

Versionhallinta ja Git

Osa 2

Sisältö

- Johdanto.
- Tilojen ja versioiden tulostaminen.
- Valmistelualueelle lisääminen, siltä poistaminen ja *.gitignore*.
- Version luontia tarkemmin.
- Versionhallinnassa olevan yksittäisen tiedoston käsittelyä.
- Aiemman version palauttaminen.
 - Palautus historia säilyttäen tai sitä tuhoten.
- Etätietovarastot.
 - Lataaminen, yhdistäminen ja Tuni-GitLab.
- Lähteitä.

Johdanto

- Tässä luvussa palataan aiemmin opittuihin Git-komentoihin ja opitaan uutta.
 - Versionhallinnan hyödyt nousevat tällä kurssilla esiin erityisesti, kun on tarpeen palata aiemmin tehdyn koodin ääreen, koska ohjelmisto saadaan harvoin valmiiksi ilman, että on tarvetta muuttaa alkuperäisiä suunnitelmia tai peräytyä valitusta kehityssuunnasta.
 - Gitissä on komennot yksittäisten tiedostojen ja kokonaisten versioiden palauttamiseen.
 - Etätietovarasto ja siihen liittyvät komennot nousevat sitä keskeisemmäksi mitä laajemmin versionhallintaa käytetään.
-

Johdanto

- Projektin kehityshaaroihin ja niiden yhdistämiseen perehdytään lähinnä siltä osin, mitä etätietovarastoa käytettäessä voi tulla vastaan, koska kehityshaarat ovat aiheena liian suuri kurssilla käsiteltäväksi.
 - Harjoitustyössä voi halutessaan testailla kehityshaaroja, mutta niiden käyttöä ei edellytetä.

Git lähemmin tarkasteltuna

- Tilojen ja versioiden tulostaminen.
 - Tiedostojen tilat (`status`).
 - Versiohistorian ja versioiden tutkiminen (`log`, `ls-tree` ja `show`).
 - Valmistelualue.
 - Lisääminen (`add`) ja valmistelualueelta poisto (`reset`).
 - Tiedostojen lisäyksen esto `.gitignore`-tiedostolla.
 - Version luonti (`commit`).
 - Versionhallinnassa olevan tiedoston käsittelyä.
 - Tiedoston poisto versiohallinnasta (`rm`).
 - Tiedoston palauttaminen (`checkout`).
 - Version palauttaminen (`revert` ja `reset`).
 - Etätietovarastot (`clone`, `push` ja `pull`).
-

Tilojen ja versioiden tulostaminen

- Tarkastellaan aluksi tiedostojen tiloja, joiden englanninkieliset nimet on hyvä tuntea, koska osa niistä esiintyy erityisesti Gitin `status`-komennon tulosteissa.
- Git jakaa työhakemiston tiedostot kahteen päätilaan.
 - *Tracked*: tiedostot, jotka ovat versionhallinnan piirissä.
 - *Untracked*: kaikki muut työhakemiston tiedostot.
- Versionhallintaan kuuluvilla tiedostoilla on tilat (jatkuu seuraavalla sivulla):
 - *Staged*: tiedosto on viety valmistelualueelle `add`-komennolla.
 - *Ignored*: versionhallinta jättää tiedoston huomiotta `.gitignore`-tiedoston ohjaamana.

Tilojen ja versioiden tulostaminen

- Versionhallintaan kuuluvilla tiedostoilla on tilat (jatkoa):
 - *Committed*: tiedosto on sama kuin se on viimeisimmällä `commit`-komennolla tehdyssä versiossa.
 - *Modified*: tiedosto on eri kuin se on viimeisimmässä versiossa tai on eri kuin se on valmistelualueella. Tähän tilaan päädytään, kun tiedostoa muokataan jollain tavoin version tallentamisen tai valmistelualueelle lisäämisen jälkeen.
- Tulostus- ja muissa Git-komennoissa versio yksilöidään usein antamalla Git-komennolle SHA-1-tiiviste joko kokonaisena tai riittävästi merkkejä tiivisteen alusta.

Tilojen ja versioiden tulostaminen

- Log-komento tulostaa versiohistorian. Lyhyempi muoto `git log --pretty=oneline` listaa historian lyhyessä muodossa, jossa rivillä on yhden version tiedot.
- Yksittäisen version sisältämät tiedostot voi listata esimerkiksi komennolla `ls-tree`.
- Listataan esimerkiksi version `640ac9c...` tiedostot: `git ls-tree --name-only -r 640ac9c`
 - *name-only* : jättää riveiltä pois osan tietoja.
 - *r* : listaa näkyviin alihakemistojen sisällöt.
- `show`-komennolla voidaan tulostaa version tiedosto.
- Esimerkiksi: `git show 640ac9c:README`

Valmistelualueelle lisääminen (add)

- Vie uusia – ei vielä Gitin seurannassa olevia – tiedostoja ja jo versionhallinnassa olevia muuttuneiksi havaittuja tiedostoja valmistelualueelle.
- Komennolle voidaan antaa yksittäisen tiedoston lisäksi lisättäväksi hakemisto.
- Lisätään valmistelualueelle *dokumentit*-hakemiston sisältö:
`git add dokumentit`
- Komento toimii rekursiivisesti ja jokerimerkkejä voi käyttää.
- Lisätään valmistelualueelle kaikki Java-lähdekoodi:
`git add *.java`

Valmistelualueelle lisääminen (add)

- Tiedosto lisätään valmistelualueelle juuri siinä muodossa, kun se on add-komentoa annettaessa. Lisäyksen jälkeiset **muutokset eivät päivity valmistelualueelle**.
 - Lisäyksen jälkeen muokattu tiedosto on sekä *modified* että *staged*.
 - Lisää uudelleen, jos haluat uusimmat muutokset mukaan.

Valmistelualueelta poisto (reset)

- Tyhjentää sellaisenaan koko valmistelualueen:
`git reset`
- Valmistelualueelta voidaan ottaa pois myös yksittäinen tiedosto. Esimerkiksi:
`git reset README`
- Reset-komentoa pitää käyttää varovaisesti, koska sitä voidaan käyttää myös kokonaisten versioiden peruuttamattomaan poistamiseen.

