Lappeenrannan–Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma

Kandidaatintyö

Mikko Mustonen

PARHAITEN OPETUSKÄYTTÖÖN SOVELTUVAN VERSIONHALLINTAJÄRJESTELMÄN LÖYTÄMINEN

Työn tarkastaja: Tutkijaopettaja Uolevi Nikula

Työn ohjaaja: Tutkijaopettaja Uolevi Nikula

TIIVISTELMÄ

LUT-yliopisto
School of Engineering Science
Tietotekniikan koulutusohjelma

Mikko Mustonen

Parhaiten opetuskäyttöön soveltuvan versionhallintajärjestelmän löytäminen

Kandidaatintyö

2019

31 sivua, 8 kuvaa, 2 taulukkoa

Työn tarkastajat: Tutkijaopettaja Uolevi Nikula

Hakusanat: versionhallinta, versionhallintajärjestelmä, Git, GitLab, SVN, Subversion, oppimateriaali
Keywords: version control, version control system, Git, GitLab, SVN, Subversion, learning material

LUT-yliopistossa on tietotekniikan opetuksessa käytetty Apache Subversionia versionhallintaan. Subversionin käyttö kuitenkin johtaa ylimääräisiin ylläpitotoimiin LUTin tietohallinnolle. Lisäksi Subversionin julkaisun jälkeen on tullut uusia versionhallintajärjestelmiä ja tässä työssä tutkitaankin, olisiko Subversion syytä vaihtaa johonkin toiseen versionhallintajärjestelmään opetuskäytössä. Työn tavoitteena on löytää opetuskäyttöön parhaiten soveltuva versionhallintajärjestelmä ja tuottaa sille opetusmateriaalia. Työssä havaittiin, että Git on suosituin versionhallintajärjestelmä ja se on myös suhteellisen helppo käyttää. Lisäksi GitLab on tutkimuksen mukaan Suomen yliopistoissa käytetyin ja ominaisuuksiltaan ja hinnaltaan sopivin Gitin web-käyttöliittymä. Näille tehtiin opetusmateriaali, joka pohjautuu Pro Git kirjaan ja GitLabin dokumentaatioon.

ABSTRACT

LUT University
School of Engineering Science
Degree Program in Computer Science

Mikko Mustonen

Finding the most suitable version control system for education

Bachelor's Thesis

31 pages, 8 figures, 2 tables

Examiners: Associate Professor Uolevi Nikula

Keywords: version control, version control system, Git, GitLab, SVN, Subversion, learning material

In LUT-University Apache Subversion has been used in the computer sciense education. However the use of Subversion causes extra work for the LUT IT-services. In addition many version control systems have been released after Subversion and thus in this work it is studied should LUT change Subversion to some other version control system in education. The aim of this work is to find the best version control system for education and produce study material for it. It was found that Git is the most popular version control system and it is also relatively easy to use. Also GitLab was found to be the best priced and the most used web-interface for Git in Finnish universities. Study material was made for these, which is based on Pro Git book and GitLab's documentation.

SISÄLLYSLUETTELO

SYMBOLI- JA LYHENNELUETTELO

ACM	Association for Computing Machinery
CM	Configuration management
CVS	Concurrent Versions System
GNU	GNU's Not Unix!
IEEE	Institute of Electrical and Electronics Engineers
RCS	Revision Control System
SCCS	Source Code Control System
SCM	Software configuration management
SVN	Apache Subversion
VSS	Visual SourceSafe

1 JOHDANTO

1.1 Tausta

Ohjelmistoprojektien kasvaessa tulee koodin hallinnasta hyvin haastavaa. Tätä varten on kehitetty versionhallintajärjestelmiä. Voidaan sanoa, että versionhallintajärjestelmät ovat hyvin tärkeä osa ohjelmistokehitystä.[1] LUT-yliopistossa on tietotekniikan opetuksessa käytetty versionhallintaan Apache Subversionia. LUTin tietohallinto ei kuitenkaan virallisesti tarjoa versionhallintajärjestelmien ylläpitoa. Tästä johtuen LUTin tietohallinnolle aiheutuu ylimääräisiä ylläpitotoimenpiteitä, jotka he tekevät muiden töiden ohessa. He joutuvat tekemään kursseja varten SVN (Apache Subversion) tietovarastoja (repository) ja eniten työtä aiheuttaa käyttäjien ja niiden oikeuksien lisääminen ja hallinta. Tämän vuoksi on syytä tutkia voisiko heille aiheutuvaa työtä vähentää. Yleinen käsitys on, että Git olisi teollisuudessa suosituin versionhallintajärjestelmä, joten on syytä tutkia, onko tämä totta ja olisiko sen opettaminen LUTissa järkevää. Tietohallinnon tarjoamasta versionhallinta palvelusta olisi hyötyä myös muille koulutusohjelmille. Sähkötekniikan osastolla ja laskennallisen tekniikan konenäön yksiköllä on heidän itse ylläpitämät GitLab instanssit, joten tietohallinon ylläpitämä versionhallintajärjestelmä helpottaisi heidän työtään.

