

Git ja versionhallinta

Eetu Järvinen, Tuomas Ukkola, Viljami Ketola

Lyhyesti versionhallinnasta

Paikalliset versionhallintajärjestelmät

- Yksinkertainen paikallinen tietokanta, joka seuraa tiedostojen muutoksia
- Yksi suosituimmista oli RCS (Revision Control Systems)
- RCS toimii patch seteillä

Keskitetyt versionhallintajärjestelmät

- Kehittäjille mahdollisuus työskennellä yhdessä
- CVS, Subversion, Perforce
- Yksi “keskus” palvelin, jossa projekti sijaitsee kokonaisuudessaan
- Pitkäaikainen versionhallinnan standardi
- Mutta haittoja esim. riski pitää koko projektin historiaa vain yhdessä paikassa

Hajautetut versionhallintajärjestelmät

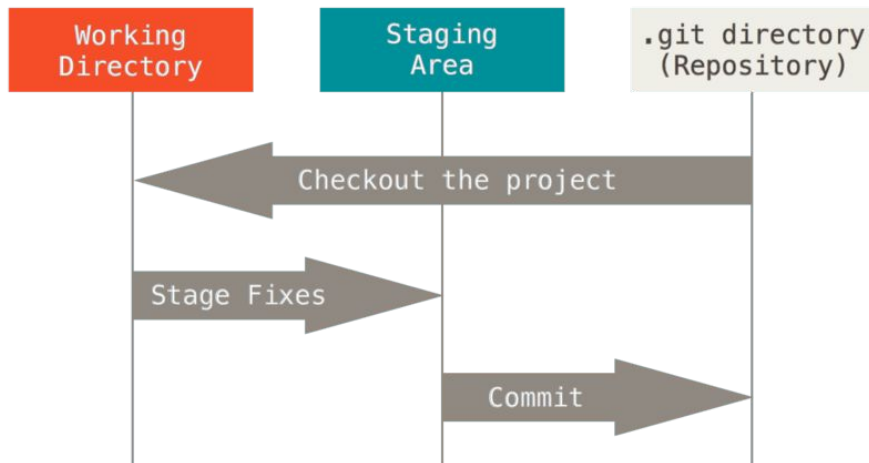
- Ratkaisevat edellisiä ongelmia
- Git, Mercurial, Bazaar...
- Jokainen kehittäjä omistaa oikeastaan täyden varmuuskopion datasta

GIT

perusteet

Gitin kolme tilaa

- Tiedostoilla kolme eri tilaa:
 - Pysyvästi muutettu (*committed*)
 - Muutettu (*modified*)
 - Valmisteltu (*staged*)
- Muodostaa yhdessä kolme osaa git projektissa:



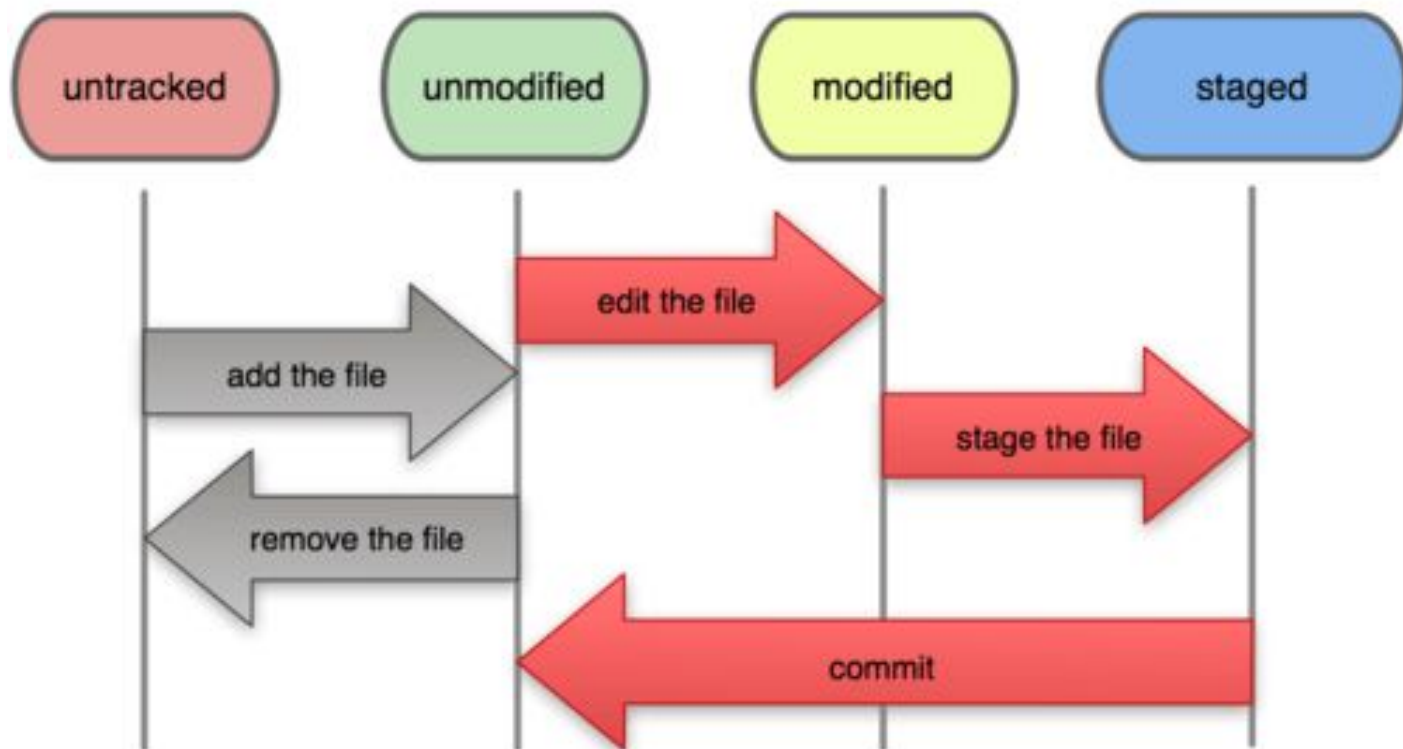
Git repon luonti

- Kaksi lähestymistapaa:
 - Tuo jo olemassa oleva projekti tai hakemisto Gittiin
 - Kloonaa olemassa oleva repo toiselta palvelimelta
- `git init`
- `git clone [url]`