.gitignore-tiedosto

- Tekstitiedosto, jonka säännöt määrittelevät mitkä tiedostot versionhallinnan on jätettävä huomiotta.
- Ristikkomerkillä # alkavat rivit ohitetaan.
- Huutomerkillä ! alkavat rivit määrittelevät poikkeuksen edelliseen sääntöön.
- Työhakemistoon sijoitettu *.gitignore* määrittelee säännöt koko hakemistolle.
 - Sääntöjä voidaan tarkentaa alihakemistojen omilla *.gitignore*-tiedostoilla.

.gitignore-tiedosto

- Tällä kurssilla on tärkeää ettei tietovarastoon tallenneta tavukooditiedostoja.
- Projetipohjan työhakemistossa on siksi seuraavanlainen *.gitignore*:

```
# Estetään tavukoodin lisääminen tietovarastoon.  
*.class
```

- Tiedostoon voi tehdä lisäsääntöjä esimerkiksi estämään mahdollisesti käyttämiesi apuohjelmien lisäys.

```
# Estetään tavukoodin ja apulaisen lisäys.  
*.class  
dos2unix.exe
```

Version luonti (commit)

- `commit`-komento luo projektin versionhallinnassa olevista muuttumattomista tiedostoista (*committed*) ja valmistelualueella olevista tiedostoista (*staged*) uuden version paikalliseen tietovarastoon.
 - Versioon on liitettävä lyhyt, mutta kuvaava kommentti esimerkiksi siitä mitä uuttaa versiossa on.
- Komento itsessään eli pelkkä `commit` avaa tekstieditorin kommentin kirjoittamista varten.
 - Versio tallentuu, kun olet tallentanut kommentin ja sulkenut editorin. Huomaa, että ristikkomerkillä `#` alkavat rivit eivät kirjoitu kommenttiin ja ilman kommenttia tallennus keskeytyy.

Version luonti (commit)

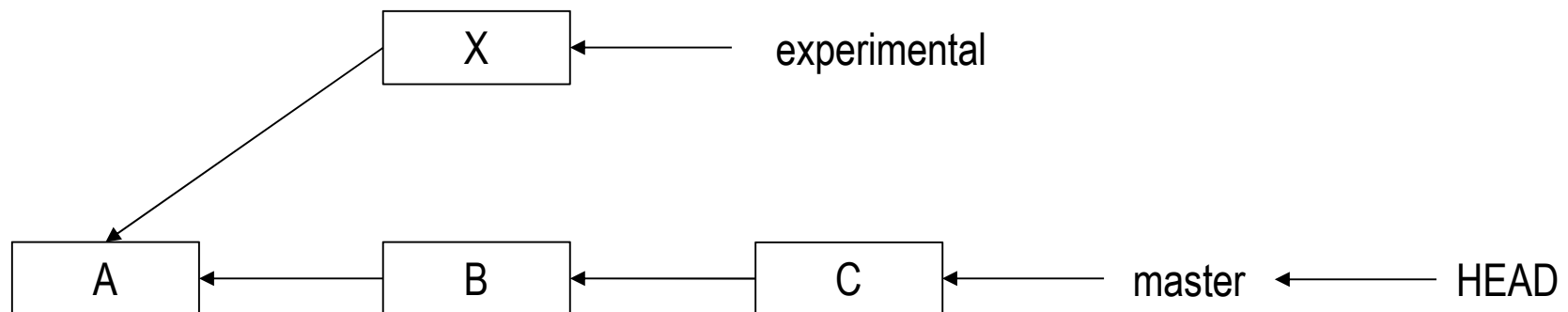
- Kommentti voidaan antaa suoraan komentoriviltä *m*-valitsimella: `git commit -m "tähän hyvä kommentti"`
- Kaikki muuttuneet tiedostot voi tallentaa uuteen versioon ilman valmistelualueen käyttöä *a*-valitsimella: `git commit -a`
- Uusinta versiota voi muokata *amend*-valitsimella.
 - Tämä voi olla tarpeen, jos esimerkiksi kommentissa on kirjoitusvirhe tai versiota unohtui tiedosto.
 - Tee muokkaus, kun on pientä korjattavaa, jotta versionhallintaan ei tulisi turhia versioita, joissa ei ole oikeastaan uutta.
 - Muokattu versio **korvaa** uusimman version. Tämä aiheuttaa ongelmia, jos korvattu versio on jo etätietovarastossa.
 - Esimerkiksi: `git commit --amend -m "korjattu kommentti"`

Version luonti (commit)

- Kukin versio sisältää tallennettujen tiedostojen ja metadatan lisäksi SHA-1-tiivisteen (viite, reference, ref) ja viitteen edelliseen versioon.
 - **Versiohistoria** muodostetaan keskinäisten viitteiden kautta.
 - Viitteen 40 merkistä 7–10 ensimmäistä riittää version yksilöintiin.
- Projektin haarat yksilöidään nimettyjen viitteiden avulla, jotka liittyvät haaran uusimpaan versioon.
 - Päähaaraan liittyy automaattisesti *master*-niminen viite.
- Projektin työskentelyhaara valitaan liittämällä siihen *HEAD*-viite, joka liittyy ilman eri toimia haaran uusimpaan versioon.

Version luonti (commit)

- Projekti, jossa on päähaara *master* ja kokeellinen haara *experimental*. Päähaara on aktiivinen, koska *HEAD*-viite liittyy päähaaraan. Päähaarassa on kolme versiota A, B ja C. Kokeellisessa haarassa on versio X.

