlige nøkkel.

SSH: Nøkkelautentisering virker slik at man har en hemmelig og en offentlig nøkkel i par. Den offentlige delen av nøkkelparet kan man fritt distribuere da denne i seg selv ikke er hemmelig. Den hemmelige delen av nøkkelparet er naturligvis hemmelig. SSH-tjenesten man kobler seg opp mot må ha tilgang til den offentlige nøkkelen og SSH-klienten må ha tilgang til den hemmelige nøkkelen.

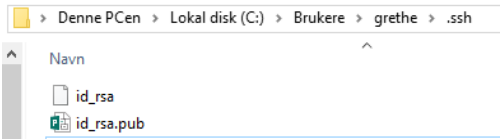
SSH-nøkler kan opprettes som anvist på nettsiden for gitlab (du finner denne under Profiles Settings og fanen SSH) hvis dere vil slippe å bruke passord for leseaksess til repoene. Dere kan droppe dette og i stedet kommunisere via HTPS-protokollen.

Generer privat og offentlig nøkkel for GitLab på din maskin. –

Gjøres av alle på teamet.

Lagre private og public key i .ssh-mappa (Windows: C:/brukere/brukernavn/.ssh, MAC: Macintosh HD/brukere/brukernavn/.ssh). Det blir generert to filer for hhv private og public nøkkel. Filene skal hete id_rsa og id_rsa.pub.

f

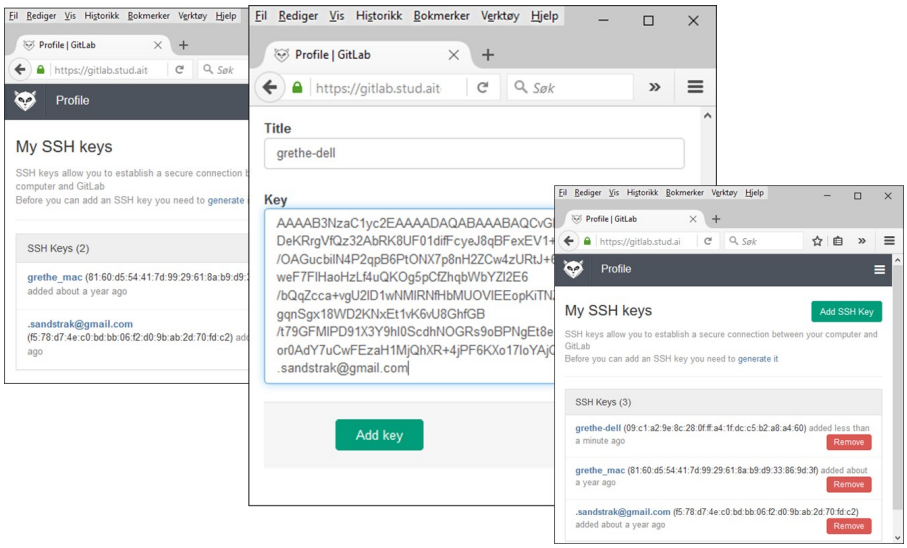


OBS: Her kan det oppstå en del trøbbel! – Alternativ til PuttyGen.exe som foreslås av gitlab for å generere nøklene kan være applikasjonen Git bash som vist i presentasjon eller Git GUI.

1. Kommando for å generert nøkler – git bash/ terminal:

ssh-keygen -t rsa -C "tittel" -f $HOME/.ssh/id_rsa
ssh-keygen -t rsa -C "tittel" -f ~/.ssh/id_rsa

I gitlab, velg Profile Settings og fanen SSH-Keys. Lim inn din offentlige nøkkel som du genererte tidligere her. –Windows: paste inn kopierte fra git bash/ terminal. Kjente feilmeldinger som kan komme: ugyldig fingerprints, linjeskrift mm



Nyttig feilsøkingskommando for debugging i tilfelle problemer ved kjøring av push mot det sentrale repoet: ssh -vT git@gitlab.stud.iie.ntnu.no

TIPS, Git-ressurser

- Git-tutorial: https://try.github.io/levels/1/challenges/1
- https://docs.gitlab.com/ee/README.html
- https://github.github.com/training-kit/

Nyttige kommandoer

Når du står i kommandolinjeverktøyet kan du få hjelp med de ulike kommandoene ved å skrive:
git help <kommando> eller git <kommando> --help

$ git help add
$ git add --help

Dersom du ønsker å slutte å følge et sentralt repo kan du skrive inn kommandoen:
$ rm –rf .git

Dersom du ønsker å angre en fil du har lagt til staging-area (kommandoen add):
$ git reset filnavn

Lister all informasjon til sentralt repo

$ git remote –v

Lister all informasjon om brancher lokalt og sentralt
$ git branch -a