1.2 Tavoitteet ja rajaukset

Tämän työn tavoitteena on valita parhaiten LUTin opetuskäyttöön soveltuva versionhallintajärjestelmä sekä web-käyttöliittymä sille ja tuottaa niille opetusmateriaalia. Versionhallintajärjestelmän valintakriteerit ovat seuraavat:

- käyttö teollisuudessa
- käyttö opetuksessa
- käyttöönoton helppous
- ominaisuudet
- hinta.

Web-käyttöliittymän valintakriteerit ovat:

- ylläpidon helppous
- käyttöönoton helppous

- kirjautuminen LUTin tunnuksilla
- ominaisuudet
- hinta

Tutkimuskysymyksinä ovat seuraavat:

- Mitä versionhallintajärjestelmää käytetään teollisuudessa eniten?
- Mitä versionhallintajärjestelmää käytetään muualla opetukseen?
- Mikä versionhallintajärjestelmä sopisi parhaiten opetukseen LUT-yliopistossa?
- Mitä versionhallinnasta pitäisi opettaa?

Työssä ei oteta kantaa siihen, miten versionhallintajärjestelmän ylläpito hoidetaan tai voitaisiin hoitaa.

1.3 Työn rakenne

Luvussa 2 kerrotaan tutkimukseen käytetyistä lähteistä. Luvussa 3 käsitellään tutkimustulokset. Luvussa 4 tehdään johtopäätöksiä tutkimustulosten perusteella ja kerrotaan opetusmateriaalin teosta. Luvussa 5 pohditaan mahdollisia jatkotutkimuksen kohteita. Luvussa 6 tehdään yhteenveto tutkimuksesta.

2 TUTKIMUSMENETELMÄ

Tutkimukseen etsittiin tietoa verkkotietokannoista. Aluksi käytettiin Google Scholaria, mutta huomattiin, että suurin osa tuloksista oli ACM Digital Library (Association for Computing Machinery) tai IEEE Xplore (Institute of Electrical and Electronics Engineers) verkkotietokannoista ja niiden lisäksi oli paljon turhia tuloksia. Tämän vuoksi siirryttiin käyttämään näitä tietokantoja. ACM tietokannasta haettiin tietoa seuraavilla hakusanoilla:

- "version control" AND education
- "version control" AND survey.

IEEE tietokannasta haettiin tietoa seuraavilla hakusanoilla:

- "version control" AND switching AND NOT electr*
- "version control" AND education AND NOT electr*.

IEEE:stä haettaessa hakusanassa käytettiin *NOT electr** termiä poistamaan kaikki elektroniikkaan liittyvät hakutulokset. Tutkimukseen etsittiin tietoja myös erinäisistä verkkolähteistä ja versionhallintajärjestelmien kotisivuilta.

Tietokannoista saaduista tuloksista luettiin tiivistelmät ja niiden pohjalta karsittiin työhön täysin liittymättömät tulokset pois. Jäljellä olevat tutkimukset luettiin kokonaan ja niistä karsiutui vielä joitain pois.

3 VERSIONHALLINTAJÄRJESTELMÄT

Luvussa 3.1 kerrotaan versionhallintajärjestelmistä yleisesti ja niiden historiasta. Luvussa 3.2 kerrotaan mitä versionhallintajärjestelmiä Suomen yliopistoissa käytetään. Luvussa 3.3 on selvitys tutkimuksessa löytyneistä tiedoista versionhallintajärjestelmien suosiosta eri vuosilta. Luvussa 3.4 kerrotaan Gitin suosioon vaikuttaneista tekijöistä. Luvussa 3.5 on testausraportti SVN:n ja Gitin käytöstä ja niiden vertailua. Luvussa 3.6 kerrotaan Gitille olemassa olevista web-käyttöliittymistä ja vertaillaan niistä kolmea.