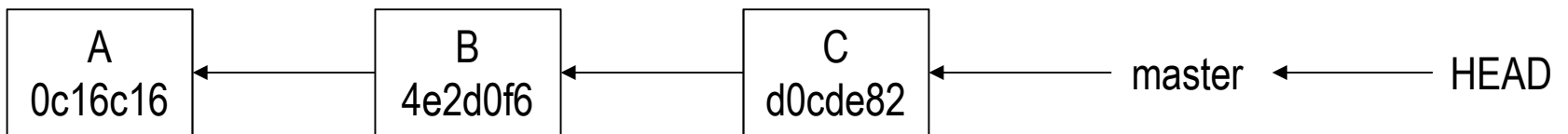
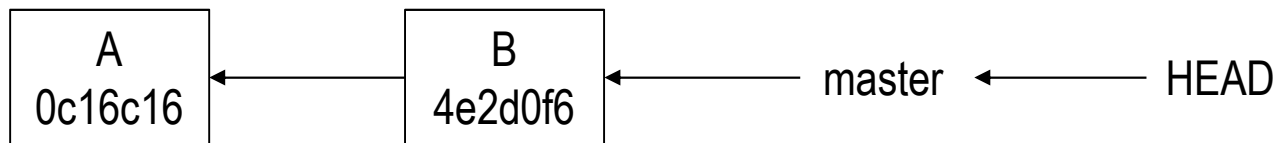
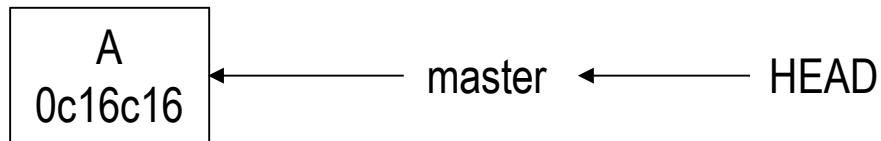


Version luonti (commit)

- Uusi versio lisätään aktiiviseen haaraan ja samalla haaran nimeävä viite ja *HEAD*-viite liitetään uuteen versioon.
 - Harjoitustyön tapaisessa, pääsääntöisesti pelkän päähaaran sisältävässä projektissa on harvemmin tarvetta siirtää viitteitä itse.
 - *HEAD*-viite, joka on ei liity minkään haaraan uusimpaan versioon, vaan on kiinnitetty johonkin vanhempaan versioon, on irtonainen (detached).
 - *HEAD*-viitettä edeltävään versioon voidaan viitata suhteellisesti. *HEAD~1* (tai *HEAD~*) viittaa toiseksi uusimpaan versioon, *HEAD~2* viittaa kolmanneksi uusimpaan versioon ja niin edelleen.

Version luonti (commit)

- Luodaan projektin päähaaraan versiot A, B ja C. *HEAD*-viite tarttuu automaattisesti aina kaikkein uusimpaan versioon.



Version luonti (commit)

- Versioi säännöllisesti, jotta voit tarvittaessa palauttaa tiedostoja.
 - Jokainen versionhallintaan tallennettu tiedosto on palautettavissa. Muut tiedostot saa takaisin vain, jos niistä muunlainen varmuuskopio.
- Tarkista ennen komennon antamista, että olet lisäämässä versioon vain sellaisia tiedostoja, joita kuka tahansa voi tarkastella.
 - Tiedosto voidaan poistaa, mutta sen kaikkien jälkien poistaminen versionhallinnasta on vaikeaa.
 - Ota yhteyttä kurssin opettajiin, jos on tarve poistaa tiedosto, mutta et tiedä kuinka toimia.

Version luonti (commit)

- Versionhallintaan ei saa vielä mitään turhaa, jotta rajallisia resursseja ei tuhlaata ja projektin tärkeimmät osat eivät huku “roskan” sekaan.
 - Erityisesti lähdekoodista kääntämällä saatavia tiedostoja on turhaa tallentaa versionhallintaan, koska ne voivat olla suuria ja muuttuvat koko ajan, jolloin pahimmillaan turhista tiedostoista tehdään koko ajan turhia uusia versioita.
 - Älä tallenna pahasti keskeneräistä koodia, jossa on esimerkiksi suuria pois kommentoituja alueita.
- Version tulisi olla toimiva ainakin siten, että koodi kääntyy.
 - Useamman hengen projekteissa versionhallintaan ei saa vielä millään tavalla “rikkinäistä” koodia.

Tiedoston poisto (rm)

- Lopettaa tiedoston seuraamisen ja **poistaa tiedoston** työhakemistosta.
 - Tämän komennon kanssa on luonnollisesti oltava varovainen!
- Esimerkki: `git rm Test.java`
- Tiedoston seuraamisen voidaan lopettaa *cached*-valinnalla siten, että **tiedostoa ei poisteta**.
- Esimerkki: `git rm -cached Test.java`

Tiedoston palauttaminen (checkout)

- Projektissa voi tulla vastaan tilanne, jossa huomataan, että on pakko palata johonkin aiempaan versioon yhdestä tai useammasta tiedostosta. Tällöin versionhallinta pelastaa päivän, jos vain versioita on tallennettu riittävän usein.
- checkout-komento palauttaa tiedoston tai tiedostoja siten, että työhakemiston tiedostot korvataan joko viimeisimmästä tai erikseen määritellystä versiosta luetuilla tiedostoilla.
 - Tätä komentoa käytetään pääasiallinen käytätapa on projektin haaran aktivointi *HEAD*-viitettä siirtämällä. Komento tulee tutummaksi tässä yhteydessä, kun työskentelet projektissa, jossa on useampia haaroja.

Tiedoston palauttaminen (checkout)

- Palautusta **ei ole mahdollista peruuttaa**.
 - Tallenna korvauksen kohteena olevat muokatut tiedostot jonnekin, jos on mahdollisuus, että niillä on sittenkin käyttöä.
- Korvataan *README*-tiedosto viimeksi tallennetulla versiolla: `git checkout -- README`
- Palautettaessa jostain muusta kuin viimeisimmästä versiosta, versio yksilöidään viitteen avulla.
- Korvataan työhakemiston *README*-tiedosto version `0c16c16c1f5dcaba8d1bfa85d9481732743e9728` sisältämällä *README*-tiedostolla: `git checkout 0c16c16c - README`

Version palauttaminen (revert)

