Pieni Git-opas

Sisältö

Pienen Git-oppaan tarkoitus on tarjota opiskelijoille helppokäyttöinen ohjekatalogi, jossa jokaiselle peruskäytön edellyttämälle ohjeelle on oma sivunsa. Ohjeen pohjana on Turun yliopiston Tulevaisuuden Teknologioiden laitoksen Pikaopas Git-versionhallintaan[1] joka tarjoaa laajemman ja syvemmän katsauksen Gitin perusteista. Lisälähteinä on käytetty mainiota Oh Shit, Git!?! -sivua[2], jossa esitellään täsmäohjeita erilaisiin ongelmiin, sekä Git Pro Book[3] -teosta.

- Tuisku Polvinen 11. maaliskuuta 2020

[1] https://gitlab.utu.fi/tech/education/git-opas/
[2] https://ohshitgit.com/
[3] https://git-scm.com/book/

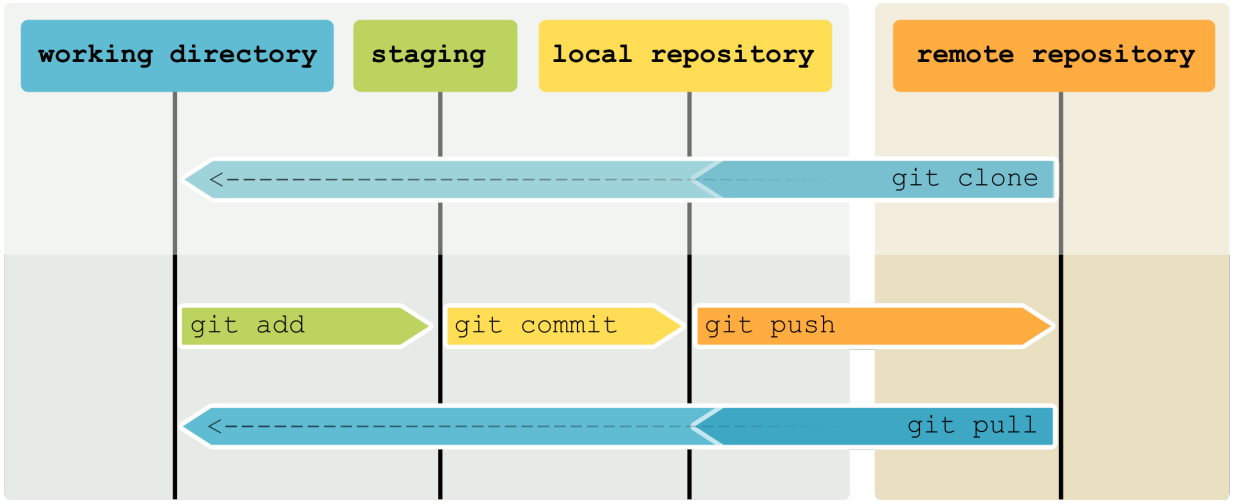
Luku 1

Gitin perusteet

Git on versionhallintajärjestelmä, jonka avulla voi pitää kirjaa projektin tiedostoista ja niiden muutoksista. Se mahdollistaa helpon paluun projektin aiempaan versioon, projektin työstämisen eri koneilta käsin ja useamman kuin yhden hengen yhtäaikaisen osallistumisen samaan projektiin.

Repositorio (repository, repo) eli tietovarasto on .git-kansio, joka sisältää tiedot kaikista projektin tiedostoista. Repositorio hankitaan joko luomalla se itse tai kloonaamalla valmis **etärepositorio** esim. GitLabista, jolloin käyttäjä saa itselleen kaikki projektiin sisältyvät tiedostot ja versionhallintatiedot.

Etärepositorio (remote repository) on palvelimella, esim. GitLabissa sijaitseva repositorio, jota useampi muokkaaja voi käsitellä klonaamalla siitä itselleen paikallisen repositorion, muokkaamalla tiedostoja omalla koneellaan, tallentamalla muutokset paikalliseen repositorioon ja lähettämällä paikalliset muutoksensa sen jälkeen etärepositorioon. Muut voivat tämän jälkeen hakea etärepositorioon lähetetyt muutokset omiin paikallisiin repositorioihinsa.



Gitin peruskäyttöön tarvitset käytännössä seuraavat komennot:

- **git clone** kloonataan etärepositorio (remote repository) omalle koneelle. Kloonaaminen lataa myös kaikki repositorion seuraamat tiedostot paikalliseen työskentelyhakemistoon (working directory).
- **git add** listätään työskentelyhakemistossa tehdyt muutokset valmistelualueelle eli indeksiin (stage).
- **git commit** sidotaan valmistelualueelle lisätyt muutokset pysyväksi muutokseksi eli solmuksi (commit) paikalliseen repositorioon.
- **git push** lähetetään pysyvät muutokset paikallisesta repositoriosta etärepositorioon.
- **git pull** haetaan etärepositoriosta muutokset omaan repositorioon ja työskentelyhakemistoon.

Ryhmätyöskentelyssä tai ongelmia kohdatessa tarvitset muitakin komentoja, joista lisää luvuissa 2 ja 3.

Gitin termeistä ei ole vakiintuneita suomennoksia, ja usein selkeintä on käyttää englanninkielistä termistöä.

1.1 Gitin asentaminen ja konfigurointi

1.1.1 Asennus

Windows

1. Lataa asennustiedosto `https://gitforwindows.org/` (listasta todennäköisesti "Git-2.25.1-64-bit.exe")
2. Suorita asennusohjelma. Kaikessa muussa voit jättää oletusvalinnat, mutta valittaessa Gitin käyttämää oletuseditoria asennusohjelman oletus on Vim, joka voi olla parempi vaihtaa yksinkertaisempaan editoriin. Nano editor on hyvä vaihtoehto.