3.1 Versionhallintajärjestelmien taustaa

Ensimmäinen versionhallintajärjestelmä SCCS (Source Code Control System) kehitettiin 1970-luvun alussa [2]. Muita tunnettuja versionhallintajärjestelmiä ovat vuonna 1982 julkaistu RCS (Revision Control Sytem) [3], vuonna 1986 julkaistu CVS (Concurrent Versions System) [4], vuonna 2004 julkaistu SVN [5] ja vuonna 2005 julkaistu Git [6]. Versionhallintajärjestelmiä on näiden lisäksi kehitetty hyvin monia. Taulukossa 1 on enemmän tai vähemmän kattava joukko versionhallintajärjestelmiä. Osa Taulukossa 1 olevista versionhallintajärjestelmistä ei ole aktiivisessa kehityksessä tai edes ylläpidetty.

Taulukko 1. Lista versionhallintajärjestelmistä [7]

Versionhallintajärjestelmä
AccuRev SCM (software configuration management)
Azure DevOps
GNU Bazaar (GNU's Not Unix!)
BitKeeper
ClearCase
Code Co-op
Codeville
CVS (Concurrent Versions System)
CVSNT
darcs

Dimensions CM (configuration management)
Endevor
Fossil
Git
GNU arch
IC Manage
PTC Integrity
Mercurial
Monotone
Perforce Helix Core
Plastic SCM
PVCS
Rational Team Concert
Revision Control System
SCM Anywhere
Source Code Control System
StarTeam
Subversion
Surround SCM
SVK
Synergy
Vault
Veracity
Vesta
Visual SourceSafe (VSS)

Nämä versionhallintajärjestelmät voidaan jakaa kahteen ryhmään: keskitettyihin ja hajautettuihin versionhallintajärjestelmiin.

3.1.1 Keskitetyt versionhallintajärjestelmät

Keskitetyt versionhallintajärjestelmät, kuten SVN ja CVS, tallentavat tiedostojen versiohistorian vain yhteen paikkaan palvelimelle. Tämän takia niitä kutsutaan keskitetyiksi. [8]. Versionhallintajärjestelmän käyttäjällä on vain yksi versio kerrallaan työhakemistossa. Tämän hyvänä puolena on binääritiedostojen käsittely. Koska käyttäjällä on vain yksi versio kerrallaan käsiteltävänä, ei haittaa, vaikka versionhallinnassa olisi monia versioita samasta binääritiedostosta, mikä saattaa viedä paljon tilaa [9]. Jos käyttäjä kuitenkin haluaa tarkastella jotain toista versiota, tallentaa muutokset versiohistoriaan tai tehdä mitä tahansa muita versionhallinta toimia, joutuu hän pyytämään niitä palvelimelta. Tämän takia keskitettyjä versionhallintajärjestelmiä ei voi käyttää ilman yhteyttä versionhallinta palvelimeen ja yhteys palvelimeen on pullonkaulana versionhallintatoimenpiteille [8], [9].

3.1.2 Hajautetut versionhallintajärjestelmät

Hajautetut versionhallintajärjestelmät, kuten Git ja Mercurial, tallentavat tiedostojen versiohistorian jokaisen käyttäjän tietokoneelle [8]. Hajautetut versionhallintajärjestelmät eivät varsinaisesti tarvitse keskitettyä palvelinta, koska muutokset versiohistoriaan voidaan jakaa monella tavalla esimerkiksi sähköpostin avulla [6]. Paljon kuitenkin käytetään keskitettyä palvelinta, jonne muutokset päivitetään, helpottamaan yhteistyötä. Koska jokaisella käyttäjällä on täysi versiohistoria omalla koneellaan ei versionhallintatoimenpiteisiin tarvita yhteyttä palvelimeen [10]. Vain tietojen päivittäminen palvelimelta tai palvelimelle vaatii verkkoyhteyden. Tästä johtuen versionhallintatoimenpiteetkin ovat nopeita suorittaa. Version tai haaran vaihdot ja muutosten tallennus ovat välittömiä. Koko versiohistorian tallentamisella on hyvänä puolena myös automaattinen tiedon varmuuskopioituminen [11]. Vaikka palvelin menisi rikki, saadaan tiedot todennäköisesti palautettua jonkun käyttäjän koneelta [6]. Koko versiohistorian tallentamisella on myös huonot puolensa. Kaikkien tiedostoversioiden säilytys vie tilaa. Tämä ei ole ongelma tekstitiedostojen kohdalla, koska vain muutokset tallennetaan, mutta binääritiedostoista tallennetaan jokainen versio kokonaisena [8], [9]. Tämä johtaa nopeasti tietovaraston paisumiseen. Tähän on onneksi kuitenkin kehitetty ratkaisuja. Esimerkiksi Git-LFS (Large File Storage) tallentaa Git versiohistoriaan vain pienen pointteritiedoston, joka osoittaa mistä sitä vastaava binääritiedosto löytyy [12]. Binääritiedostot tallennetaan eri paikkaan ja vain haluttu versio ladataan käyttäjän koneelle [12].