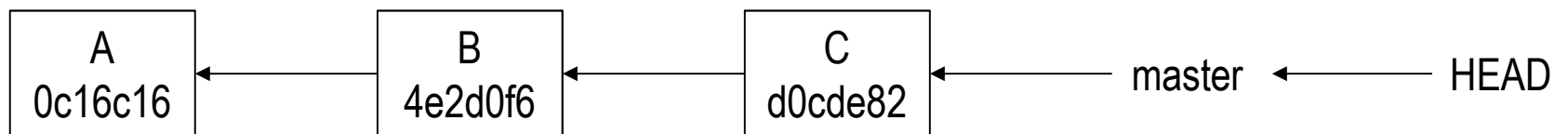
- Kokonaiseen aiempaan versioon palaaminen on mahdollista `revert`-komennolla siten, että komennolla perutaan yhden tai useamman viimeisimmän version muutokset kunnes ollaan oikeassa versiossa.
 - Komennolla voidaan peruuttaa myös historian “väliversioita”, jolloin ei palauteta suoraan aiempaa versiota, vaan perutaan valikoiden useita huonoksi osoittautuneita muutoksia.
 - Version palauttaminen voi olla helpompaa palauttamalla tarpeelliset tiedostot yksitellen `checkout`-komennolla.
- Jokainen peruutus tuottaa uuden version.
 - Historia säilyy katkeamattomana eikä tuota ongelmia etätietovarastoa käytettäessä.

Version palauttaminen (revert)

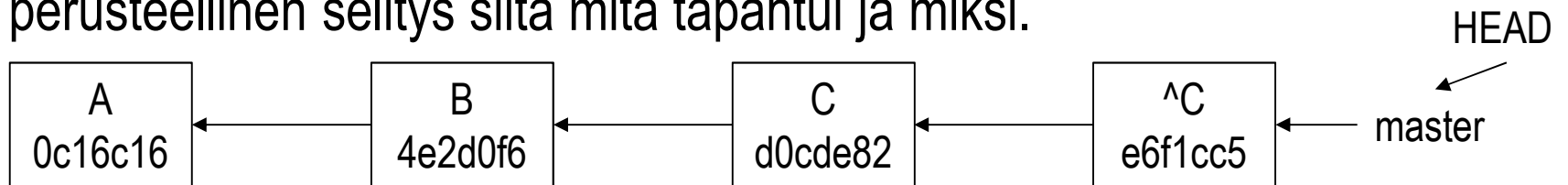
- Peruuttaa voidaan vain, kun työhakemistossa ei ole muutoksia ja valmistelualue on tyhjä.
- Version 045b89ae33f5ed47f6f15a856861aabfde20dcb5 muutosten peruminen: `git revert 045b89a`
- Perutaan esimerkiksi viimeisimmän ja toiseksi viimeisimmän version muutokset:
`git revert HEAD`
`git revert HEAD~2`
 - Toinen komento täytyy kohdistaa kolmanneksi viimeiseen versioon, koska ensimmäinen komento lisäsi versionhallintaan version, jossa on ennen komennon suorittamista viimeisimpänä olleessa versiossa tehdyt muutokset.

Version palauttaminen (revert)

- Oletetaan, että C-version tallennuksen jälkeen tehdään vähän aikaa töitä ja huomataan, että ollaan menossa väärään suuntaan. Päätetään, että on parempi palata B-versioon. Poistetaan ensin muuttuneet tiedostot `checkout`-komennolla ja tyhjennetään valmistelualue `reset`-komennolla, jotta voidaan käyttää `revert`-komentoa.



- `git revert HEAD` peruuttaa kaikki C-versiossa tehdyt muutokset ja tekee uuden version \wedge C, jonka tiedostot ovat samat kuin B-versiossa. C jää muistoksi versiohistoriaan. \wedge C-version kommenttiin kirjoitetaan perusteellinen selitys siitä mitä tapahtui ja miksi.



Version palauttaminen (revert)

- Komennolle voidaan antaa myös parametriksi useita viitteitä. Edellä annetut komennot voidaan yhdistää näin:
`git revert HEAD~2..HEAD`
 - Väliä käytettäessä ensimmäinen viite on vanhempi ja toinen nuorempi eikä ensimmäinen viite kuulu väliin.
- Käyttäjän on päätettävä mitä tehdä, jos palauttamisessa ilmenee konflikti.
 - Käyttäjää tarvitaan esimerkiksi, kun välistä poistaminen johtaa tiedoston poistamiseen nykyisestä versiosta.
 - Konfliktin ratkaisun jälkeen jatketaan: `git revert --continue`
- Peruuttaminen voidaan keskeyttää (`git revert --abort`) tai peruuttaa peruutuksen peruutuksella.

Version palauttaminen (reset)

- Reset-komento *hard*-valinnalla **tuhoaa** kaikki parametriksi annettuun viitteeseen liittyvää versiota **uudemmat versiot peruuttamattomasti** ja kiinnittää haaran nimetyn viitteen uuteen viimeisimpään versioon.
 - Esimerkiksi komento `git reset --hard HEAD~2` tuhoaa viimeisimmän (*HEAD*) ja toiseksi viimeisimmän (*HEAD~1*) version.
 - Ilman parametria eli `git reset --hard`, komento peruu kaikki muutokset työhakemistossa ja tyhjentää valmistelualueen. Tämä on hyödyllistä, kun halutaan antaa `revert`-komento.
- Komento **tuhoaa versiohistoriaa**.
 - Paikallisen tietovaraston yhdistäminen etätietovarastoon (`push`) vaatii enemmän työtä, jos paikallisesti tehdään uusia versioita `reset`-komennolla katkaistuun historiaan pohjautuen.

Version palauttaminen (reset)

- Käytä tätä menetelmää vain, kun a) sinulla on **varmuuskopio** työhakemistosta ja versiohallinnasta ja b) tiedät täsmälleen mitä olet tekemässä.
- **Kysy opettajalta, jos olet vähänkin** epävarma onko täysituho oikea ratkaisu sinulle.
- Lue tarkkaan mikä tahansa ohje, jossa neuvotaan käyttämään `reset`-komentoa *hard*-valinnalla.
- `Reset`-komentoa voidaan käyttää vähemmän vaarallisesti *soft*- ja *mixed*-valinnoilla.
 - <https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified>

Etätietovarastot

- Etätietovarasto on keskeinen työväline tiedon jakamiseen useamman hengen projekteissa.
- Yksin yhdellä koneella ohjelmoidessa etävarasto on enemmän **varmuuskopion** luonteinen.
 - Varmuuskopioita voi tehdä edelleen myös kopioimalla työhakemistoa, jolloin samalla saadaan varmuuskopio versionhallinnasta. Kopiointi kannattaa tehdä tilan säästämiseksi tiivistämällä tiedot esimerkiksi zip-muotoiseksi paketiksi.
 - Etävarasto voi olla myös massamuistilla, jolloin varmuuskopioita voi tehdä siten, että paikallisessa varastossa oleva projekti tallennetaan palvelimella olevan etävaraston sijasta esimerkiksi usb-tikulla olevaan tietovarastoon.