Git on mahdollista asentaa myös Windows Subsystem for Linuxiin[1].

macOS

1. Avaa pääte (terminal) ja tarkista, onko koneellasi jo Git komennolla `git --version`
2. Jos Git ei ole vielä asennettuna, aukeaa ikkuna, jossa kysytään haluatko asentaa XCode developer tools -työkalupaketin. Voit vastata kyllä, jolloin paketti asennetaan automaattisesti, tai asentaa itse uusimman version asennusohjelmalla osoitteesta https://git-scm.com/download/mac

Linux

1. Avaa terminaali (Ctrl+t)
2. Kirjoita distribuutioosi sopiva oikeanpuolisen listan mukainen komento.
3. Tarkista että git asentui komennolla `git --version`

Linux-distribuutioita & Git-asennuskomentoja

Ubuntu/Debian	`apt-get install git`
Arch Linux	`pacman -S git`
Fedora v. $\leq$18	`dnf install git`
Fedora v. $\geq$17	`yum install git`
openSUSE	`zypper install git`
CentOS	`yum install git`
Gentoo	`emerge --ask --verbose dev-vcs/git`
Nix/NixOS	`nix-env -i git`

1.1.2 Käyttöönotto ja asetukset

Jos käytät Windowsia, käynnistä Git Bash. Linuxilla ja MacOSilla voit käyttää Gitiä suoraan terminaalista. Windowsin Git Bashissa git-komennot toimivat täysin samalla tavalla kuin muissa järjestelmissä.

Aseta käyttäjätietosi `git config` -komennoilla:

```
$ git config --global user.name "Etunimi sukunimi"
$ git config --global user.email "tunnus@utu.fi"
```

Nyt olet valmis käyttämään Gitiä. Asetuksia voi muokata myös ~/.gitconfig -tiedostossa.

Windows-käyttäjille tiedoksi

Git Bash sisältää monenlaisia Unix-tyyppisistä järjestelmistä tuttuja ominaisuuksia, joita voit käyttää sen komentorivin kautta. Tässä muutama käyttökelpoinen komento:

`cd <määränpää>` hakemistojärjestelmässä liikkuminen, `cd ~` siirtää sinut kotihakemistoosi.

`pwd` näyttää hakemistopolun siihen kansioon jossa olet

`ls` listaa kansion sisällön

`less <tiedosto>` avaa tiedoston luettavaksi (poistu painamalla q-näppäintä)

`mkdir <uusi kansio>` luo uuden kansion senhetkiseen sijaintiisi

`nano <tiedosto>` avaa tiedoston nano-tekstieditorissa (jos tiedostoa ei vielä ollut, se luodaan samalla)

[1] https://docs.microsoft.com/en-us/windows/wsl/install-win10

1.2 SSH-avainten luominen ja käyttö

Avainta käytetään GitLab-tunnistautumiseen, jolloin erillistä kirjautumista ei tarvita.

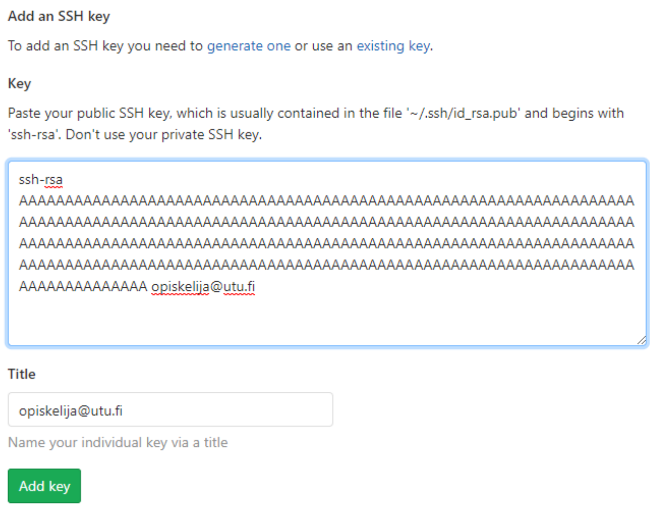
1.2.1 Avaimen luonti

Yliopiston virtuaalikoneissa avain on luotu automaattisesti kirjautumisvaiheessa antamillasi syötteillä ja voit siirtyä suoraan seuraavaan vaiheeseen.

1. Avaa Git Bash (windows) tai terminaali/pääte (linux & mac)
2. luo avain komennolla `ssh-keygen -t rsa -b 4096 -C ''<sahkopostiosoitteesi>''` (esim. `ssh-keygen -t rsa -b 4096 -C ''opiskelija@utu.fi''`)
3. Paina enter hyväksyäksesi avainten oletussijainnin, sitä ei ole tarvetta muuttaa.
4. Git kehottaa luomaan avainfraasin, joka toimii jatkossa salasanan kaltaisesti otettaessa yhteyttä etärepositorioon. Avainfraasin on hyvä olla helposti muistettava. Avainfraasin syöttämisen jälkeen Git pyytää varmistukseksi syöttämään sen uudelleen.
5. Onnistuneen avainten luonnin jälkeen Git tulostaa avainten yleistiedot.

1.2.2 Avaimen liittäminen GitLabiin

1. Näytä avain komennolla `cat ~/.ssh/id_rsa.pub` ja kopioi se leikepöydälle
2. Liitä avaimesi sivulta https://gitlab.utu.fi/profile/keys löytyvään kenttään ja paina 'Add key'
3. Avaimesi pitäisi nyt näkyä 'Your SSH keys' -listassa.



