Git

Harjoitustehtävät

Osa 2

Sovelto

Sisällys

Harjoitus 1: **Submodules**

Tausta

Jaetaan projekti pienempiin osiin käyttäen git submodule komentoja

Tehtävä

Luodaan uusi projekti, joka emuloi ohjelmaa ja sen käyttämää kirjastoa. Koodi on JavaScriptiä, mutta koodiin ei välttämättä tarvitse koskea. Pohjaprojekti löytyy opettajalta

Harjoitukseen on otettu *hyvin* paljon mallia blogista: https://medium.com/@porteneuve/mastering-git-submodules-34c65e940407. Sitäkin voi seurata näiden ohjeiden sijaan.

Toimenpiteet

Hae pohjaprojekti opettajan kertomasta paikasta ja tutustu siihen. Projekti on Zip-tiedostossa, sillä se sisältää myös paikallisella levyllä sijaitsevan ulkoisen repositorion

- Pura zip-tiedosto paikalliselle levylle
- Paketin sisällä on kaksi hakemistoa: lokaalit ja ja remotet, kummankin alla on kaksi hakemistoa
  - ohjelma: projektin "pääohjelma", jonka alle teemme submodulen
  - kirjasto: ohjelman käyttämä "kirjasto", josta tehdään submodule
- lokaalit hakemisto sisältää lähdekoodit ja remotet sisältävät remote repositoriot

Submodulen tekeminen

- Siirry lokaalit/ohjelma-projektiin, katso ensin status
- Tee nyt kirjastosta submodule
  ```
  git submodule add ../kirjasto kirjastot/noppa
  ```
- Katso mitä git status näyttää
- Tehdään pieni muutos asetuksiin, sillä haluamme nähdä myös alimoduulien tietoja
  ```
  git config --global status.submoduleSummary true
  ```
- Katso .gitmodules tiedoston sisältö, jos haluat niin myös .git/config
- Vertaa alkuperäisen kirjasto-projektin ja luodun kirjastot/noppa hakemistojen sisältöä
- Siirry nyt kirjastot/noppa hakemistoon, ja katso status siellä. Huomaa myös että aktiivinen repositorio on vaihtunut
- Kun kyseessä on submodule, niin hakemistosta löytyy .git, mutta ei hakemisto vaan tiedosto. Katso vaikka `cat`illa sisältö
- Palaa ohjelma-repositorion juureen (luultavasti cd -), committoi muutokset ja pushaa remoteen

Oletetaan tiimityötä, tehtävän helpottamiseksi teeskennellään toista kehittäjää vaikka ollaankin samalla koneella ja samassa hakemistorakenteessa

- Siirry lokaalit hakemiston juureen
- Kloonaa ohjelma-repositorio uuteen hakemistoon, esimerkiksi dev2
  ```
  git clone ../remotet/ohjelma dev2
  ```
- Siirry uuteen repositorioon (dev2) ja katso sen sisältöä. Totea että submodule ei tullut kloonauksessa mukaan. Status ei kerro mistään ongelmasta, kirjasto/noppa hakemisto on mukana, mutta se on tyhjä

    - Gitin alimoduulien toiminnasta johtuen kloonattu projekti ei ole tietoinen siitä, että siellä pitäisi olla alimoduuli. Se täytyy ensin alustaa, ja sen jälkeen hakea
- Alusta alimoduuuli (dev2 hakemistossa):
  `git submodule init`
- Tarkista status, ja tarkista onko alimoduuli nyt kunnossa
- Haetaan siis alimoduulin sisältö erikseen
- git submodule update
- Nyt kaiken pitäisi olla kunnossa
- Usein init ja submodule ajetaan yhdellä komennolla
  git submodule update --init
- Myös kloonauksen yhteydessä voisi tehdä samat asiat sopivilla optiolla, koitetaan sitä seuraavaksi
- Poista koko dev2 hakemisto
- Kloonaa uudestaan, mutta käyttäen --recursive optiota
  `git clone --recursive ../remotet/ohjelma dev2`
- Siirry dev2:n kirjasto/noppa hakemistoon. Huomaa että branchin nimi on muuttunut. Tämä johtuu siitä, että alimoduuli on *detached head* -tilassa
- Aja `git status`, niin sen pitäisi vahvistaa detached head tila
- Jos näkyvän haaran nimi tulee commitid:stä, niin sen saa muutettua ympäristömuuttujalla GIT_PS1_DESCRIBE_STYLE
  `export GIT_PS1_DESCRIBE_STYLE=branch`
- Detached head tila on juuri missä alimoduulin kuuluukin olla, joten ei tehdä sille mitään vaan siirrytään seuraavaan osaan harjoituksessa