Etätietovarastot

- Pidä etätietovarastosi **ajan tasalla**.
 - Vanhentuneita versioita sisältävästä etävarastosta ei ole hyötyä katastrofin kohdatessa paikallista varastoa.
 - Lataa paikallisesti tekemäsi työsi etävarastoon muun muassa, kun olet saavuttanut jonkin välipisteen ja ennen kuin pidät pitempää taukoa, jolloin on varmaa, että uusin versio on aina tallessa etävarastossa.
- Kaikkien tärkeintä on muistaa tehdä **jollain tavalla varmuuskopioita**, vaikka versionhallinta on käytössä, koska massamuistilaitteen rikkoutuessa versiot sisältävä paikallinen tietovarasto voi tuhoutua yhtä hyvin kuin työhakemistossa oleva projekti.

Etätietovarastot

- Etätietovaraston kautta voi siirtää näppärästi tietoa, kun yhden hengen projektia tehdään eri koneilla.
 - Tietoja siirrettäessä verkossa oleva etävarasto on selkein ratkaisu varsinkin, jos koneissa on eri käyttöjärjestelmät.
 - Versiohallinnoitujen tietojen siirto eri tietokoneiden välillä ilman etävarastoa on virhealtista.
 - Käytä USB-tikun tapaiselle siirrettävälle massamuistilaitteelle perustettua etävarastoa vain, jos et voi siirtää tietoja verkossa olevan varaston kautta.
- Verkossa oleva etätietovarasto pitää suojata, jotta kuka tahansa ei voi ladata harjoitustyötäsi omalle koneelleen.
 - Tuni-GitLabia käytettäessä täytyy kirjautua tunnusta ja salasanaa käyttäen. GitLabissa on keinoja helpottaa kirjautumista. Myös SSH-avaimilla pääsee nopeammin ovesta sisään.

Etätietovarastot

- Kahta tai useampaa konetta käytettäessä on oltava tarkkana, että työtä jatketaan paikallisesti varmasti kaikkein uusimmasta harjoitustyön versiosta.
 - Koneita X ja Y käytettäessä on esimerkiksi mahdollista, että koneella X tehty työ jää lataamatta etävarastoon tai että etävarastoon ladattu työ jää lataamatta koneelle Y, jolloin koneella Y tehty työ perustuu vanhaan versioon, jos ohjelmoija ei muista korjata unohdustaan ennen version tallennusta koneella Y. Unohduksen seurauksena koneella Y tehty työ ei perustu koneella X tehtyyn uusimpaan työhön ja edessä saattaa olla eri koneilla olevien projektien *master*-haarojen yhdistäminen vaikeammalla tavalla.

Etätietovarastot

- Harjoitustyössä etävarastosta paikalliseen varastoon ladattaessa tai päinvastoin toimiessa tietojen (*master*-haarojen) pitäisi **yhdistyä suoraan** (fast-forward merge).
 - Projektin haarojen **aito yhdistämisen** (true merge) on harjoitustyössä ajankohtaista lähinnä vain, jos
 - paikallisesta varastosta on tuhottu versio voimalla (hard reset) ja jatkettu töitä katkenneeseen historiaan pohjautuen,
 - eri koneilla olevien tietovarastojen versiohistoriat pääsevät eriytymään unohduksen seurauksena (edellinen sivu) tai
 - `commit`-komento suoritetaan *amend*-valinnalla sen jälkeen, kun korvattava versio on ladattu etävarastoon (push-komento).
 - **Ota tarvittaessa yhteyttä opettajaan**, kun on tehtävä yhdistäminen, joka ei suostu tapahtumaan automaattisesti.
-

