Versionhallinta

GitLab

Muutos on jatkuvaa

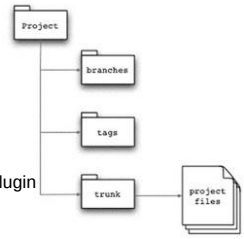
- Ohjelmistoihin kohdistuvat muutokset johtuvat
  - muutoksista liiketoiminnassa
  - muutoksista käyttäjän tarpeissa
  - uudelleenorganisoinnista tai liiketoiminnan kasvusta/vähenemisestä
  - muutoksista budjetissa tai aikataulussa
- Aiheuttavat muutoksia
  - suunnittelutason dokumentteihin
  - koodiin
  - dataan
  - testeihin
  - projektisuunnitelmaan
- Korjaava, mukauttava, kehittävä tai ennaltaehkäisevä muutos

Versionhallinta

- Ohjelmistotyötä hallitaan versionhallintaohjelmiston avulla
  - takaa stabiilimman kehitysympäristön
  - estää tietojen päällekkäiset editoinnit
  - työskentely ei häiritse muita
- Repository
  - (kansioiden) säilytyspaikka, säilyttäjä, repositorio
  - palvelimella ylläpidettävä hakemistorakenne, jonne talletetaan
    - kokonaisia tiedostoja
    - muutostietoja eli metadataa
- Tapa merkitä kaikki tuotokset "nimilapuilla"
  - mahd. palata takaisin vanhoihin versioihin
  - mahd. luoda uudestaan joskus aiemmin koottu kokonaisuus
- Ylläpitää hallinta-alkioiden versioita ja päivitystietoja
  - muutokset edelliseen versioon (delta, backward delta, metadata)
  - kuka muutti, milloin muutti, miten muutti

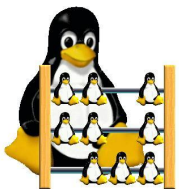
Subversion (SVN)

- Alkujuuret UNIX-skripteissä
  - pohjautuu CVS:ään (Concurrent Version System)
  - saavutti suosiota verkko-ominaisuuksiensa vuoksi
- Keskitetty versionhallinta
  - yksi palvelin, jossa projektin repositorio
  - itsellä oma versio muokattavasta tiedostosta
- Käyttö aluksi komentoriviltä, nykyisin IDE:stä
  - NetBeans: sisäänrakennettuna, Eclipse: erillinen plugin
- Tehokas versiopuiden käsittely
  - koko hakemistopuun voi tallettaa omaksi versioksi
  - osaa lomittaa versiohaaroja takaisin päähaaraan
- Hakemistojen ja tiedostojen uudelleennimeämisen voi versioida, samoin metadatan
  - tiedostonimen tai attribuuttien muuttaminen tulkitaan muutokseksi samaan tapaan kuin tiedoston sisällön muuttaminen
- http://subversion.apache.org/
- http://svnbook.red-bean.com/nightly/en/svn-book.html



Git

- Gitin kehittämisen aloitti Linus Torvalds, v. 2005
  - Linux-kernelin hallinnassa käytetty BitKeeper lakkautti ilmaisversion
- Koostuu useista pienistä perusohjelmista
  - käyttö komentoriviltä tai
  - IDE:n kautta
    - IDE on vain lisäkikkare, pohjalla samat perusohjelmat
- Hajautettu malli
  - palvelimella yhteinen päärepositorio (origin)
  - itsellä oma repositorio (local) omalla koneella
  - päärepositorion ja oman repositorion välillä siirretään vain tietoa tehdyistä muutoksista
- Jokainen voi kehittää ja testata itsenäisesti omaa osaansa
  - vie muutokset/versio jakeluun vasta, kun toiminta taatusti testattu
- Käsittelee muutoksia kokonaisuuksina
  - ei yksittäisten tiedostojen pohjalta
- https://git-scm.com/



git = ääliö, paskiainen, typerys

Git

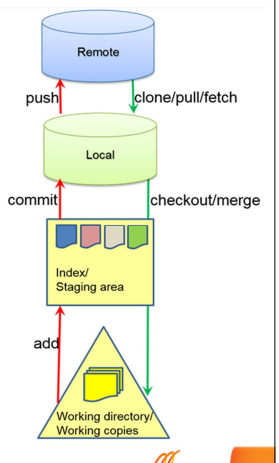
Käytä kurssin projektissa vain yhtä git-tunnusta.