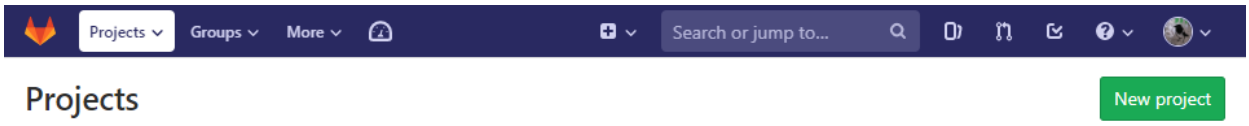
1.2.3 Esimerkki

```
$ ssh-keygen -t rsa -b 4096 -C "opiskelija@utu.fi"
Generating public/private rsa key pair
Enter file in which to save the key (/c/Users/user/.ssh/id_rsa):
>  Hyväksytään oletussijainti painamalla enter
Enter passphrase (empty for no passphrase):
>  Syötetään avainfraasi
Enter same passphrase again:
>  Syötetään avainfraasi uudestaan
Your identification has been saved in /c/Users/user/.ssh/id_rsa.
Your public key has been saved in /c/Users/user/.ssh/id_rsa.pub.
< avaimen sormenjälki >
< avaimen randomart-kuva >
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA opiskelija@utu.fi
```

1.3 Uuden repositorion luominen (git init)

1.3.1 Uusi etärepositorio GitLabissa

Voit luoda uuden projektin sivulla `https://gitlab.utu.fi/projects/new` jonne pääset myös GitLabin etusivulta.



1. Anna projektillesi nimi ja valitse sijainti. Oletuksena se luodaan oman käyttäjätilisi alle, mutta jos kuulut GitLabissa johonkin ryhmään, voit luoda projektin myös ryhmän alle.
2. Valitse haluamasi näkyvyysasetus. Kurssien harjoitustöissä on sopivaa valita näkyvyydeksi "Private".
   Jos haluat tyhjän etärepositorion johon voit lähettää koneellasi jo olevan repositorion, jätä "Initialize repository with a README" -kohta valitsematta.
3. Paina "Create project"ja pääset uuden projektisi sivuille, jossa näet tyhjän repositoriosi.
4. Seuraava askel on äsken luodun repositorion kloonaaminen omalle koneellesi tai jonkin paikallisen repositorion tuominen luomaasi etärepositorioon.
   Lue GitLabin tarjoamat ohjeet tai katso sivu "Valmiin repositorion kloonaaminen"

1.3.2 Uusi repositorio paikallisesti

1. Siirry hakemistoon, jonne haluat luoda repositoriosi Git Bashia (windows) tai terminaalia (linux, macOS) käyttäen.
   - `pwd` (present working directory) tulostaa hakemiston jossa olet.
   - `cd <hakemisto>` siirtää sinut haluamaasi hakemistoon (esim. `cd ~/projektit`).
2. `git init <uuden repositoriosi nimi>` luo hakemiston, jonka sisällä on uuden repositoriosi .git-tiedosto (esim. `git init harkkatyo`).
   - Pelkkä `git init` luo repositorion suoraan siihen hakemistoon jossa olet luomatta sille omaa hakemistoa. Käytä tätä jos olet jo luonut projektille sopivan hakemiston.
3. `cd <uuden repositoriosi nimi>` siirtää sinut repositoriosi hakemistoon (esim. `cd harkkatyo`).

1.3.3 Etärepositorion yhdistäminen paikalliseen repositorioon

Jos olet luonut uuden paikallisen repositorion, jolle ei ole vielä asetettu etärepositorioita, voit lisätä etärepositorion komennolla `git remote add origin <etarepositorion osoite>`
(esim. `git remote add origin gitlab@gitlab.utu.fi:opiskelija/harkkatyo.git`)

Esimerkki

Luodaan projektit-kansioon uusi projekti "git-opas"repositorioineen.

- Käyttäjän antamat komennot näkyvät merkin $ jälkeen ja komentojen selitykset alkavat merkillä #

```
$ cd ~/projektit # Siirrytään kotihakemiston alikansioon "projektit"
$ git init git-opas # Luodaan projekti
Initialized empty Git repository in C:/Users/user/projektit/git-opas/.git/
$ cd git-opas # Siirrytään projektikansioon
```

1.4 Valmiin repositorion kloonaaminen GitLabista (git clone)

1.4.1 Valmistelut

1. Siirry Git Bashia (windows) tai terminaalia (linux, macOS) käyttäen hakemistoon, jonne haluat kloonatun projektin repositorioineen.
   - `pwd` (present working directory) tulostaa hakemiston jossa olet
   - `cd <hakemisto>` (change directory) avaa halutun hakemiston (esim. `cd ~/projektit`)

1.4.2 Kloonaaminen SSH-linkillä

1. `git clone <git-tiedoston ssh-linkki>` kloonaa haluamasi repositorion (esim. `git clone gitlab@gitlab.utu.fi:opiskelija/harkkatyo.git`).
   - Jos projektin nimessä on välilyöntejä, merkitään ne SSH-linkissä yhdysmerkillä (-).
   - GitLabista löydät projektin sivulta Clone-valikon, josta saat kopioitua kloonaukseen tarvittavan linkin.
2. syötä SSH-avainfraasisi Gitin tulostettua `Enter passphrase for key '/c/Users/user/.ssh/id_rsa':`
3. Git kopioi etärepositoriosta kaikki tiedot sekä luo master-haaran uusimmasta versiosta työkopion, jonka tiedostot ovat projektin hakemistossa nähtävissäsi ja muokattavissasi.

