Grunnleggende/enkel bruk av Git (Bash) og GitLab:

1. Logg inn vha. din vanlige Feide-bruker på ***IDIs GitLab*** (**https://gitlab.stud.idi.ntnu.no**)
   Sett evt. opp to-faktor-autentisering**.**

2. *En* av gruppens medlemmer oppretter et repo(sitory): '+'-tegnet (øvre venstre hjørne)
   → «New project/repository» → «Create blank project» → prosjekt-/reponavn
   (***navnet starter med «GruppeXX-…», uten* blanke eller bindestrek,** *der 'XX' (01, 02, ….., 99) er gruppenr dere valgte i BlackBoard. Husk innledende '0' når mindre enn 10*). La det være «Private», sett evt. hake foran «Initialize repository with a README» → «Create project»

3. Inviter inn de andre gruppemedlemmene: Manage (langs venstre marg) → Members →
   Invite members → brukernavn, «Select a role» ***må** være «Developer»* → «Invite».

4. Oppsetting av SSH-nøkkel: Grønn prikket ikon øverst i venstre hjørne → Preferences →
   SSH Keys (langs venstre marg). Finn aktuell fil på din maskin (søk etter filen «id_*.pub».
   På emnelærers maskin ligger den på katalogen: c:\users\frh\.ssh), og klipp inn *hele* dens
   (lange og merkelige) innhold i feltet → «Add new key» (Finnes den ikke, lages den ved å skrive
   i Command-window: ssh-keygen -t rsa -b 2048 (svar gjerne bare ENTER på spørsmålene))

5. Installer Git fra https://git-scm.com/downloads (eller https://gitforwindows. org)
   og start Git Bash. (avslutte Git: **exit** Ut av vi-/vim-editor: ESC, ':' og 'q/exit/quit')

6. Sett opp initielle (person)data:
   git config --global user.name "<name>"
   git config --global user.email "<email address>"
   git config --global color.ui auto
   (git config --global core.editor <for Git-editering>)
   Eks: git config --global core.editor /c/windows/notepad.exe
   (git config --list)

7. Gå til katalogen der du vil ha prosjektet liggende *under* (*Working Directory*) på din maskin:
   cd <path> Eks: cd /c/ntnu/prosjekter ('c' er driven)
   (**ls [-la]** - viser filene i en katalog)

8. Git-repo opprettes lokalt (*en* gang) *under* dette ved å clone/kopiere repoet dere laget i pkt.2
   ovenfor (eieren av repoet må også gjøre dette). Evt. må *egen* bruker/passord oppgis.
   git clone <domene>:<bruker>/<reponavn>.git
   Eks: git clone git@gitlab.stud.idi.ntnu.no:kari/GruppeXX.git
   (Er å finne oppe til høyre under den blå «Codegit » på repoets webside.)
   (*Skulle cloningen feile svar teksten «yes», ikke bare 'y', evt. prøve igjen!*)
   Det lages en .git-katalog (med subkataloger og filer) under denne katalogen.
   Slike dot-kataloger blir ofte skjult/usynlige i filvisningen (i Windows).
   Ved cloning blir eksternt repo automatisk hetende 'origin'.
   (Alternativt: Lag underkatalogen der det nye prosjektet skal ligge. Gå til denne (cd ….).
   Lag tomt Git-repo der: **git init**
   Underkatalog på egen PC med samme navn som prosjektet/repoet?
   Svar: Skjer automatisk når clones. Anbefalt når opprettes manuelt (init).)

9. Registrer fil(ene) i *Staging Area*: **git add <filnavn>** Eks: git add hus.cpp
   git add *.h git add *.cpp git add *.dta
   Hver gang en ny fil er laget i prosjektet, eller er editert og skal bli en del av en ny commit, kjøres add-kommandoen. Mulig å adde filer fra andre kataloger ved også å angi deres path/sti.