Muutosten tallennus

- Jokainen tiedosto joko jäljitetty tai jäljittelemätön
- Jäljitetyt tiedostot:
 - Esiintyivät viime tilannekuvassa
 - Muokkaamattomia, muokattuja tai valmisteltuja
- Jäljittämättömät tiedostot:
 - Kaikki muut tiedostot
 - Jotka eivät esiintynyt viime tilannekuvassa
 - eikä valmistelualueella
- Seuraa tiedostojen tilaa ***git status*** komennolla

File Status Lifecycle



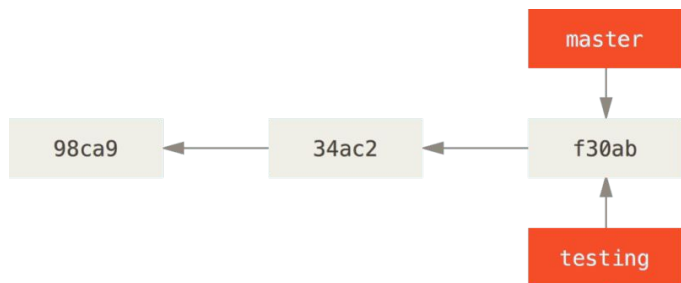
GIT Haararar

Haarat

- Gitissä kevyt ja nopea operaatio: käytä sitä!
- Gitissä haara on kevyt liikuteltava pointteri johonkin pysyvään muutokseen (commit)
- Oletushaaran nimi on “master”

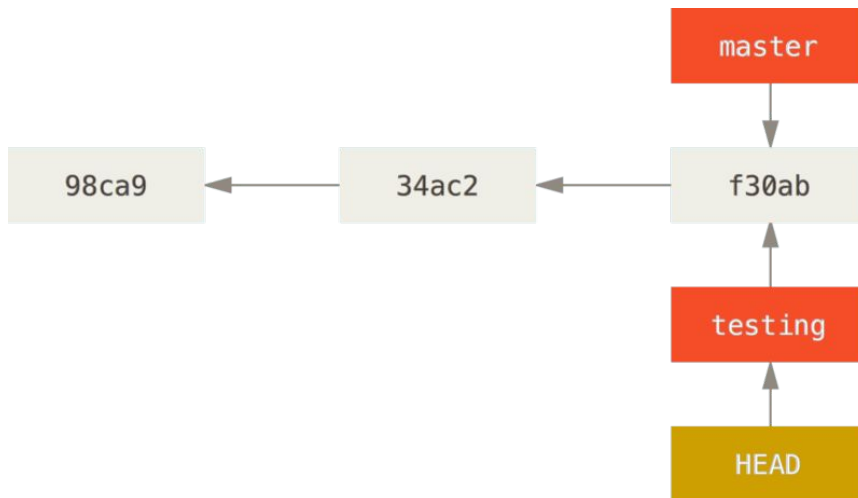
Uuden haaran luonti

- ***git branch*** komennolla
- Nyt kaksi haaraa osoittaa samaan pysyvään muutokseen (missä olet)
- Sen takia olemassa spesiaali “HEAD” osoitin:
 - viittaa haaraan, missä olet



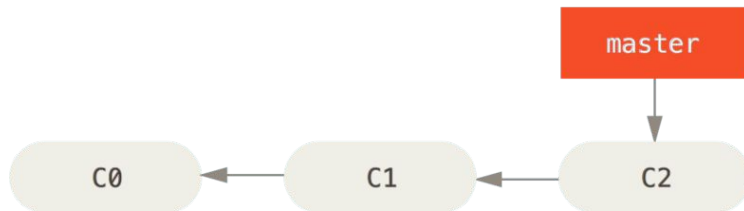
Haaran vaihtaminen

- ***git checkout*** komennolla
 - Muuttaa HEAD:in osoittamaan toiseen haaraan
 - Vaihtaa tiedostot (eri) pysyvän muutoksen mukaiseksi

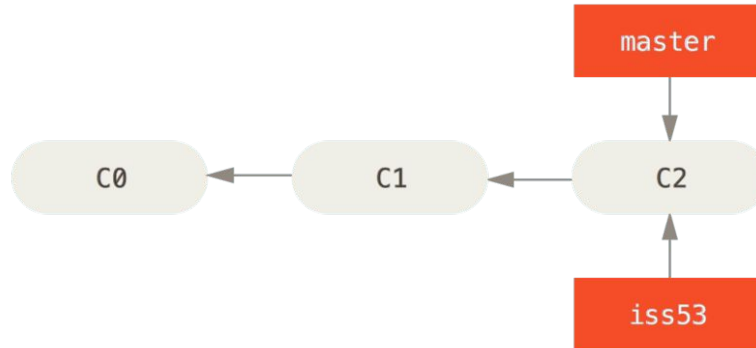


Haarojen yhdistys

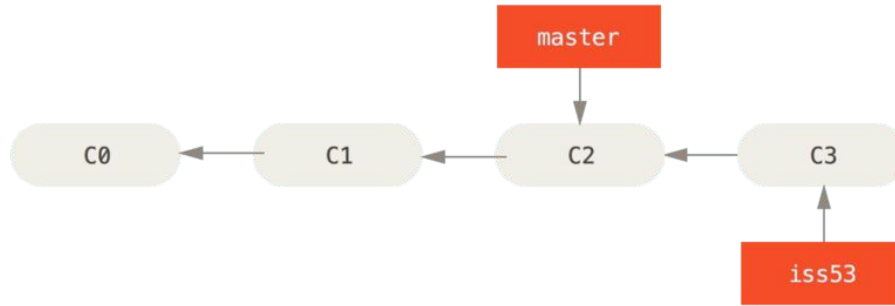
- ***git merge*** komennolla
- Katsotaan esimerkin avulla:
- Olet esimerkiksi koodaamassa nettisivuja ja olet tehnyt master haaraan muutoksia:



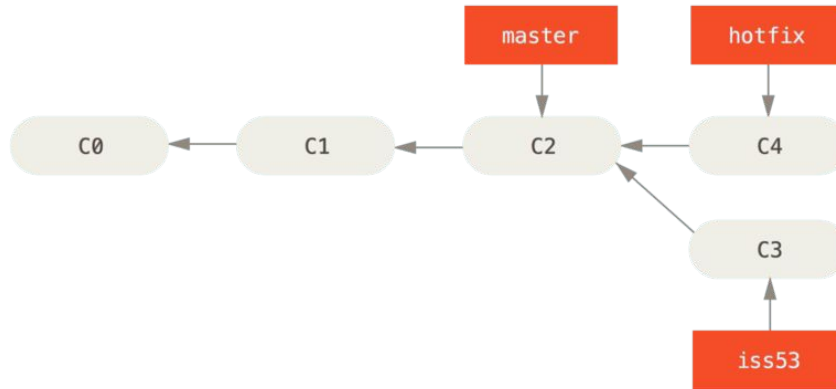
- Päättät sitten luoda uuden haaran nimeltä “iss53”, jossa korjaat footerin bugia.
- **git branch iss53** - luot uuden haaran
git checkout iss53 - siirryt iss53 haaraan



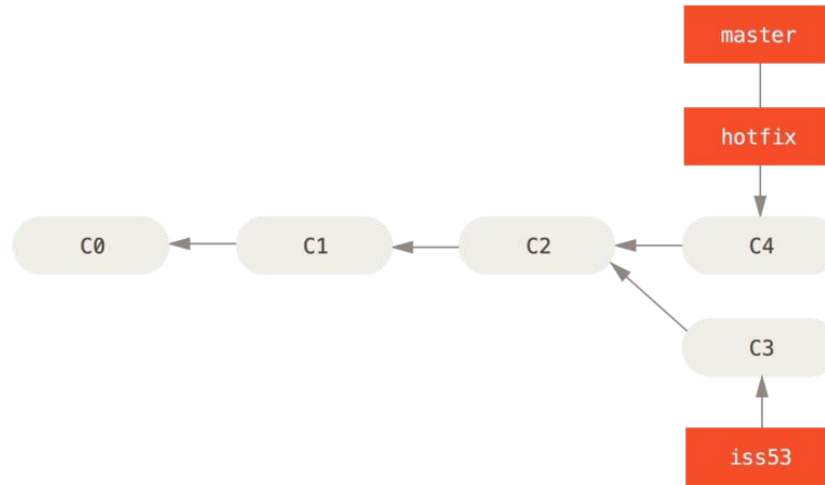
- Korjaillet footer bugia ja teet pysyviä muutoksia nyt iss53 haarassa.
- Huomaa kuinka master haara viittaa edelleen vanhaan muutokseen, josta lähdit ja iss53 jatkaa eteenpäin.



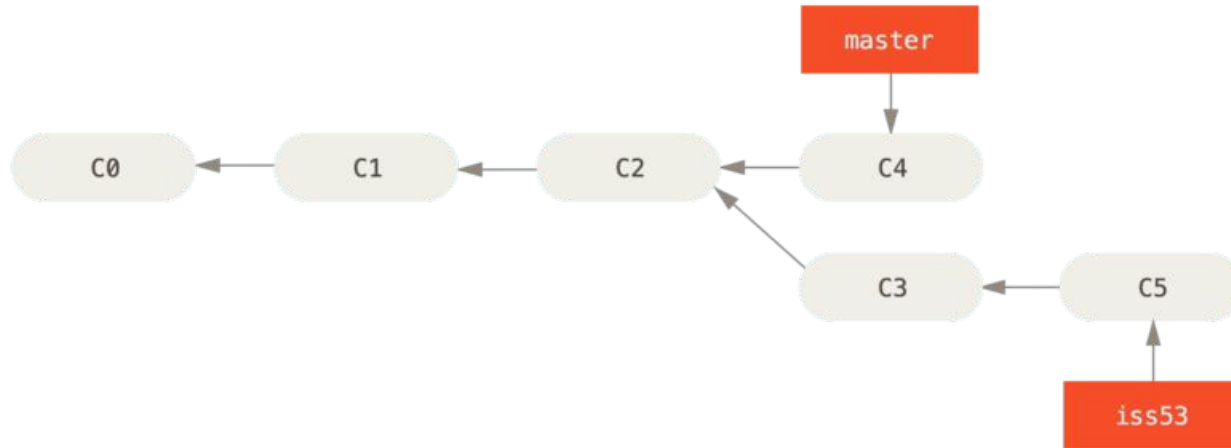
- Mutta sitten kesken footer bugin korjausta, kaveri soittaa ja ilmoittaa vakavasta kirjoitusvirheestä etusivulla.
- Hyppäät takaisin master haaraan ja luot uuden haaran “hotfix”, jossa korjaat typon.



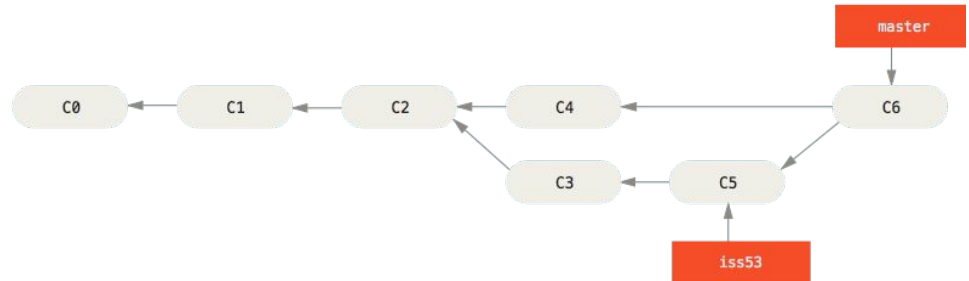
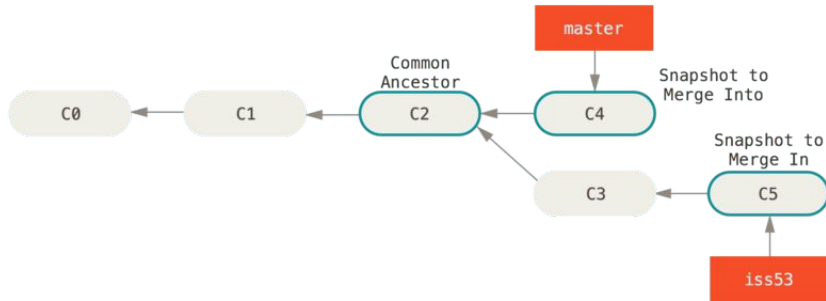
- Korjaat typon ja olet valmis yhdistämään “hotfix” ja “master” haaran.
- ***git checkout master***
- ***git merge hotfix***



- Poistat “hotfix” haaran koska se on yhdistetty masteriin.
- Hyppäät takaisin “iss53” haaraan ja jatkat footer bugin korjausta.