1.4.3 Lopuksi

`cd <kloonatun repositorion nimi>` siirtää sinut hakemistoon jossa juuri kloonattu repositorio on (esim. `cd harkkatyo`).

Esimerkki

Kloonataan GitLabista pienen git-oppaan projekti repositorioineen käyttäjän kansioon "projektit".

- Käyttäjän antamat komennot näkyvät merkin $ jälkeen ja komentojen selitykset alkavat merkillä #.
- Gitin tulostamat rivit ovat harmaita. Huomiota vaativat välikommentit alkavat merkillä >.

```
$ cd ~/projektit # Siirrytään kansioon jonne projekti halutaan kloonata
$ git clone gitlab@gitlab.utu.fi:tuisku/pieni-git-opas.git # Kloonataan etärepositorio
Cloning into 'pieni-git-opas'...
Enter passphrase for key '/c/Users/user/.ssh/id_rsa':
>   tässä kohtaa Git pyytää avainfraasia ja jatkaa vasta kun se on syötetty.
remote: Enumerating objects: 95, done.
remote: Counting objects: 100% (95/95), done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 95 (delta 41), reused 92 (delta 39)
Receiving objects: 100% (95/95), 1.50 MiB | 11.32 MiB/s, done.
Resolving deltas: 100% (41/41), done.
$ cd git-opas # Siirrytään projektikansioon
```

1.5 Uuden etärepositorion asettaminen (git remote add)

Etärepostitorion luominen aivan uudelle projektille on jo käsitelty osiossa 1.3.3, mutta jos olet kloonannut projektin esimerkiksi kurssinpitäjän antamasta pohjasta, haluat todennäköisesti päivittää tekemäsi muutokset omaan projektiisi etkä opettajan versioon. Tätä varten sinun on asetettava projektiin uusi etärepositorio.

1.5.1 Valmistelut

1. Luo oma, tyhjä etärepositorio GitLabiin kuten osiossa 1.3.1.
2. Varmista, että olet paikallisen repositoriosi hakemistossa komennolla `pwd`. Siirry tarvittaessa oikeaan hakemistoon komennolla `cd <hakemisto>`.

1.5.2 Aiemman etärepositorion uudelleennimeäminen

Kloonaamallasi projektilla on etärepositorio nimeltä *origin*, jonka osoite on kloonatun projektin alkuperäinen osoite. Näet etärepositoriot komennolla `git remote`.

1. Nimeä alkuperäinen etärepositorio uudelleen komennolla `git remote rename origin old-origin`.
2. Näet muutoksen komentamalla uudelleen `git remote`.

1.5.3 Uuden etärepositorion asettaminen

Kun vanha etärepositorio on nimetty uudelleen, voit asettaa oman etärepositoriosi origin-tunnukselle.

1. Aseta uusi etärepositorio komennolla `git remote add origin <uuden etarepositorion osoite>` (esim. `git remote add origin gitlab@gitlab.utu.fi:opiskelija/harkkatyo.git`).
2. Työnnä paikallisen repositorion sisältö uuteen origin-etärepositorioosi komennolla `git push -u origin --all`.
3. Nyt komennolla `git remote` pitäisi näkyä kaksi etärepositorion tunnusta, *origin* ja *old-origin*.

Esimerkki

Kloonataan GitLabista pienen git-oppaan projekti repositorioineen käyttäjän kansioon "projektit"ja asetetaan sen uudeksi etärepositorioksi opiskelijan luoma oma-git-oppaani.git

- Käyttäjän antamat komennot näkyvät merkin $ jälkeen ja komentojen selitykset alkavat merkillä #.

```
$ cd ~/projektit # Siirrytään kansioon jonne projekti halutaan kloonata
$ git clone gitlab@gitlab.utu.fi:tuisku/pieni-git-opas.git # Kloonataan etärepositorio
Cloning into 'pieni-git-opas'...
Enter passphrase for key '/c/Users/user/.ssh/id_rsa':
>   tässä kohtaa Git pyytää avainfraasia ja jatkaa kloonaamista kun se on syötetty.
$ cd git-opas # Siirrytään projektikansioon
$ git remote rename origin old-origin # Nimetään alkuperäinen origin old-originiksi
$ git remote add origin gitlab@gitlab.utu.fi:opiskelija/oma-git-oppaani.git
>   # Asetetaan uusi origin-etärepositorio
$ git push -u origin --all # Työnnetään sisältö uuteen etärepositorioon
```

1.6 Repositorion uusimman version hakeminen (git pull)

Jos etärepostioriota muokataan useammalta taholta, on vähän väliä päivitettävä oma paikallinen repositorio ajan tasalle hakemalla etärepositorion uusin versio. Tätä ei voi tehdä liian usein. Esimerkiksi omien muutosten lähettäminen muuttuneeseen etärepositorioon aiheuttaa konflikteja, sillä ei ole mahdollista automaattisesti tietää, mitkä muutokset ovat oikeellisempia.

1. Tarkista että olet oikeassa hakemistossa. Komento `pwd` tulostaa sijaintisi.
    - Voit tarvittaessa siirtyä hakemistoissa komennolla `cd <hakemistopolku>`
2. Hae etärepositorion uusin versio komennolla `git pull`.
    - `Already up to date` tarkoittaa, että hakemistosi oli jo ajan tasalla.
    - Jos hakemistosi ei ollut ajan tasalla, hakee Git muutokset repositoriosta ja päivittää paikallisen hakemiston.