   NB: Add *kun* .h-, .cpp- og .dta-filer - *ikke* binære filer!
   Add medfører altså at de legges i «Staging Area» - arbeidsområde for å bli committed (snart).
   Det motsatte – fjerne fra Staging Area: **git reset <filnavn>**

10. Se status for endrede filer som *kan* addes til (rød-merkede) eller er endret men added allerede i (grønn-merkede) Staging Area: **git status**

11. Registrere en «utgivelse/versjon» *lokalt* som det kan/skal utføres versjonskontroll på (Staging Area blir committed): **git commit [-m «en-linjes forklarende tekst»]**
    Utelater to siste parametre: Editor startes automatisk og lengre beskrivelse kan angis.
    Variant: Adder *alle* endrede *og* commiter: **git commit -am «en-linjes forklarende tekst»**

12. Se forskjellen mellom:
    - nylig endrede filer og Staging Area: **git diff**
    - Staging Area og siste commit: **git diff --cached**
    - endrede filer og siste commit: **git diff HEAD**

13. Få status for/info om ulike commits: **git log [-p | -<n> | --oneline | --stat | --patch | --word-diff | --pretty=….. | --since=….. | ……..]**

14. Eksterne repoer:
    Se hvilke: **git remote [-v]**
    Se mer info: **git remote show <reponvn>**
    Koble til: **git remote add <reponvn> <domene>/<bruker>/<reponavn>**
    Eks: git remote add origin https://per@git.gvk.idi.ntnu.no/per/GruppeXXX.git
    Hente ned: **git pull [<reponvn>]** Fetcher og merger inni i nåværende branch
    (**git fetch [<reponvn>]** NB: Merger *ikke*!)
    Laste opp: **git push [-u] [<reponvn> <branchnavn>]**
    Eks: git push origin master/main

Konflikter: En del må løses manuelt (editeres). Addes og commites på nytt, og så pushes.

Arbeidsflyt i prosjektet:

1. **Skrive kode/utvikle**
2. **git status**
3. **git add <filnavn> | *.h | *.cpp | *.dta | .** (= alle filer)
4. **git commit …….**
5. **git pull**
6. **(git merge <gren>)**
7. **git push […..]** **NB:** Push *bare* kode som kompilerer og *fungerer* korrekt!

- Branch/grening («arbeidskopi»):
  Se alle nåværende brancher: **git branch**
  Hva committed sist i hver branch: **git branch -v**
  Editere/utvikle/teste i ny gren: **git branch <nytt-branchnavn>**
  Bytte til annen branch: **git checkout <bytte-til-branchnavn>**
  NB: Kopierer det siste committed for aktuell branch (i repoet) til Working Directory
  Variant: Lage ny *og* bytte: **git checkout -b <nytt-branchnavn>**
  Flette gren med nåværende: **git merge <flette-inn-branchnavn-i-nåværende>**
  NB: Stå i *en* branch og merge inn en *annen* branch (med endringene).
  Hva merged inn i nåværende: **git branch [--merged | no-merged]**
  Slette en gren: **git branch -d <branchnavn>**
  Ved merging: Konflikt når samme tekst/kode endret i begge – må løses manuelt.
  Hovedgrenen i Git heter alltid 'master/main'. Nåværende gren omtales også som 'HEAD'.
  Ny branch i eksternt repo: Vite om/oppdage evt. **fetch/pull**. **checkout** til den nye.
  Utvikle i 'master/main' – testet/ferdig kode i *en* egen branch (hetende 'release' el.l)?

- Slette fil(er) *totalt*: **git rm <filnavn>**
  Fjerne kun fra repo (versj.kontr): **git rm --cached <filnavn>**

- Rename/flytte en fil til annen katalog: **git mv <orginalt navn> <nytt navn (m/path)>**
  Rydder opp i flere renamede/flyttede filer (utført utenfor git): **git add -A .**

- Angrer/undo noe (men prøv å unngå, da kan få store/merkelige konsekvenser):
  Overskriver *aller siste* commit: **git commit --amend**
  Siste endring på en committed fil: **git checkout -- <filnavn>**

- Forhindre at (statiske/binære) filer/kataloger stadig blir med i versjonskontrollen:
  Opprette og bruke **.gitignore**-fil. Der filnavnene som skal ignoreres ligger på hver sin linje.
  (Mest relevant om lokalt repo er opprettet via alternativ måte sist i pkt.8.
  Gjøres det vha. cloning (fra ikke-tomt repo) skjer det mest nødvendige automatisk.)

- Få hjelp (på «ulike» måter): **git help <ord>** eller **git <ord> --help**
  git <ord> -h

Noen ressurser:

- God (omfattende) Git-lærebok: https://git-scm.com/book (spesielt kap. 1, 2, 3 og 5.2)
- https://git-scm.com
- https://about.gitlab.com/images/press/git-cheat-sheet.pdf
- https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet
- https://ndpsoftware.com/git-cheatsheet.html (click inside the page/columns)
- https://www.youtube.com/watch?v=vWdmXunGQC8

Ikke behandlet/omtalt/relevant bl.a:

- **git stash**
- Rulle tilbake / fremhente / se tidligere versjon (commit)