- Iso osa ohjelmoijista käyttää versionhallintaa suoraan komentoriviltä
  - Linuxissa komentoikkunassa, Windowsissa Git Bashissa
  - Tämänkin lärpäkesatsin toiminnot esitetty Git Bash -komentoina
- Monet netissä olevat ohjeet komentorivikäyttöä varten (myös tässä)
- Yksi projektiryhmän jäsen perustaa etärepositorion (origin)
  - GitLabiin, opelle Reporter-oikeudet
  - Kurssin ja ryhmän tiedot selkeästi näkyviin esimerkiksi README.md:ssä (esim. 2022 Kevät – OTP R1 – Tuotteen nimi)
- Muut kloonaavat repositorion itselleen (local)
- OTP 1/2-projekteissa tsekataan myös versionhallinnan käyttöä
  - Kaikilla ryhmäläisillä oltava committeja
- Konfiguroi nimesi ja s-postiosoitteesi Gitiin, ja oletuseditori (Linux)

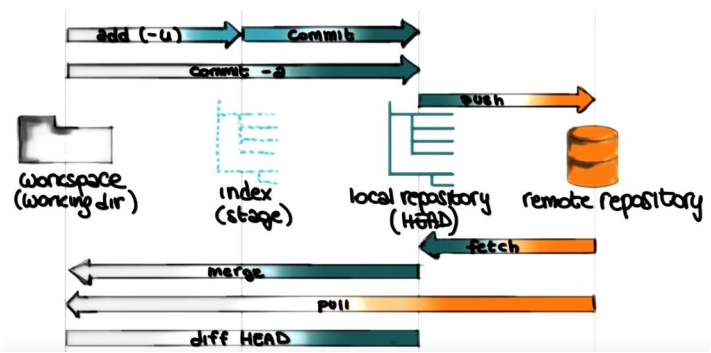
```
git config --global user.name "Oiva Oppija"
git config --global user.email Oiva.Oppija@metropolia.fi
git config --global core.editor nano  Windows: esim. Notepad++
```


Yleiskuva

- Ryhmän yhteinen etärepositorio, palvelimella
  - Ryhmän yhteinen repositorio
- Paikallinen oma repositorio, omalla koneella
  - `git status, git log`
- Index (staging area)
  - staging ~ näytillepano
  - Muuttuneet tiedostot, joita ei vielä ole kommitoitu
- Paikallinen tiedostorakenne
  - Muokattavat työversiot
  - Voi sisältää tiedostoja, joita ei haluta gitiin



Git workflow

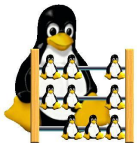


Etärepositorion kloonaaminen

- Olemassa olevan repositorion voi kloonata suoraan palvelimelta
- Kloonaaminen muodostaa automaattisesti yhteyden etärepositorioon ja antaa yhteydelle nimen origin

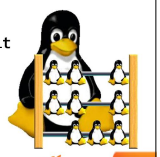
```
git clone http://host/path/to/otprepo.git
git clone ssh://user@host/path/to/otprepo.git
```

- Oletuksena git lataa repositorion hakemistoon, jossa komento suoritettiin
  - Kohdehakemiston voi myös määritellä itse



Olemassaolevan koodin vienti GitLabiin

- Käy ensin GitLabissa luomassa projekti
- Siirry hakemistoon, jonka haluat siirtää versionhallintaan
- Perusasetukset
  - `git config --global user.name "Oiva Opiskelija"`
  - `git config --global user.email "Oiva.Opiskelija@metropolia.fi"`
- Tyhjän paikallisen repositorion saa kohdehakemistossa
  - `git init`
- Koodin voi kytkeä paikalliseen repoon
  - `git add .`
- Lisää etärepon URI
  - `git remote add origin https://gitlab.com/user/repo.git`
- Puske koodit paikallisesta reposta etärepoon
  - `git push -u origin master`



.gitignore

- Kaikkia tiedostoja ei ole järkevää viedä repositorioon
- Merkitse poisjätettävät git-hakemiston juuren tiedostoon `.gitignore`

```
$ cat .gitignore
# Binääritiedostot – ne voi luoda vauhdissa uudestaan
*.com
*.class
*.dll
*.exe
*.o
# Tiedostopakkaukset - tiedostot mieluummin yksitellen
# git hallitsee tiedostojen tiivistämisen (compression methods)
*.7z
*.zip
# Paikalliset lokit ja paikalliset tietokannat
*.log
*.sql
*.sqlite
```


.gitignore

- gitignore-tiedoston tarkoitus määritellä, mitkä tiedostot halutaan pitää versiohallinnan ulkopuolella (tracked vs. untracked)
- Jos haluat lopettaa tällä hetkellä versionhallinnassa olevan tiedoston seurannan, käytä komentoa

```
git rm –cached
```