Repositorioiden ja alimoduulien synkronointi

- Tehdään hieman muutoksia repositorioihin, ja katsotaan miten päivitykset saa kaikille
- Muutetaan ensin kirjastoa, tämän voisi elävässä elämässä tehdä joku joka kirjaston koodin omistaa - esimerkiksi täysin meidän organisaatiomme ulkopuolelta
- Siirry kirjasto-repositorioon (eli juuren kirjasto hakemistoon)
- Lisää uusi tiedosto, esimerkiksi aikaleimalla (ja jos haluat edes vähän koodata, niin muuta vaikka noppa oletuksena 8-sivuiseksi)
  `date > koodia.not`
- Committoi muutos, esimerkiksi nimellä "Mukamuutos kirjastoon 1"
- Tehdään samaan tahtiin toinenkin muutos, lisätään toinen aikaleima tehtyyn uuteen tiedostoon ja tehdään commit
  date >> koodia.not
  git commit -am "Mukamuutos kirjastoon 2"
- Vie muutokset remoteen pushaamalla
- Nyt kirjaston hoitaja on valmis ja siirtyy muihin töihin, ohjelman kehittäjä taas jatkaa hommia ja haluaa tehdyt muutokset näkymään omassa alimoduulissaan
- Vaihda repositorioon ohjelma, ja hae muutokset. Tehdään se kuitenkin tässä fetchillä eikä pullilla, näin pääsemme tarkistelemaan tehtyjä committeja ennen mergeämistä
  cd kirjasto/noppa
  git fetch
  git log --oneline origin/master
- Nyt voisi tehdä mergen, mutta katsotaan miten muutoksia voisi tutkia
- Siirry checkoutilla haetun projektin viimeisimpään committiin sen commitidllä (joka näkyy ylemmän kohdan git log tulostuksessa ylimpänä).

Jos et halua nähdä valitusta detached head tilanteesta, niin käytä -q optiota
- Checkoutin jälkeen siirry ohjelman juureen (cd -), ja katso status. Jos asetit status.submodulesummary asetuksen aiemmin, niin statuksen pitäisi kertoa, että alimoduulissa on uusia committeja
- Nyt voit tehdä commitin, esimerkiksi viestillä "Ohjelman alimoduuli päivitetty"
- Vie nyt muutokset remoteen, eli `git push`

Seuraavaksi katsotaan miten meidän repositoriomme kloonannut muutokset omaan repositorioonsa meidän repositoriomme kautta
- Siirry siis "toisen kehittäjän" dev2 hakemistoon
- Tee normaali pull, git login pitäisi kertoa että olet saanut uusimman version. Tarkista vielä kirjastot/noppa tiedostojen sisältö ja katso git status
- **Alimoduulien sisältö ei tule git pullin mukana!** Eli jos teet commitin tässä tilanteessa, niin periaatteessa hävität kirjastoon tehdyt muutokset
- Eli muista **aina** hakemisen (fetch/pull) jälkeen myös päivittää alimoduulin muutokset. Tämä on yksi suuria ongelmia git submoduulien kanssa, liian helppo unohtaa
- Päivitys sinänsä on helppo:
```
git submodule update
```

Katsotaan vielä yksi potentiaalinen tilanne alimoduulien kanssa eli alimoduulin päivitys sitä käyttävän repositorion kautta - tässä muutos tehdään dev2 repositorion kautta
- Siirry dev2 repositorioon (tai pysy siellä)
- siirry alimoduuliin, eli `cd kirjastot/noppa`
- Varmista että olet ajan tasalla paikallisen repositorion kanssa
```
git checkout master
git pull --rebase
```
- Ylätasolla olisi voinut tehdä saman (älä siis tee tätä jos teit edellisen kohdan, on siis vaihtoehto):
```
git submodule update --remote --rebase -- kirjasto/noppa
```
- Nyt voit tehdä muutoksia, mennään taas kohtalaisen yksinkertaisella ja helpolla
```
echo Dev2 to the rescue >> koodia.not
git commit -am " Mukamuutos kirjastoon 3"
```
- Sitten committoidaan ylätason repositorio
```
cd ../..
```
git commit -am "Devin alimoduuli päivitetty"
- Seuraavaksi push, **mutta: pushaa molemmat, myös alimoduuli**. Tämä on erittäin helppo unohtaa ja pushata vain ylätason repositorio
- Leikitään kuitenkin että se unohtuisi, esimerkiksi graafisen työkalun kautta pushia tehden. Tee siis dev2 hakemistossa: `git push`
- Siirry seuraavaksi ohjelma repositorioon, katsotaan miltä tilanne näyttää kun alimoduuulia *ei* pushattu
- Koita ensin tehdä git pull, se epäonnistuu seuraavalla virheilmoituksella
```
Fetching submodule vendor/plugins/demo
error: Server does not allow request for unadvertised object 28f7
```
- Jos ohjelma-repositorion päivittäjä on osaava, hän voi koittaa korjata tilanteen
```
git submodule sync
git pull
```
- Tämä näyttääkin toimivan, paitsi kun katsot git statuksen
- Älä sotke tilannetta enempää, vaan mene takaisin dev2 repositorioon ja korjaa tilanne siellä, eli tee myös alimoduulille push
```
cd ../dev2/kirjastot/noppa
git push
```

- Nyt voitkin palata mainiin ja päivittää alimoduulin
```
cd -
git submodule update
```
- Tarkista vielä että kaikki on kunnossa