- Ja nyt olet valmis myös yhdistämään “iss53” ja “master” haarat
- Haaroilla ei ole yhteistä kantaa, git muodostaa “merge commitin”.

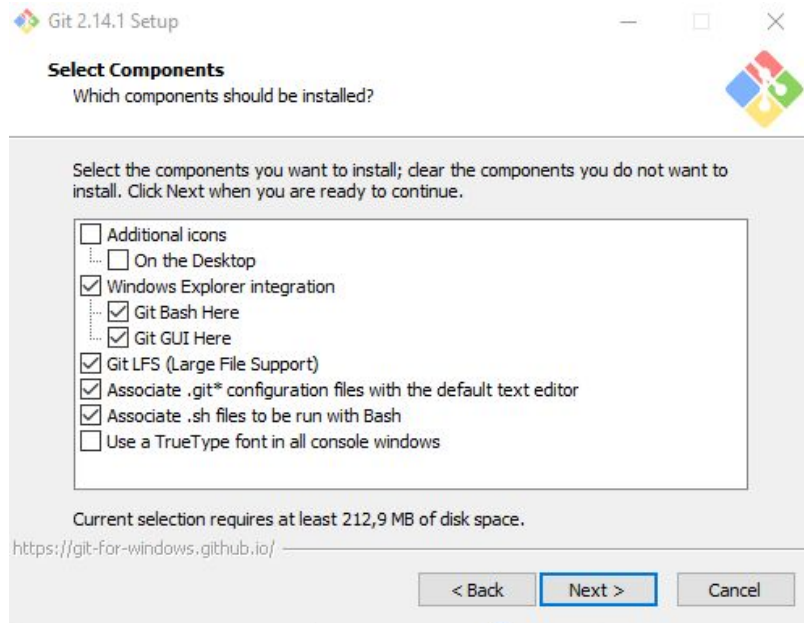


Konfliktit yhdistämisessä

- Yhdistäminen pysähtyy ja git odottaa kunnes käyttäjä on selvittänyt konfliktit
- **git status** komento kertoo missä tiedostossa konfliktit esiintyvät
- Git merkkaa tiedostoihin tiedot konfliktista
 - Käyttäjä avaa ne manuaalisesti ja selvittää ongelman/ristiriidan
- Selvityksen jälkeen valmistelee tiedosto **git add** komennolla
- Viimeistele yhdistäminen tekemällä pysyvä muutos: **git commit**

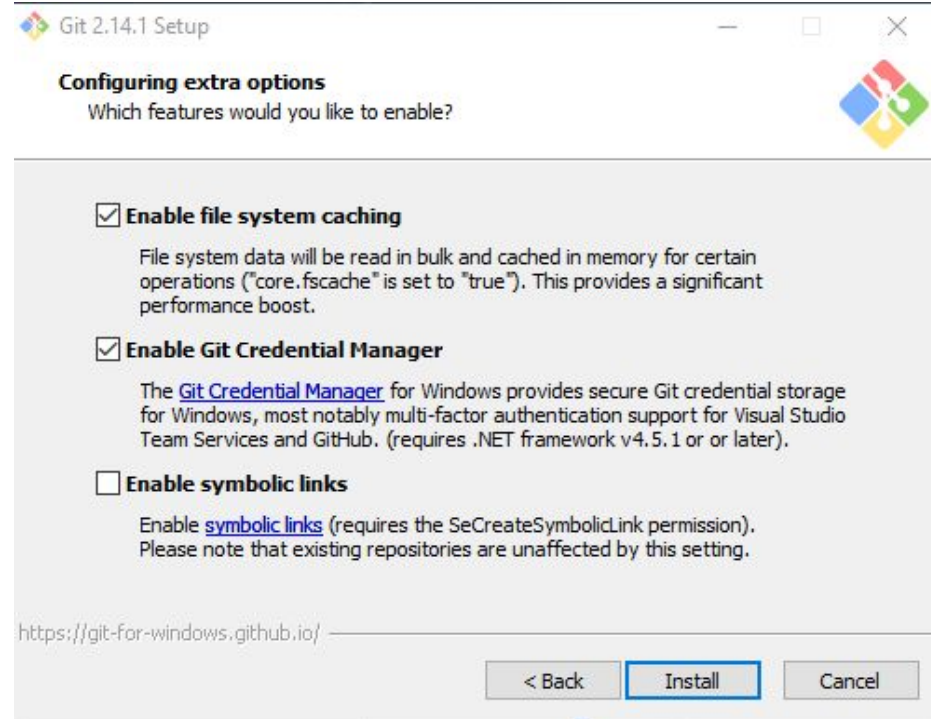
Asenna GIT #1

- Lataa GIT omalle koneellesi:
<https://git-scm.com/>
- **Asennus**
 - Komponentit
 - Windows Explorer integraatio (bash & gui)
 - PATH
 - “Use git from the windows Command Prompt”
 - Tällöin hommat toimii myös oletus command promptilla, eikä pelkästään git bash.