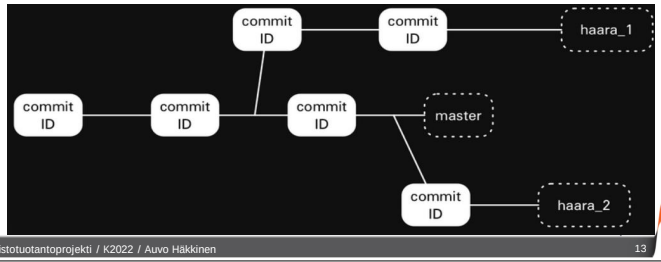
- Jos tiedosto on jo mennyt versionhallintaan, mutta haluat ignorata
  - poista tiedosto ensin repositoriosta
  - lisää .gitignore-tiedostoon huomiottajättämissääntö

```
git rm --cached debug.log
echo debug.log >> .gitignore      lisää loppuun rivin
git commit -m "Start ignoring debug.log"
```

- Jos --cached –valitsinta ei ole, tiedosto poistuu myös paikallisesta hakemistosta

Git branches

- Kehityshaarat ohjaavat inkrementaaliseen kehitystyöhön
  - yksi osa-alue tehdään omassa haarassa valmiiksi
  - lomitetaan vasta sen jälkeen varsinaiseen projektiin
- HEAD osoittaa siihen haaraan, jossa työskennellään
- Pääkehityshaara on nimeltään master
- Siitä tehty kehityshaara ikäänkuin erillinen projekti masterin sisällä

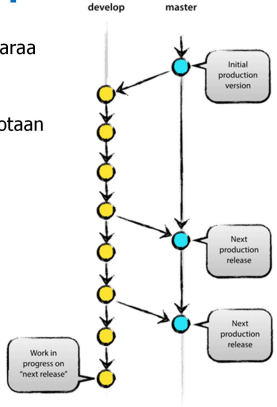


Tyypillinen kehitystyö

- Aloitus
  - Lomita etärepositoriosta nykytilanne itsellesi (origin/master)
- Työskentely
  - Tee itsellesi uusi oma develop-haara, jossa kehität uutta ominaisuutta
  - `git add` siirrä uusi juttu staging-alueelle
  - `git commit` kommitoi muutokset omaan haaraasi
  - Kun kehittämäsi ominaisuus valmis ja testattu, lomita omaan master-haaraasi
- Lopuksi
  - Lomita omat muutoksesi vielä etärepositorioon
- Jokaiseen commitiin tulee liittää asiallinen selitys
  - Niitä voi tutkia `git log` –komennolla

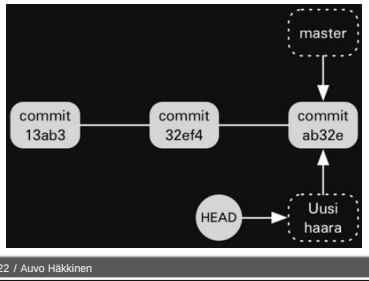
origin/master ja /develop

- Etärepositoriossakin kannattaa pitää kahta haaraa
- origin/master -haarassa projektin yhteinen tuotantoversio (release)
  - se versio, jota demottiin viimeeksi / demotaan seuraavaksi asiakkaalle
  - versio voitaisiin asentaa asiakkaalle tuotantokäyttöön
- origin/develop -haarassa on aina viimeiset kehitysmuutokset projektista
  - joskus tätä sanotaan integrointihaaraksi
  - on se haara, mistä jatkuvan integroinnin palvelin (Jenkins) tekee koonnin
- Kun koodi develop-haarassa on stabiili, lomita se mukaan master-haaraan



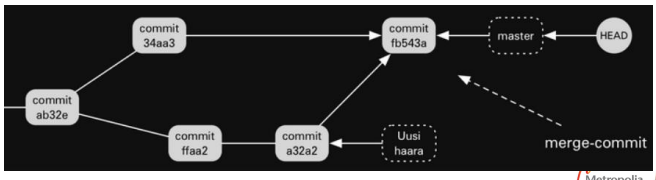
Git branches

- `git branch` listaa mitä haaroja olemassa
- `git branch uusihaara` luo uusi haara
- `git checkout uusihaara` siirry uuteen haaraan (HEAD=uusihaara)
- `git checkout –b uusihaara` luo ja siirry (HEAD=uusihaara)
- `git branch –d uusihaara` poista haara (kun muutokset talletettu)
- `git branch –D uusihaara` poista haara (vaikkei talletettu)



Local: merge

- Lomittaa kaksi haaraa yhdeksi
- Siirry ensin haaraan, johon haluat lomittaa, esimerkiksi master
  - `git checkout master`
- Lomita uuden haaran muutokset nykyiseen haaraan
  - `git merge uusihaara`
- Kirjoita merge-commit viesti
- Jos mukaan liitetyllä haaralla ei ole enää käyttöä sen voi poistaa
  - `git branch -d uusihaara`