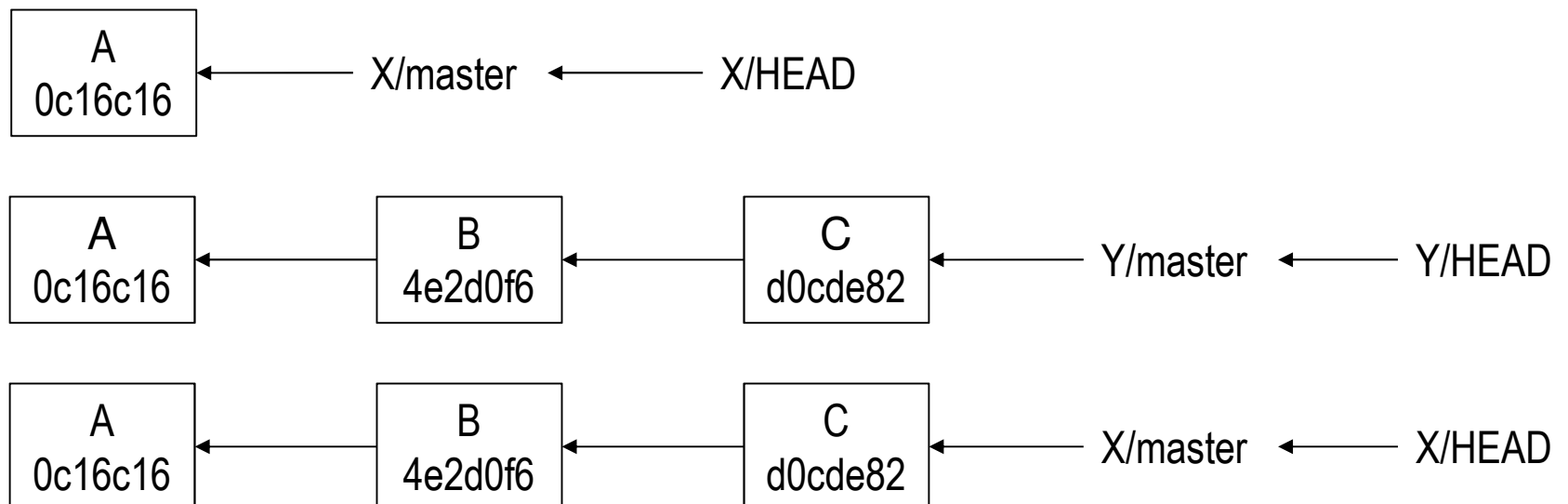
Etätietovarastot

- Oletetaan, että tietovarastossa X olevan projektin *master*-haaraan ollaan yhdistämässä tietovarastossa Y olevan projektin *master*-haaraa.
- Haarat voidaan yhdistää automaattisesti suoraan, jos Y:n *master*-viitteeseen liittyvä versio V_Y on X:n *master*-viitteeseen liittyvän version V_X jälkeläinen.
- **Suoraan yhdistämiseen** vaaditaan näin V_Y :n ja V_X :n välinen periytymispolku eli keskinäistä historiaa.
- Alla versiot B ja C periytyvät versiosta A ja C periytyy B:stä.



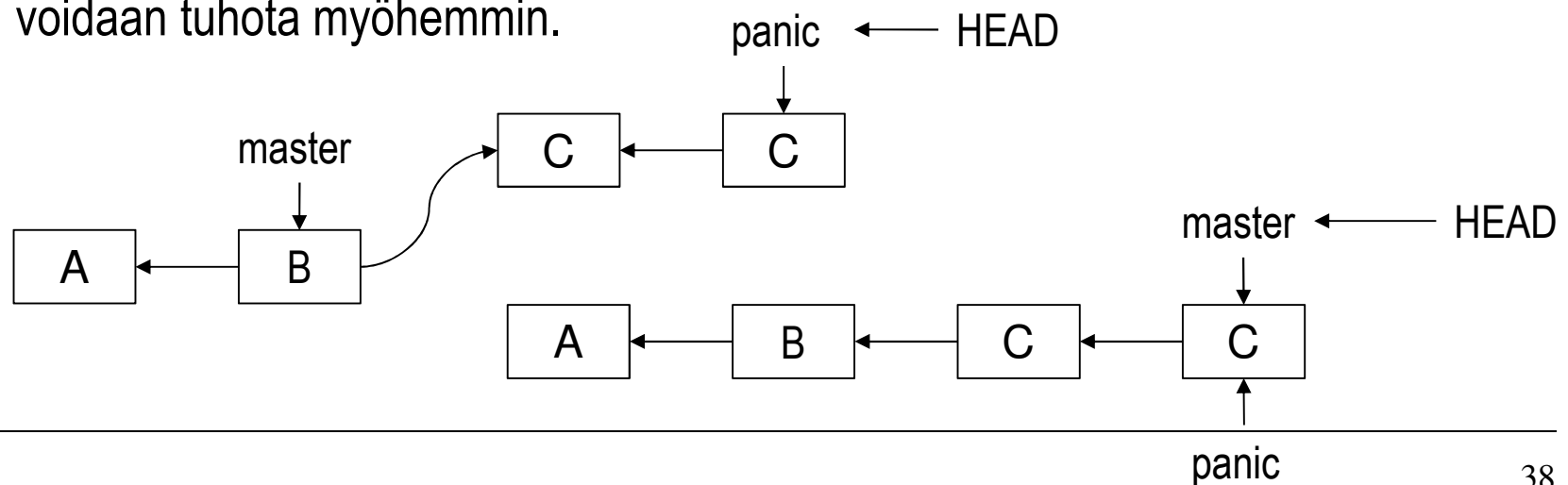
Etätietovarastot

- Yhdistetään tietovarastojen X (versio A) ja Y (versiot A, B ja C) *master*-haarat. Yhdistäminen onnistuu, koska versio C periytyy versiosta A. Tietovarastosta X puuttuvat versiot B ja C kopioidaan tietovarastosta Y ja *master*-viitteen paikka päivitetään.



Etätietovarastot

- Suoran yhdistämisen englanninkielinen nimi *fast-forward merge* ilmentää perimmiltään sitä, että yhdistäminen voidaan tehdä vain nimettyjä osoittimia siirtämällä. Eri tietovarastoissa olevia haaroja yhdistettäessä tämä ei ole suoraan ilmeistä. Samassa varastossa olevan projektin eri haarojen kanssa työskennellessä nimen tausta käy paremmin ilmi. Alla annetussa esimerkissä *panic*-haara on luotu *master*-haaran päästä. *Master* koostuu versioista A ja B ja *panic* versioista C ja D. *Panic*-versioiden valmistuttua paniikki on ohi ja haarat voidaan yhdistää suoraan liittämällä *master*-viite samaan versioon, johon *panic*-viite liittyy. *Panic*-haaralle ei ole yhdistämisen jälkeen käyttöä. Se voidaan tuhota myöhemmin.



Etätietovarastot (clone)

- `clone` kopioi etätietovaraston paikalliseksi tietovarastoksi.
- Esimerkiksi komento `git clone https://gitlab.com/csjola/test` muodostaa GitLabin *csjola*-käyttäjän *test*-nimisestä projektista paikallisen kopion *test*-nimiseen hakemistoon.
- Git luo kopioinnin yhteydessä *origin*-nimisen aliaksen etätietovarastolle, jotta varaston käyttö olisi helpompaa.
- Näet käytössä olevat aliakset komennolla `git remote -v`
- Aliaksen voi asettaa myös itse. Esimerkiksi: `git remote add origin https://gitlab.com/csjola/test`
 - Ennen asetusta on tarpeen `remote rename` tai `remote remove`, jos alias on jo käytössä.