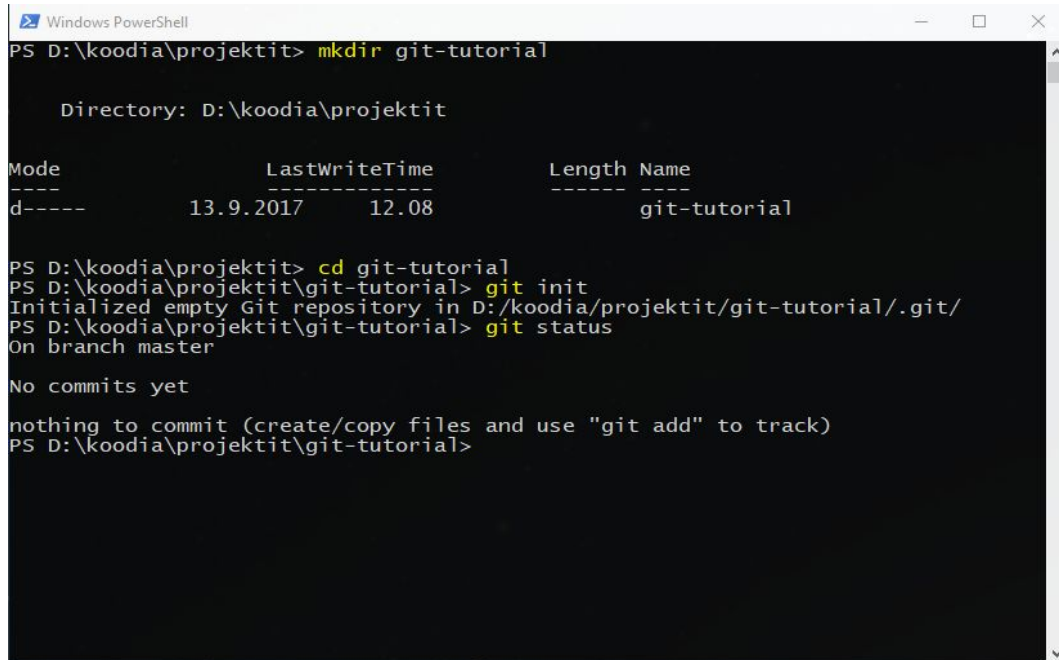
Asenna GIT #2

- Line endings
 - checkout Windows-style, commit Unix-style line endings (Windows)
 - checkout as-is, commit unix-style line endings (MAC & linux)
- Terminal Emulator
 - Use MinTTY
- Extra options
 - enable file system caching
 - enable git Credential Manager



Local Repository

- Paikallinen git “repo”
 - Gitin avulla jollekin koneelle tehty tiedostosijainti. Lokaalissa git -repossa on aina .git kansio (huom. piilotiedosto)
- **git init**
 - Jos tarkoituksena on käyttää repoa vain paikallisesti, voidaan luoda repo suorittamalla git init -komento
 - Komennolla luodaan .git kansio ja tämän jälkeen git komennot toimivat repossa



```
Windows PowerShell
PS D:\koodia\projektit> mkdir git-tutorial

Directory: D:\koodia\projektit

Mode                LastWriteTime         Length Name
----                -
d-----          13.9.2017    12.08             git-tutorial

PS D:\koodia\projektit> cd git-tutorial
PS D:\koodia\projektit\git-tutorial> git init
Initialized empty Git repository in D:/koodia/projektit/git-tutorial/.git/
PS D:\koodia\projektit\git-tutorial> git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
PS D:\koodia\projektit\git-tutorial>
```

Kuvassa luodaan projektille git-hakemisto

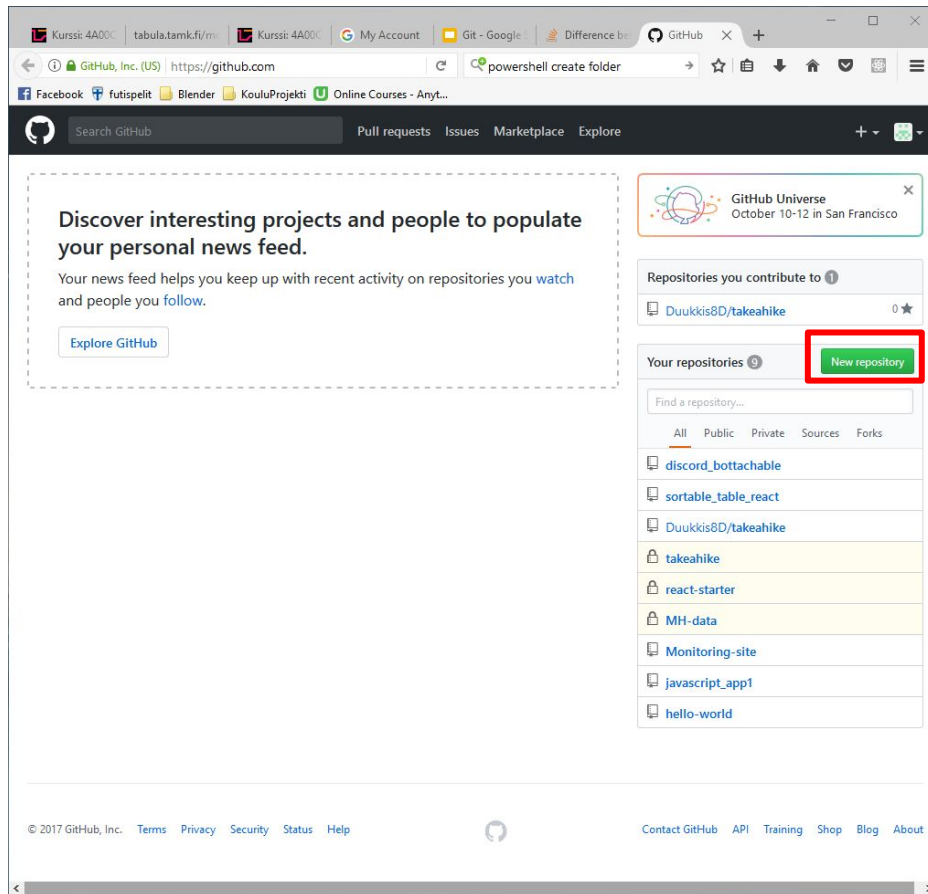
- **git Status** -komennolla näet, onko repo luotu.

Remote Repository

- Github/Gitlab/Bitbucket repo
 - verkkopalvelussa sijaitseva repo, johon koodi lähetetään omalta koneelta
 - Käytetään projektitiedostojen säilyttämiseen ja eri tiedostoversioiden hallintaan.
- **Tehtävä 1:** Tee tunnus githubiin ja kirjaudu sisään.