1.6.1 Lisätietoa: git pull = git fetch + git merge

Komento `git pull` on käytännössä yhdistelmä kahdesta komennosta – `git fetch` joka hakee etärepositorion tiedot ja `git merge` joka varsinaisesti liittää haetut tiedot omiisi. Tätä yhdistelmää käyttämällä voit välttää pulmatilanteita, sillä voit vertailla haettuja muutoksia ennen liittämistä konfliktien havaitsemiseksi.

Esimerkki

```
$ git pull # Vedetään repositorion uusin versio
Enter passphrase for key '/c/Users/user/.ssh/id_rsa':
>   Syötetään avainfraasi
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From gitlab.utu.fi:tuisku/pieni-git-opas
   de1655c..93520b1  master     -> origin/master
Updating de1655c..93520b1
Fast-forward
readme.md | 3 ++-
1 file changed, 2 insertions(+), 1 deletion(-)
>   Repositoriossa oli yksi muuttunut tiedosto, jonka muutokset haettiin.
```

Kokeillaan uudestaan:

```
$ git pull # Vedetään repositorion uusin versio
Enter passphrase for key '/c/Users/user/.ssh/id_rsa':
>   Syötetään avainfraasi
Already up to date
>   Työhakemisto oli nyt ajan tasalla.
```

1.7 Omien muokkausten lähettäminen etärepositorioon (git push)

Kun työkopion tiedostoja on muokattu ja halutaan lähettää muokatut tiedostot GitLabiin, tehdään se luomalla ensin omaan paikalliseen repositorioon Git Bashin (windows) tai terminaalin kautta ns. pysyvä muutos eli sitouma tai solmu (commit), joka lähetetään (push) etärepositorioon. Jokainen solmu on kuin tallennuspiste, johon voi automaattisesti generoidun tunnisteen avulla halutessaan palata myöhemmin.

1.7.1 Valmistelut

1. Tarkista että olet oikeassa hakemistossa. Komento `pwd` (present working directory) tulostaa sijaintisi.
   - Voit tarvittaessa siirtyä hakemistoissa komennolla `cd <hakemistopolku>`
2. `git status` näyttää viimeisimmän pysyvän muutoksen (commit) jälkeen tehdyt muutokset.
   - Muokkaamasi ja luomasi tiedostot näkyvät punaisella, jos niitä ei ole vielä lisätty indeksiin (stage). Jos et näe tiedostojasi ja git sanoo `nothing to commit, working tree clean`, tarkista, että olet varmasti muistanut tallentaa tiedostoissa tekemäsi muutokset.
3. Jos etärepositoriota käsitellään useammalta koneelta, saattaa useampi henkilö olla tehnyt ja lähettänyt muokkauksia samanaikaisesti. Hae repositorion senhetkinen tila itsellesi komennolla `git pull` konfliktien välttämiseksi.

1.7.2 Muutosten tallentaminen paikalliseen repositorioon (add, commit)

1. `git add` lisää työkopiosi tiedostot indeksiin (stage).

   `git add --all` tai `git add .` lisää kaikki tiedostot
   - Rajataksesi osan tiedostoista vakituisesti all-komennon ulkopuolelle luo .gitignore

   `git add <tiedostonimi>` lisää yksittäisen tiedoston

   `git add <tiedosto1> <tiedosto2> <tiedosto3>` lisää useamman tiedoston kerralla

   `git add <hakemisto>` lisää hakemiston Voit käyttää tabulaattorinäppäintä täydentääksesi automaattisesti tiedostopolut ja -nimet kirjoittaessasi.

   ○ Nyt `git status` näyttää indeksiin lisäämäsi tiedostot vihreänä.
2. `git commit -m ''<viesti>''` sido indeksiin merkityt tiedostot uuteen solmuun (commit).
   - Viestistä olisi hyvä käydä ilmi, mitä muutoksia olet tehnyt. Ääkkösiä on hyvä välttää.

1.7.3 Viimeisimmän commitin lähettäminen etärepositorioon (push)

1. Ennen lähettämistä on aina hyvä hakea etärepositorion mahdolliset muutokset komennolla `git pull`.
2. `git push` lähettää (push) paikalliseen repositorioon luomasi solmun (commit) etärepositorioon.
   - Jos kaikki sujui hyvin, Git tulostaa muutaman rivin tietoa, viimeisenä lähetettyjen tiedostojen määrän.

Esimerkki

Tiedostoa pieni_git_opas.pdf on muokattu. Uusi versio halutaan lähettää etärepositorioon.

```
$ cd ~/projektit/pieni-git-opas # Siirrytään projektin kansioon
$ git status # Tarkistetaan näkyykö muutettu tiedosto
$ git add pieni_git_opas.pdf # Kaikki ok, lisätään tiedosto valmistelualueelle
$ git pull # Vedetään repositorion uusin versio
$ git commit -m "lisatty kappale gitin push-ohjeesta" # Tehdään pysyvä muutos
$ git push # Pusketaan muutos repositorioon
```

1.8 Muita hyödyllisiä komentoja

`git help <komento tai toiminto>` avaa Gitin omia ohjeita. Kokeile myös `git help tutorial`.

`git diff <tiedosto>` näyttää miten työkopion tiedosto eroaa sen indeksissä olevasta versiosta. Nähdäksesi komennon muita käyttötarkoituksia, kokeile `git help diff`.

`git reflog` listaa commit-historian.

`git log` näyttää muokkaushistorian.

`git whatchanged` näyttää mitä tiedostoja on muutettu.