Etätietovarastot (pull)

- `PULL`-komento lataa etävarastosta paikalliseen tietovarastoon siitä puuttuvat versiot ja yhdistää haarat.
 - Paikallinen tietovarasto ei muutu, jos etävarastossa ei ole paikallisesta varastosta puuttuvia versioita.
 - Lataus voidaan kohdistaa johonkin etätietovaraston haaraan.
 - Oletusarvoisesti ladataan haarasta, johon etävaraston *HEAD* liittyy.
 - Esimerkiksi `git pull origin master` lataa versiot *origin*-aliakseen liittyvän etävaraston *master*-haarasta.
 - Harjoitustyössä riittää yleensä pelkkä `git pull`
- Komento yrittää suorittaa suoran yhdistämisen ja yrittää sen epäonnistuessa aitoa yhdistämistä.
 - Aidossa yhdistämisessä voidaan tarvita ihmisen päätöksiä.

Etätietovarastot (pull)

- `pull`-komento suorittaa `fetch`- ja `merge`-komennot, joihin olisi hyvä siirtyä sitten, kun Git on tullut tutummaksi.
- Kahta komentoa käyttäen on helpompi huomata, jos onkin vahingossa yhdistämässä paikalliseen tietovarastoon väärää etätietovaraston haaraa.
- Voit tarkastella yhdistämisen tulosta visuaalisesti komennoilla `git log -graph` ja `gitk`, jos `pull`-komento vaikuttaa tekevän jotain yllättävää.
 - Nämä komennot ovat hyödyllisiä muutenkin.

Etätietovarastot (push)

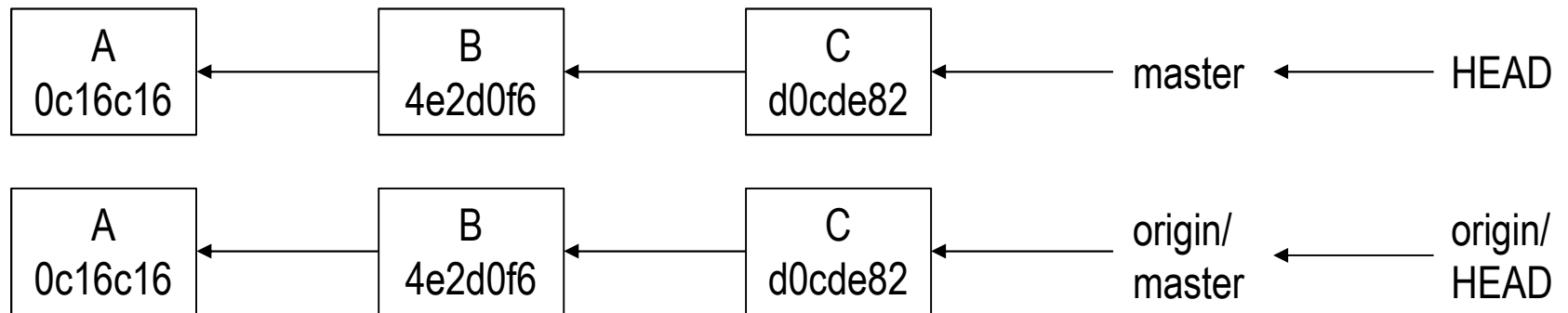
- `push` lataa paikallisesta tietovarastosta etävarastoon etävarastosta puuttuvat versiot ja yhdistää tietovarastot.
 - Lataamisen yhteydessä voidaan määritellä etätietovarasto ja ladattavan haaran nimi.
 - Esimerkiksi komento `push origin master` lataa projektin päähaaran aliakseen *origin* liittyvään etävarastoon.
 - Harjoitustyössä riittää yleensä pelkkä `git push`
- Lataaminen onnistuu vain, jos etätietovarastossa ei ole uudempia versioita kuin paikallisessa varastossa.
 - Toisin sanoen yhdistämisen pitää onnistua suoraan.
 - Näin toimitaan, koska etävarastossa ei ole ihmistä päättämässä mitä tehdä koneälyllä ratkeamattomissa konfliktitilanteissa.

Etätietovarastot (push)

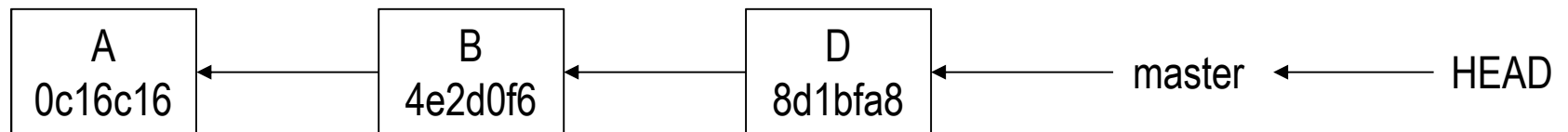
- Lataamisen epäonnistuesssa on suositeltavaa yrittää yhtenäistää historia etävarastosta lataamalla (`pull`).
 - Yhden hengen projekteissa etävaraston historia voidaan hätätilanteessa korvata paikallisella historialla pakottamalla lataaminen etävarastoon (`push --force`).
 - Pakkolatausta ei saa tehdä koskaan, jos projektissa on muita henkilöitä, koska etävaraston historian pyyhkimisen jälkeen paikallisesti vanhan historian parissa jatkaneilla projektilaisilla (kaikilla muilla kuin sinulla) on edessä ongelma etävarastoon ladattaessa.
 - Älä käytä *amend*-valintaa `commit`-komennossa, jos olet ehtinyt ladata uusimman version etätietovarastoon.
 - *Amend* tuhoaa tietovarastojen uusimpien versioiden keskinäisen historian, koska se tuhoaa ja korvaa paikallisen varaston uusimman version.
-

Etätietovarastot (push)

- Oletetaan, että paikallinen tietovarasto on juuri ladattu etätietovarastoon *origin*.