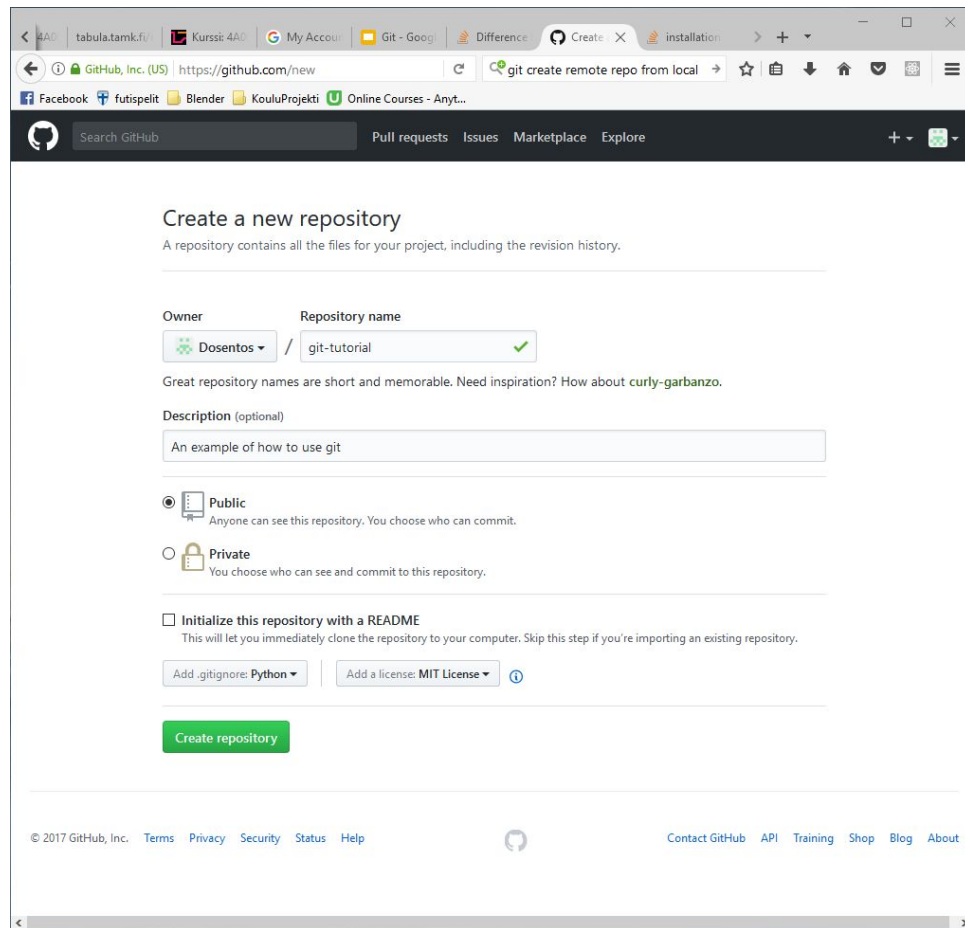
Create Github repo #1

Tehtävä 2: Luo uusi repo



Create Github repo #2

- Täytä tarvittavat tiedot
- Voit jättää repon julkiseksi
 - **Pro tip!**
 - Opiskelijana saat githubiin privaterepot käyttöön.
<https://education.github.com/pack>
- Lisenssiä ja .gitignorea ei tarvita, mutta saat lisätä jos haluat.



440 tabula.tamk.fi Kurssi: 440 My Account Git - Goog Difference Create X installation

GitHub, Inc. (US) https://github.com/new git create remote repo from local

Facebook futispelit Blender Kouluprojekti Online Courses - Anyt...

Search GitHub Pull requests Issues Marketplace Explore

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner Repository name

Dosentos / git-tutorial

Great repository names are short and memorable. Need inspiration? How about curly-garbanzo.

Description (optional)

An example of how to use git

☒ Public
Anyone can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: Python Add a license: MIT License

Create repository

© 2017 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

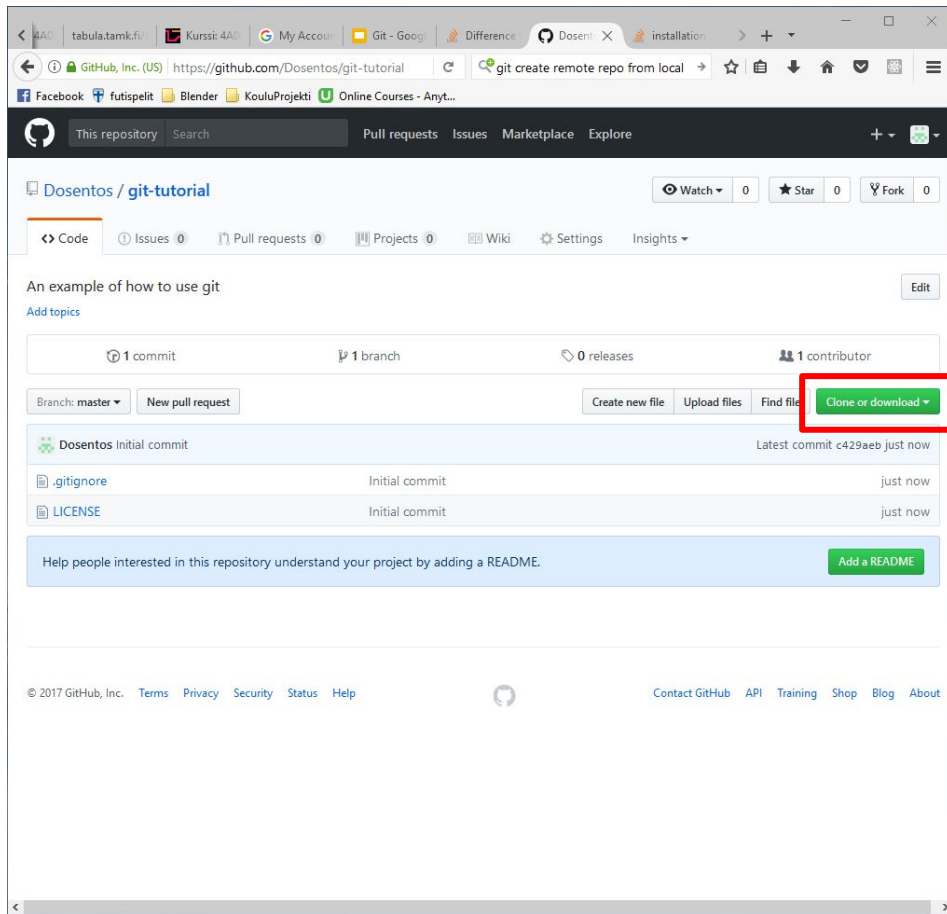
Clone repository

Tehtävä 3: Kloonaa repo.