`git rm <tiedosto>` poistaa yksittäisen tiedoston sekä työskentelyhakemistosta että indeksistä.

`git mv <vanha nimi> <uusi nimi>` muuttaa tiedoston nimen.

`git branch <uuden haaran nimi>` luo uuden haaran, jonka sisältö on sama kuin senhetkisen haaran. Pelkkä `git branch` näyttää projektin haarat.

`git checkout <haara>` siirtyy annettuun haaraan sekä muuttaa työkopion sen mukaiseksi.

`git checkout <tiedosto>` korvaa työkopiossa olevan tiedoston sen indeksissä olevalla versiolla. Checkoutilla on kahden esitellyn lisäksi monia muitakin käyttötarkoituksia. Lisätietoa saat komennolla `git help checkout`.

`git fetch` hakee etärepositorion tilan muuttamatta työkopion tiedostoja.

`git merge` yhdistää kaksi rinnakkaista tilaa. Yhdistä fetchillä hakemasi tila omaasi komennolla `git merge origin/master`. Yhdistä toinen haara senhetkiseen haaraan komennolla `git merge <toinen haara>`

`git stash` siirtää indeksissä olevat tiedostot jemmaan (stash) eli syrjään työkopiosta, jolloin ne ovat turvassa esimerkiksi haarojen välissä liikuttaessa. Käytä `git stash -u` jos haluat jemmata myös ne työkopion tiedostot joita ei ole lisätty indeksiin.

`git stash pop` siirää jemmassa olevat tiedostot senhetkiseen työkopioon.

`git rebase` järjestelee uudelleen jo solmittuja muutoksia, tällä voi epähuomiossa tehdä paljon tuhoa.

`git revert` luo uuden commitin, joka peruu edeltäjänsä.

1.9 Hyödyllisiä tiedostoja

.gitignore

Jos haluat Gitin jättävän huomiotta tietyt tiedostot ja hakemistot, voit luoda .gitignore-tiedoston repositorion juurihakemistoon. Luo se esimerkiksi komennolla `> .gitignore`. Windows-koneella voit myös luoda uuden tekstitiedoston, jota luodessa korvaat nimen "New Text Document.txt" kokonaisuudessaan nimellä ".gitignore." (huomaa piste lopussa), jolloin .txt-loppu jää pois.

Kirjoita .gitignore-tiedostoon huomiotta jätettävät tiedostot omille riveilleen. Voit käyttää asteriskia ja sivuuttaa esimerkiksi kaikki .class-tiedostot rivillä `*.class`.

README.md

Readme-tiedosto näkyy GitLabissa projektisi etusivulla. Siihen on hyvä kirjoittaa esimerkiksi kuvaus projektista, projektiin osallistuvien ryhmäläisten nimet kurssityön tarkistamista varten, ohjeita projektin käyttöönottoon tai muuta hyödyllistä.

Luku 2

Muutosten peruminen

2.1 Indeksiin lisätyn tiedoston poistaminen ennen commitia

Indeksiin lisättyjä yksittäisiä tiedostoja saa poistettua komennolla `git reset <tiedosto>`. Komento ei vaikuta työkopiossa olevaan tiedostoon, vaan resetoi sen osalta indeksiin viime commitin jälkeen lisätyt muutokset.

2.2 Paikallisen commitin muokkaus (git amend)

Komennolla `git amend` korvataan edellinen commit senhetkisellä indeksin tilalla.

Unohtuneen tiedoston lisäys jo tehtyyn commitiin

Jos huomaat unohtaneesi lisätä jonkin tiedoston indeksiin ennen sen sitomista uudeksi commitiksi tai joudut vaikka korjaamaan tiedostosta pienen kirjotusvirheen, on commitiin mahdollista lisätä se vielä heti sitomisen jälkeen. Vipu `--no-edit` tarkoittaa sitä, että commitin viesti pysyy samana.

```
$ git add <tiedosto>
$ git commit --amend --no-edit
```

Jos edelliseen commit-viestiisi on tullut vaikka kirjoitusvirhe, voit amendilla korvata myös pelkän viestin.

```
$ git commit --amend -m "uusi viesti"
```

Paikallisen commitin peruminen

Ennen etärepositorioon lähettämistä voit vielä purkaa sitomasi commitin soft tai hard resetillä. Soft reset purkaa viimeisimmän commitin niin, että muutokset jäävät vielä indeksiin ja työkopioon. Hard reset purkaa commitin ja poistaa indeksistä sekä työkopiosta kaikki viime commitin jälkeiset muutokset. Huom! hard resetiä ei voi perua etkä saa menetettyjä tiedostojasi takaisin.

```
$ git reset HEAD~ --soft
```

```
$ git reset HEAD~ --hard
```

2.3 Kaikkien paikallisten muutosten peruminen

2.3.1 Paikallisen tilan muuttaminen täysin etärepositoriota vastaavaksi

Huom! Näin hävitettyjä paikallisia tiedostoja et välttämättä koskaan saa takaisin eikä tätä voi perua.

```
$ git fetch origin
$ git checkout master
$ git reset --hard origin/master
$ git clean -d --force
```

Luku 3

Git ja ryhmätyöskentely

Git on hyödyllinen työväline jo yhden hengen projektissa, mutta useamman henkilön työstäessä samaa projektia toimivan versionhallintajärjestelmän ja sen sujuvan käytön merkitys korostuu. Eri ihmisillä voi olla hyvinkin erilaisia tapoja ylläpitää repositorioitaan, mistä voi seurata monenlaisia konflikteja. Ongelmien välttämiseksi on hyvä sopia yhtenäinen työnkulku eli **workflow**, jota kaikki noudattavat.