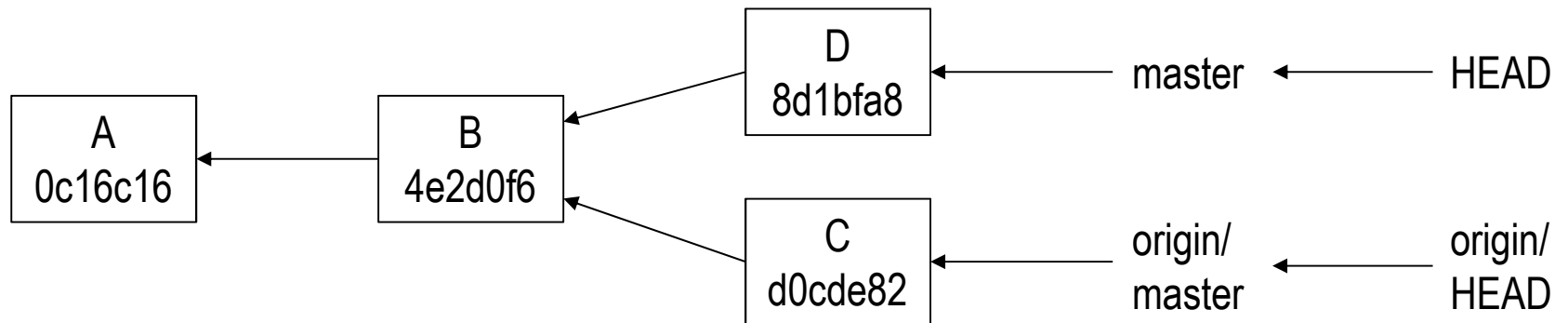


- Tehdään pieni paikallinen korjaus `commit`-komennon `--amend`-valintaa hyödyntäen. C-versio katoaa paikallisesta versiohistoriasta, koska se korvataan korjatulla versiolla D.



Etätietovarastot (push)

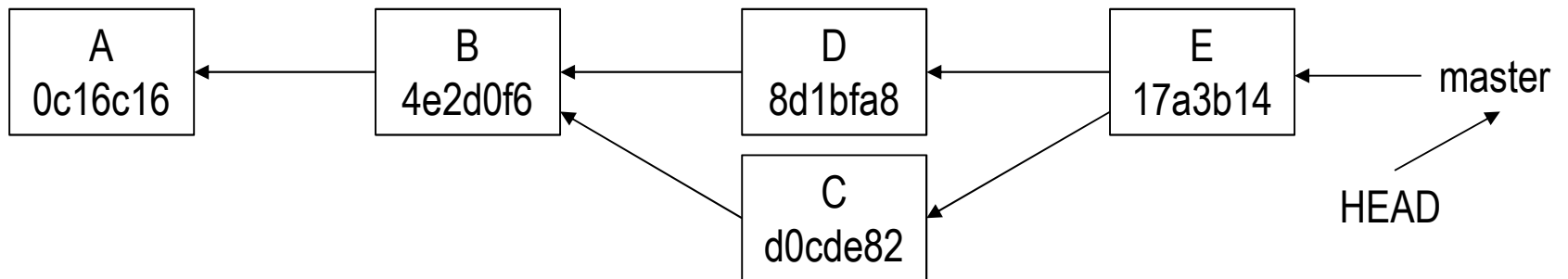
- Versiot C ja D muodostavat yhdessä tarkastellen versiosta B “periytyvät” haarat, joiden yhdistäminen ei ole mahdollista suoraan keskinäisen historian puuttuessa.
 - C:llä ja D:llä on edelleen yhteistä aiempaa historiaa, koska molemmat periytyvät versioista A ja B.



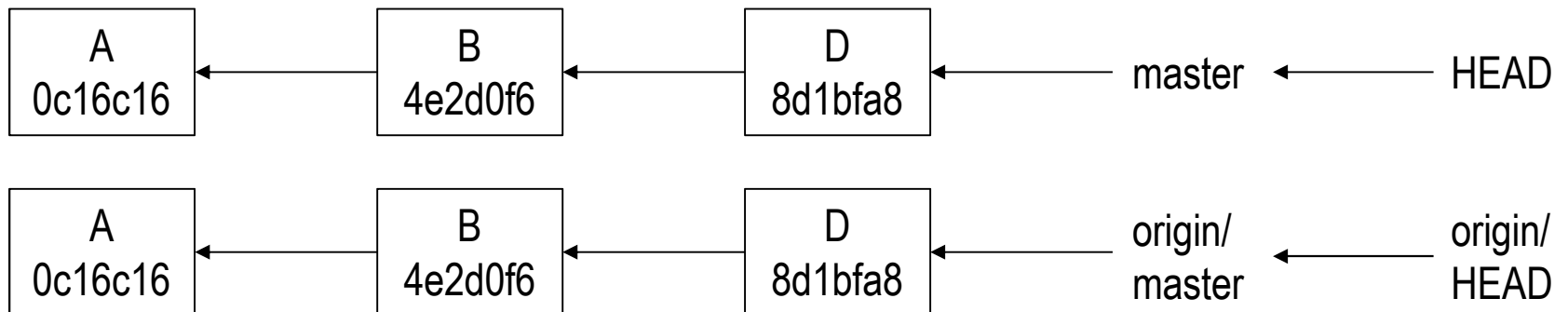
- Historia katketessa korjaus voi olla vaikeaa.

Etätietovarastot (push)

- `PULL`-komennolla tehty yhdistäminen tuo haarat yhteen. Etätietovaraston uusin versio C jää sivuhaaraan. Tulos voidaan ladata nyt etävarastoon, koska E periytyy C:stä.



- Pakkolataus pyyhkii etävaraston historian.



Tuni-GitLab

- Tuni-GitLabiin pääsee sisään Tuni-tunnuksella ja sen salasanalla osoitteessa <https://course-gitlab.tuni.fi/>.
 - Verkkokäyttöliittymä on sama kuin julkisessa GitLabissa.
 - Huomionarvoisia työkaluja ovat muun muassa
 - historian listaus (History-nappi ja Repository | Commits),
 - projektin historian graafinen visualisointi *gitk*-komennon tapaan (Repository | Graph) ja
 - projektin tilastot (Repository | Charts).
 - Omatoiminen ihmettely ja keksiminen on paras tapa oppia tuntemaan GitLab, koska kurssin puitteissa ei ole mahdollista kertoa paljoa.
 - Ole kuitenkin varovainen myös etävarastossa. **Tee muutoksia vain paikalliseen varastoon**, jotta projektinhallinta on helpompaa.
-

Lähteitä

- Ilmainen Pro Git -kirja: <https://git-scm.com/book/en/v2>
- Git-komentojen kuvaukset: <https://git-scm.com/docs>
- GitLabin LML Markdown:
<https://docs.gitlab.com/ee/user/markdown.html>