- Kopioi https osoite githubista
- Avaa komentorivi haluamaasi sijaintiin (shift + mouse2)
- Kloonaa repo komennolla

git clone [url]

- jossa [url] -tekstin tilalle liität kopioimasi https-osoitteen githubista.



git branch

- **git branch** -komennolla nähdään tämänhetkinen haara
- tähän voi kirjoittaa perään myös haaran nimen, jolloin luodaan uusi haara annetulla nimellä.
- Haaraa luodessa täytyy huomioida, missä kohtaa versionhallinnan puuta sillä hetkellä on, jotta uusi haara luodaan oikeasta kohdasta, sillä uuteen haaraan kopioidaan sen hetkisestä paikasta tiedostot.

```
PS D:\koodia\projektit\git-tutorial> git branch
* master
PS D:\koodia\projektit\git-tutorial>
```

```
PS D:\koodia\projektit\git-tutorial> git branch develop
PS D:\koodia\projektit\git-tutorial> git branch
develop
* master
PS D:\koodia\projektit\git-tutorial>
```

Taululle piiretään esimerkki, jos tarvetta...

git checkout [branch]

- Checkout -komennolla voidaan siirtyä haarasta tai commitista toiseen
- Esim.
 - Luo uusi branch (`git branch new_branch`)
 - Nyt meillä on uusi haara, mutta emme ole siirtyneet tekemämme haaran sisään, joten meidän täytyy vielä ajaa komento `git checkout new_branch`
 - Nyt olemme new_branch -haarassa ja voimme alkaa siellä kehittämisen
- Joskus on tarvetta myös siirtyä vanhan commitin kohdalle. Tällöin voit ajaa komennon `git checkout [commit_hash]`. Commit hash on kullekin commitille automattisesti generoitu uniikki tunnus, jonka näet esimerkiksi komennolla `git log`

git checkout/reset [file]

- Checkout -komennolla voidaan myös hakea takaisin jo muutettuja tiedostoja
- esim 1.
 - Pirjo kirjoittaa pitkän pätkän PHP-koodia index.html -tiedostoon
 - Pirjo tallentaa muutokset ja toteaa, että “eiku...”
 - Pirjo ottaa koodinsa talteen toiseen tiedostoon.
 - Pirjo ajaa komennon “**git checkout index.html**” hakeakseen vanhan index.html tiedoston takaisin. (HUOM! Tämä toimii vain jos Pirjo EI ole ehtinyt ajamaan **git add** tai **git commit**)
- Jos Pirjo ajaa komennon “**git add .**” ennen kuin huomaa virhettään, hänen pitää
 - otaa koodi talteen toiseen tiedostoon
 - ajaa komento “**git reset index.html**” (kumoaa git add index.html -komennon)
 - ajaa komento “**git checkout index.html**” (kumoaa tehdyt, gittiin tallentamattomat muutokset index.html tiedoston osalta)

git pull

- Pull - komennolla haetaan remotesta uusin versio reposta.
- Vaikka komennon ajaa kesken työskentelyn, se ei yliaja jo tekemiäsi muutoksia.
- Esim.
 - Pirjo kirjoittaa neljän päivän tauon jälkeen uudelleen PHP-koodinsa, mutta kesken kirjoittamisen hän muistaa, että työpartnerinsa Orvokki on edellisenä päivänä tehnyt muutoksia koodiin.
 - Pirjo ajaa komennon "**git pull**", jonka jälkeen hän näkee komentorivin syötteestä, että jotakin päivitettävää oli, joten hän siirtyy tarkastelemaan muutoksia komennolla "**git log --graph**".
 - Jos pirjo olisi tehnyt muutoksia tiedostoihin, joihin Orvokki on edellisenä päivänä tehnyt muutoksia, niin git pull ilmoittaisi konfliktista. Tällöin Pirjon olisi helpointa ottaa tekemistään muutoksista varmuuskopio, kumota tekemänsä koodi ja ladata Orvokin koodit. Sen jälkeen hänen olisi helpompi ratkaista konflikti kirjoittamalla oma koodinsa Orvokin koodin päälle.

git add [file]

- Komennolla siirretään tehty koodi välimuistiin, jossa se on tallessa seuraavaan committiin asti. git add on sitä varten ettei kaikkia tiedostoja ole pakko commitoida.
- Komennon perään voidaan kirjoittaa tiedoston nimi tai ".", joka lisää kaikki lisäämättömät tiedostot.
- Esim.
 - Pirjo tekee delete_bad_spaces-tiedostoon funktion, jolla voidaan poistaa ylimääräiset välilyönnit. Samalla pirjo työstää toista tiedostoa, jossa hän tarvitsee kyseistä funktiota.
 - Pirjo saa funktion valmiiksi, mutta ei halua commitoida tekemäänsä muuta koodia samalla.
 - Pirjo ajaa komennon **git add delete_bad_spaces.js**
 - ja Pirjo ajaa **git commit "Added new file that contains function that can delete bad spaces from string"**
 - Nyt Pirjo on commitoinut esimerkillisen commitin, joka ei sisällä muuta kuin commit -viestissä lukevia asioita.

`git commit (-m "commit viesti")`

- Commit -komennolla siirretään tiedostot versionhallintaan.
- Pelkkä `git commit` avaa komentoriville tekstityökalun, jonne voi ja kannattaa ja pitäisi kirjoittaa commit -viesti.
- `git commit -m "commit viesti"` ohittaa tekstityökalun avaamisen ja viestiksi asetetaan -m -parametrin jälkeinen merkkijono
- esim. `git commit "Added new file: login.php"`
- `git commit` siirtää versionhallintaan kaikki tiedostot, jotka on lisätty `git add` -komennolla.

git remote

- Remote komennolla näkee mitä palvelimia on lisätty repoon. Palvelimia voi olla useita, koska joskus koodi halutaan työntää useaan paikkaan esim. asiakkaan versionhallinta, oman firman versionhallinta, tuotantoympäristö (esim.heroku), jne..
- **git remote add [nimi] [osoite]** -komennolla voidaan lisätä uusi remote.
- Remotet ovat paikalliseen ympäristöön asetettavia osoitteita, jotka tallentuvat paikallisesti repoon.
- Esim.
 - Pirkko lisää uuden remoten komennolla **git remote add kissa <https://git.kissa.fi>**
 - Orvokki ei pysty käyttämään komentoa **git push kissa** ennen kuin hän on itsekin lisännyt kissa-nimisen remoten omaan lokaaliin repositoryynsa.

