Harjoitus 2: **Subtrees**

Tausta

Subtree on yleensä submodulea parempi vaihtoehto. Sillä ei ole vastaavaa virallista git komentoa kuin submodulella, eli se on enemmänkin konsepti. Konsepti on kuitenkin niin yleinen, että sitä varten Gitin viralliseen distribuutioon on otettu mukaan esimerkiksi git-subtree työkalu (git-subtree.sh). Sitä voidaan käyttää kuten muitakin git komentoja, esim. git subtree

Tässä harjoituksessa käydään läpi manuaalinen tapa

Aloitetaan käyttämällä samaa projektia kuin submodules harjoituksessa, mutta käytä sitä versiota johon ei ole tehty muutoksia.

Alipuun alustus ja haku

- Siirry ohjelma-repositorioon
- Lisätään nyt uusi remote pluginia varten, tämä pääasiassa jatkossa käytettävien komentojen lyhentämiseksi. Sen jälkeen haetaan tiedot FETCH_HEADiin
  git remote add kirjasto ../../remotet/kirjasto
  git fetch plugin
- Seuraavaksi täytyy päivittää staging area (index) ja working directory sillä mitä kirjastossa on. Samalla täytyy luoda sopiva alihakemisto. Tähän helpoin vaihtoehto on git-readtree
  git read-tree --prefix=kirjastot/noppa -u kirjasto/master
- Nyt kun katsotaan status, niin nähdään että kirjaston tiedostot on lisätty indeksiin, eli voimme edetä committiin
- Tee commit esimerkiksi viestillä "Lisätty noppa alipuu"
- Sitten voidaan jo pushata, harjoitus kun on niin suoraan

Alipuun sisältävän repositorion käyttö

- Oletetaan että uusi kehittäjä päättää lähteä tekemään töitä meidän ohjelma-repositoriomme kanssa. Ja taas, simuloidaan sitä yksinkertaisesti luomalla uusi klooni rinnakkaiseen hakemistoon
  cd .. (eli pois repositoriosta, lokaalit hakemistoon)
  git clone ../remotet/ohjelma dev2
  cd dev2
  ls -lR
- Viimeinen ls on vain siksi, että nähdään että myös kirjaston koodit ovat tulleet mukaan. Alipuiden kanssa ei siis tarvitse muistaa yhtään mitään erityistä (submodule komentoa, --recursive optiota yms.) alipuuhan on yksinkertaisesti yksi normaali osa varsinaista repositoriota
- Seuraavaksi katsotaan taas muutamaa käytännön tilannetta ja miten niiden kanssa toimitaan

Kirjaston koodi muuttuu alkuperäisessä repositoriossa

- Siirry nyt lokaalit/kirjasto hakemistoon
- Tehdään taas pari leikki-committia (tai muokkaa koodia jos siltä tuntuu)

```
date > koodia.not
git add koodia.not
git commit -m "Mukamuutos kirjastoon 1"
date >> koodia.not
git commit -am "Mukamuutos kirjastoon 2"
```

- Pushaa muutokset

- Vaihdetaan takaisin ohjelmaan, ja haetaan muutokset. Päivitetään tiedostot käyttämällä squash mergeä, strategiana subtree (takakeno mergerivin lopussa tarkoittaa sitä, että komento jatkuu seuraavalla rivillä)
```
git fetch kirjasto
git merge -s subtree --squash --allow-unrelated-histories \
kirjasto/master
```
- Nyt statuksen pitäisi kertoa, että Indeksissä on uusi tiedosto, joten voidaan tehdä commit vaikka viestillä "Kirjastoa muutettu"
- Katso myös git log --oneline niin näet että kuinka squash mergestä ei jää jälkeä lokiin
- Sitten vain git push ja muillekin ohjelman remote repositoriota käyttäville näkyy uusi kirjasto

Seuraavaksi katsotaan miten ohjelmaan tehtäviä muutoksia kohdellaan. Yhdistetään tässä erilaisia muutoksia:

1. vain alipuu,
2. vain päätaso,
3. sekä päätaso että alipuu,
4. vain alipuu, mutta siten että sitä muutosta ei viedä takaisin kirjastoon

Sitten viedään muutoksia eteenpäin

- Alla on joukko muutoksia ja committeja tekeviä komentoja, jotka voi ajaa yksi kerrallaan tai copy-pastella kaikki kerralla. Katso kuitenkin että kaikki menee kuten pitääkin
- Tee nämä ohjelma repositoriossa
```
echo '// Nopan heitto on helppoa' >>
kirjastot/noppa/index.js
git commit -am "[backportattava] Helpompi noppa"
date >> koodia.not
git commit -am "Vain ohjelmallinen muutos"
date >> kirjastot/noppa/leimoja.txt
date > koodia.ei
git add .
git commit -am "[backportattava] Aikoja (myös ohjelmaa)"
echo '// Ohjelman osa' >> kirjastot/noppa/index.js
git commit -am "Ohjelman vaaatima noppamuutos, ei jatkoon"
```
- Koska osa muutoksista backportataan, eli viedään takaisin nopan varsinaiseen repositorioon, mutta osaa ei, niin yksi käyttökelpoinen työmalli on luoda uusi versiohaara tätä varten
```
git checkout -b noppa-backport kirjasto/master
```
- Seuraavaksi poimitaan ne commitit mistä ollaan kiinnostuneita. Ensin voidaan tarkistella hieman lokia ja muistella mitä tehtiin
```
git log --oneline --stat -5
```
- Seuraavaksi poimitaan masterista ne commitit mitkä ovat backportia varten kiinnostavia, ja tehdään push. Poiminnalla saadaan poistettua viimeisin commit, jota ei haluta viedä kirjastoon asti
```
git cherry-pick -x master~3
git cherry-pick -x --strategy=subtree master^
git log --oneline --stat -2
git push kirjasto noppa-backport
```
- Nyt olemme vieneet muutokset myös kirjastoon, mutta noppa-backport branchiin
- Voit vielä vaihtaa kirjasto repositorioon ja hakea muutokset masteriin, mergellä nyt pull request ei ole paras vaihtoehto

Harjoitus 3: **Työnkulut**

Tehdään Gitin käyttöä vertaileva harjoitus. Atlassianin Git tutoriaali (https://www.atlassian.com/git/tutorials/comparing-workflows) kuvaa neljää yleistä työnkulkua:

- Centralized Workflow
- Feature Branch Workflow
- Gitflow Workflow
- Forking Workflow

Jakaudutaan nyt ryhmiin, ja kukin ryhmä käy yhden työnkulun läpi. Tämän jälkeen

1. esittää siitä pääkohdat,
2. Tärkeimmät/suurimmat hyvät ja huonot puolet, sekä
3. sopiiko se omaan työhön.

Kun kaikki työnkulut on käyty läpi, niin pohditaan yhdessä mikä niistä sopii mihinkin tilanteeseen. Lisäksi tarpeen vaatiessa voidaan tutustua muihinkin työnkulkuihin.