Yksin työskennellessä moni on työnkulkutyyppiä kummemmin miettimättä käyttänyt keskitetyksi työnkuluksi (centralized workflow) kutsuttua tapaa, jossa on yksi keskushaara, master, johon kaikki muutokset lähetetään suoraan. Jos tekijöitä on yksi, ongelmia ei välttämättä synny, mutta useamman tehdessä muutoksia suoraan master-haaraan voi konflikteja ja muita ongelmia syntyä harmillisen paljon.

Yleisesti käytössä olevia työnkulkutyyppejä on useita ja jokaisella voi olla oma suosikkinsa, mutta haaroja hyödyntävä **feature branch** -workflow sopii monenlaiseen ryhmätyöskentelyyn.

3.1 Työnkulku: feature branch

Feature branch-työnkulussa kehitystä ei koskaan tehdä suoraan master-haaraan, vaan varsinainen kehitys tapahtuu omissa ominaisuushaaroissaan. Kun kehitettävä ominaisuus tai osa on valmis, se liitetään master-haaraan. Tarkoitus on, että master-haarassa on aina viimeisin toimiva versio, ja ominaisuushaaroissa projektin jäsenet voivat muokata haaraa vapaasti ja tehdä kokeellisiakin muutoksia ilman huolta koko projektin hajottamisesta.

Ominaisuushaaroja voi puskea etärepositorioon siinä missä master-haaraakin, jolloin myös niiden tila ja historia on muidenkin projektin jäsenten saavutettavissa ja hekin voivat tehdä haaroihin muutoksia.

Yksinkertaisimmillaan feature branch -työnkulku toimii näin:

1. Kloonataan projekti:
   ```
   $ git clone git@example.com:project-name.git
   ```
2. Luodaan uusi haara:
   ```
   $ git checkout -b haara
   ```
3. Kehitetään ominaisuutta, minkä jälkeen tiedostot lisätään indeksiin kuten tavallisesti
   ```
   $ git add .
   ```
4. Sidotaan muutokset uuteen solmuun:
   ```
   $ git commit -m "Uusi ominaisuus"
   ```
5. Pusketaan haaran solmut etärepositorioon:
   ```
   $ git push origin haara
   ```
6. Nyt voit jatkaa haaran muokkaamista palaten taas kohtaan 3, tai liittää sen master-haaraan liitospyynnöllä (merge request) tai paikallisesti komentorivillä.

Seuraavassa kappaleessa käydään haaroittamista läpi hieman tarkemmin.

3.2 Haarojen käsittely

Haaran luonti (git branch)

Uusi haara luodaan komennolla `git branch <haaran nimi>`. Haaran nimi voi kuvata ominaisuutta jota siinä kehitetään, jolloin on helppo hahmottaa missä tehdään mitäkin. Uusi haara on käytännössä kopio siitä haarasta jossa olet antaessasi komennon. Muokataksesi haaraa sinun on siirryttävä siihen.

Haarojen käsittely ja niiden välillä siirtyminen (git checkout)

Git pitää kirjaa kaikista kaikissa haaroissa olevista tiedostoista, mutta pitää työkopiossasi näkyvillä tiedostoista aktiivisen haaran mukaiset versiot. Muutokset joita sidot liittyvät aina siihen haaraan, jossa sillä hetkellä olet. Haarojen välillä voi liikkua komennolla `git checkout <kohdehaara>`.
Jos huomaat olleesi väärässä haarassa tehdessäsi työkopioon muutoksia ja haluat siirtää ne toiseen haaraan, voit jemmata ne stash-komennolla, jolloin ne eivät jää senhetkiseen haaraan, ja palauttaa ne jemmasta haluamassasi haarassa seuraavasti:

```
$ git add .
$ git stash
$ git checkout oikea_haara
$ git stash pop
```

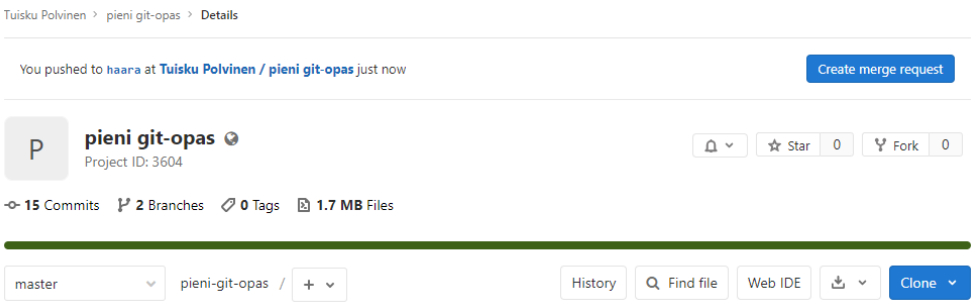
Jos ehdit jo sitoa muutokset commitiin, voit perua sen soft resetillä jonka jälkeen ylläoleva jemmaus onnistuu.

```
$ git reset HEAD~ --soft
```

Haarassa ollessasi voit muokata tiedostoja, solmia muutoksia ja puskea etärepositorioon kuten tavallisesti. Muutokset eivät näy muissa haaroissa ennen kuin ne liitetään niihin.

Liitospyyntöjen tekeminen GitLabissa (merge request)

Kun haara pusketaan etärepositorioon GitLabiin, näkyy projektin sivulla ehdotus liitospyynnön tekemisestä. Liitospyyntöjä voi tehdä myös branches-sivulta, jossa näkyvät kaikki haarat.