git push

- Push komennolla siirretään tiedostot remoteen.
- Push tarkoittaa sitä, että tiedostot siirtyvät palvelimelle (esim. Github, gitlab tms)
- Oletuksena on origin -niminen repository, joka on se repository, josta olet alunperin kloonannut projektin (ellet ole muuttanut sitä jälkeen)
- Jos remoteja on useita (esim. heroku ja github), voidaan **git push** -komennon perään kirjoittaa remoten nimi (**git push heroku**) ja lähettää tiedostot näin muualle kuin alkuperäiseen remoteen
- Jos syystä tai toisesta halutaan työntää lokaalissa repossa sijaitsevat tiedostot remoteen niin, että lokaali develop haaran tiedostot menevät vaikkapa remoten master-haaraan, voidaan lisätä komennon perään **lokaali_haara:remote_haara**

git stash

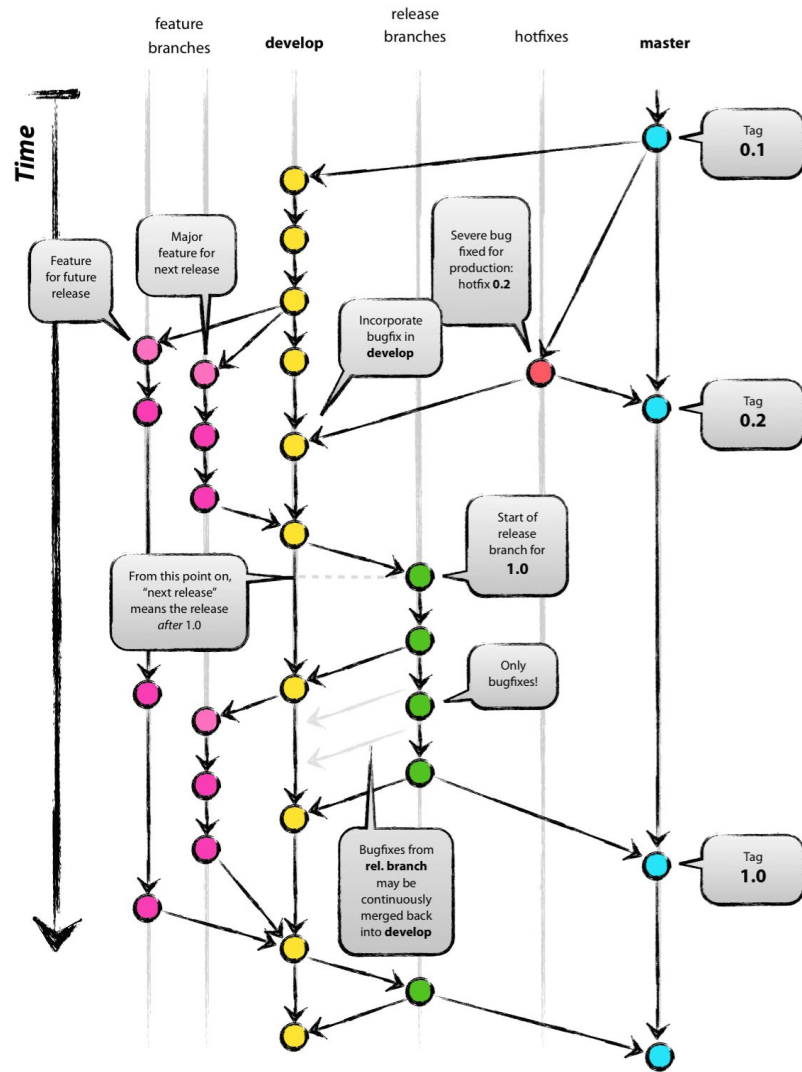
- Tallentaa lokaalit muutoksen ja palauttaa viimeisimpään HEADin tilaan
- **git stash save "WIP: new footer"**
 - Mahdollisuus vaihtaa haaraa, tehdä muita committeja...
- **git stash list**
- **git stash apply <stash>**
 - Palauttaa aiemmin tehdyt muutokset
- **git stash drop <stash>**
 - Poistaa määritellyn tai viimeisimmän
- **git stash clear**
 - Poistaa kaikki

Merge conflicts

- Tapahtuu henkilö A ja henkilö B molemmat ovat tehneet muutoksia samaan tiedostoon samalle riville
- Konflikti täytyy ratkaista ennen kuin voidaan jatkaa
- **git mergetool**
- Työkaluja
 - [Meld](#)
 - [DiggMerge](#)
 - [TortoiseGit](#)
 - IDEt

Git workflow

- Masteriin ei voi commitoida
 - Vain Pull Request
- Feature/Bugfix haarat
 - `git branch feature/pagination`
 - `git checkout feature/pagination`
 - *Coding intensifies*
 - `git add .`
 - `git commit -m "Added pagination"`
 - `git push --set-upstream origin feature/pagination`
 - `git push`
 - Tee Pull Request
 - Peer review
 - Merge feature to master



Yleiset ongelmatilanteet #1

```
PS D:\koodia\projektit\git-tutorial> git push
fatal: The current branch develop has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin develop
```

- Git push epäonnistuu, koska luomaasi develop -haaraa ei ole asetettu upstreamiksi.
- Tämä pitää tehdä vain kerran kun haara on luotu. Syötä ehdotettu komento:
git push --set-upstream [vanha_branch] [uusi_branch]
- Tämä komento muuttaa uuden haaran oletus haaraksi lokaalissa kehitysympäristössä (eli vain koneella, jolla oot sillä hetkellä).

GIT vs. GitHub, GitLab, Bitbucket

mitä ne on, mitä niillä tekee, miten niitä käytetään

Mitä ne on?

Selainpohjainen käyttöliittymä lähdekoodin hallintaan

Git on CLI (Command-line interface) versionhallintaan

Mihin niitä käytetään?

- Pull request
- Code review
- Inline editing
- Issue tracking
- Markdown support
- Two factor authentications
- Advanced permission management
- Hosted static web pages
- Feature rich API
- Fork / Clone Repositories
- Snippets
- 3rd party integrations

Demo

repon luominen, commits, pull requests, branches,
workflow,

Linkkejä

- <http://rogerdudler.github.io/git-guide/>
- <https://www.atlassian.com/git/tutorials/what-is-version-control>
- <http://nvie.com/posts/a-successful-git-branching-model/>
- <https://www.gitignore.io/>
- <https://try.github.io/levels/1/challenges/1>
- <https://githowto.com/>
- <https://git-scm.com/book/en/v2>
- <https://www.youtube.com/watch?v=0fKg7e37bQE>