Reset

- Kun kaksi haaraa lomitetaan, tallentuu lokiin myös kaikki yhdistettävässä haarassa tehdyt yksittäiset commitit
  - Voi tehdä esim. master-haaran lokista turhan yksityiskohtaisen (turhatkin nippelitiedot kirjattu)
- Yksi vaihtoehto on siivota ennen lomittamista kehityshaarassa tehdyt välivaiheiden commit-merkinnät pois
  - `git checkout uusihaara`
  - `git reset` poistaa kehityspätkän yksittäiset lokitiedot
- Sitten kokonaan uusi commit, johon mukaan kaiken kattava selitys
  - `git commit .` avaa editorin, koska `-m` puuttuu
- Tee merge-operaatio vasta tämän jälkeen

Tagit

- Tunnisteet (tagit) helpottavat commit-lokien tulkitsemista
- `git tag` listaa kaikki tagit
- `git tag -l "v1"` listaa kaikki "v1" alkuiset tagit
- `git show v1.0` näytä tagilla v1.0 merkitty commit
- `git tag -a sprint1 -m "Sprint1 päättynyt" 4abe15`
  lisää tagi commitiin, jonka tunniste 4abe15
- `git tag -a sprint2 -m "Sprint2 päättynyt"`
  lisää tagi viimeisimpään commitiin (tunniste HEAD)
- Tunnisteesta riittää kirjoittaa sen yksilöivä alkuosa
- Mitä muutoksia tehty
  - `git log sprint1..sprint2`
  - `git diff sprint1 sprint2`
  - `git diff sprint2 HEAD`

Tagin voi käydä asettamassa GitLabissa

Origin: fetch ja merge

- Nouda aina ensin etärepositoriossa olevat tuoreimmat tiedot
  - `git fetch origin` tunniste FETCH_HEAD
- Nouto ei vielä lomita niitä paikalliseen repositorioon
- Kun haettu, voi lomittaa omaan paikalliseen repositorioon
  - `git checkout master` nyt HEAD=local master
  - `git merge origin/master`
- Puske lomitettu versio takaisin etärepositorioon
  - `git push` oletuksena origin/master
- Vastaava develop-haaralle
  - `git checkout develop` nyt HEAD=local develop
  - `git fetch origin develop`
  - `git merge origin/develop`
  - `git push origin develop`

Push

- Puskee muutokset etärepositorioon ja lomittaa muutokset
- Jos HEAD=master, muutokset pusketaan ja lomitetaan oletuksena origin/master -haaraan
- `git push` muutokset etärepoon
- `git push origin uusihaara` muutokset etärepoon uuteen haaraan
- `git push origin --all` kaikkien haarojen muutokset etärepooon
- `git push origin --tags` muutokset ja tagit etärepoon

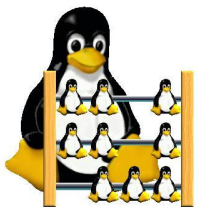
```
Your branch is behind 'origin/master'
```

- Etärepositoriossa muuttuneiden tietojen päälle ei voi puskea vahingossa
  - etärepositorio ja oma paikallinen repositorio lomitettava ensin

Merge-konflikti

- Haarojen lomittamisessa syntyy konflikti, jos saman tiedoston samasta rivistä on olemassa kaksi eri versiota
- Ohjelmoijan on itse ratkottava konfliktit
  - Avaa konflikteja sisältävä tiedosto editorilla
  - Muokkaa tiedostoa
    - poista ei-haluttu pätkä
    - poista myös gitin lisäämät rivit
  - Lisää muokattu tiedosto ja commitoi

```
<<<<< HEAD:tiedosto.txt
tekstiä
======
viriteltyä tekstiä
>>>>>> uusihaara:tiedosto.txt
```



Pull

- `pull = fetch + merge`
- Oletuksena paikallisen repositorion haarat ovat kytkettyinä etärepositorion samannimisiin haaroihin
  - Esim. paikallinen master ja origin/master sekä paikallinen develop ja origin/develop ovat kytkettyinä
- `git pull` oletuksena origin/master
  - Kun HEAD=master käsky olisi sama kuin
    `git fetch origin ; git merge origin/master`

IDE-Ohjesivuja

- IDE:n kautta käyttö on yleensä melko selkeää ja suoraviivaista
  - Kun ymmärtää gitin perulogiikan
- Eclipse Git Tutorial
  - https://www.vogella.com/tutorials/EclipseGit/article.html
- IntelliJ IDEA Git
  - https://www.jetbrains.com/help/idea/version-control-integration.html
- Using Version Control in VS Code
  - https://code.visualstudio.com/docs/editor/versioncontrol
- Using Git in Apache NetBeans
  - https://netbeans.apache.org/kb/docs/ide/git.html