Kuva 3.1: Projektin sivu GitLabissa. Oikeassa yläkulmassa näkyy ehdotus haara-nimisen haaran liittämispyynnön tekemisestä. Kaikki haarat löytyvät projektin nimen alla näkyvästä *Branches*-linkistä.

Liitospyyntöehdotus näkyy aina kun haara on puskettu etärepositorioon, mutta pyyntö kannattaa tehdä vasta kun se on valmis masteriin liitettäväksi.

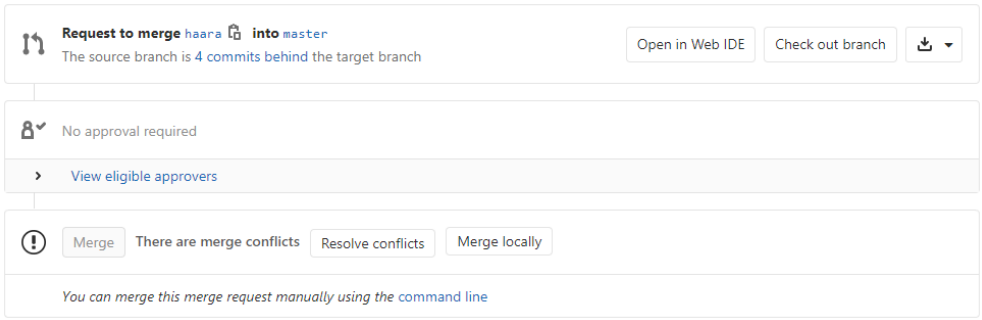
Merge request -sivulla voi pyynnölle antaa otsikon ja kuvauksen. Valintoja, asetuksia ja vaihtoehtoja on muitakin, ja sivulla voi määritellä esimerkiksi liitospyynnön hyväksymiskriteereitä. Suuremmissa projekteissa haara- ja liitospyyntöjärjestelmä voi olla paljonkin tavallista kurssityötä monimutkaisempi, eikä kaikilla ole esimerkiksi oikeuksia itse liittää haaroja masteriin tai edes hyväksyä liitospyyntöjä.

Liitospyyntöä voi ennen sen hyväksymistä ja liittämistä vielä muokata puskemalla muutoksia siihen haaraan, josta pyyntö on tehty.

Liitospyyntöjen hyväksyminen GitLabissa

Liitospyynnöt löytyvät GitLabissa projektinäkymän sivupaneelista. Liitospyynnön hyväksyntäsivu aukeaa myös pyynnon luonnin jälkeen, mutta on hyvä pyytää muita projektin jäseniä tarkistamaan se ennen liittämistä.

Jos master-haaraan ei ole liitetty tai tehty muita muutoksia haaran luonnin jälkeen tai ne ovat keskenään yhteensopivia, hoituu haaran liittäminen *Merge*-painikkeella. GitLab luo merge commitin ja käyttäjä voi halutessaan poistaa liitetyn haaran. Kehitystä voi myös jatkaa samassa haarassa ja sen voi liittää myöhemmin muutoksineen uudestaan masteriin.



Kuva 3.2: Tätä haaraa ei voi liittää masteriin sellaisenaan, vaan on selvitettävä merge-konflikti.

Jos masterissa ja siihen liitettävässä haarassa on konfliktiin johtavia eroja, voi niitä korjata suoraan GitLabissa *Resolve conflicts* -valinnalla tai paikallisesti komentorivillä.

Haarojen liittäminen paikallisesti (git merge)

Vetopyyntöjen ja liitosten tekeminen master-haaraan ainoastaan GitLabissa varmistaa, ettei masteriin tehdä eri tahoilla risiriitaisia muutoksia, mutta liittämisen voi tehdä myös paikallisesti, mikä on yhden hengen projektissa näppärämpää. Kun haluat liittää ominaisuushaarassa kehittämäsi muokkaukset masteriin, siirry ensin master-haaraan ja liitä haluamasi haara merge-komennolla:

```
$ git checkout master
$ git merge liitettava_haara
```

Käsittele mahdolliset merge-konfliktit ja puske master-haara etärepositorioon.

3.3 GitLabin issues-ominaisuuden käyttö

GitLab tarjoaa projektinhallintaan hyödyllisiä toimintoja. Projektinäkymän vasemmasta sivupaneelista löytyy GitLabin *issues*-työkalu, jolla voi nostaa esiin ja pitää kirjaa projektin tarpeista, epäkohdista, tavoitteista ja muista pulmista. Pulmia voi määrittää projektin tiettyjen jäsenten vastuulle, niitä voi kommentoida, niille voi asettaa aikarajoja ja etikettejä. Kaiken kaikkiaan projektien kasvaessa issues-ominaisuus helpottaa ja selkeyttää töiden jäsentelyä.

Kurssityötä aloittaessa issues-työkaluun voi esimerkiksi merkitä kaikki projektin deadlinet ja tehdä työn vaiheista alustavat issue-merkinnät. Niitä voi luoda lisää ja merkitä tehdyiksi sitä mukaa kun työ etenee.

Merkintöjen kommentointi antaa mahdollisuuden suunnitella projektin osien toteutusta etukäteen ja jakaa ideoita jo ennen yhdenkään koodirivin kirjoittamista. Bugien ylös kirjaaminen hoituu myös issue-merkinnöillä, ja niiden kuten muidenkin merkintöjen kohdalla merkinnän tekijä voi kirjoittaa huomaamansa epäkohdan ja joku muu projektin jäsen voi tarttua siihen ja ottaa hoitaakseen korjauksen tai toteutuksen.