3.2 Versionhallinta Suomen yliopistoissa

Versionhallintajärjestelmien käyttöä opetuksessa selvitettiin tutkimalla mitä versionhallintajärjestelmiä Suomen yliopistoissa on tällä hetkellä käytössä. Tietoa haettiin Googlesta hakusanoilla muotoa yliopiston nimi tai yliopiston verkkotunnuksen osa, kuten lut, ja jokin seuraavista: versionhallinta, GitLab, GitHub, Git, Subversion tai SVN. Näillä hakusanoilla löydettiin seuraavaa. LUT-yliopistosta löytyi opasraportti, jossa mainittiin versionhallinan opettaminen C-ohjelmoinnin perusteet kurssilla [13]. Tällä kurssilla käytetään [14] opasta, josta selviää, että käytössä on SVN. LUT-yliopistolta löytyi myös Gitlab instanssi, mutta siitä ei selviä missä käytössä se on [15]. Tampereen yliopistossa opetetaan Gitiä ”Olio-ohjelmoinnin perusteet II” kurssilla [16]. Lisäksi heillä on Gitlab instanssi pilottikäytössä [17]. Taideyliopiston kurssin ”Pelimusiikin perusteet: Teoria ja työympäristöt” opintojakso kuvauksessa mainitaan versionhallinnan opiskelu, mutta ei mitä versionhallintaa käytetään [18]. Aalto-yliopistolla on Gitlab Community Edition instanssi [19], [20]. Itä-Suomen yliopiston Bioinformatics Centerissä vaikuttaa olevan käytössä Gitlab [21]. Oulun yliopistossa vaikuttaa olevan opetuksessa Git [22]. Jyväskylän yliopistolla on Gitlab instassi sekä YouSource palvelu, joilla voi käyttää Gitiä [23], [24]. Heillä on myös Trac palvelu SVN:n ja CVS:n käyttöön, mutta se ei välttämättä ole enää käytössä kun ottaa huomioon, että palvelun viimeisin tiedotus on viiden vuoden takaa [25]. Turun yliopistolla on Gitlab instanssi [26]. Åbo akademilla on Gitlab instanssi sekä SVN palvelin [27], [28]. ”Helsingin yliopistossa on käytössä Gitlab-versionhallintaohjelmisto.”[29] Heillä on oma Gitlab instanssi [30]. Maanpuolustuskorkeakoulusta, Svenska handelshögskolanista ja Lapin ja Vaasan yliopistoista ei löytynyt tietoa versionhallinnan opetuksesta näillä hauilla.

Tästä voidaan huomata, että Git on käytössä kaikkissa yliopistoissa, joista tieto käytetystä versionhallintajärjestelmästä löytyi. Tämä ei ole yllättävää, kun huomioi Gitin suosion oikeassa käytössä. Huomataan myös, että kaikkialla missä on käytössä Git on myös käytössä Gitlab. Tämä voi johtua siitä, että Gitlabista on avoimen lähdekoodin Community Edition, jota saa käyttää ilmaiseksi [31], jota ainakin Aalto-yliopisto käyttää. Gitlab tarjoaa myös Ultimate ja Gold lisenssit, joilla saa Gitlabin kaikki ominaisuudet käyttöön, ilmaiseksi opetusta varten [32].

Taulukko 2. Versionhallinta Suomen yliopistoissa

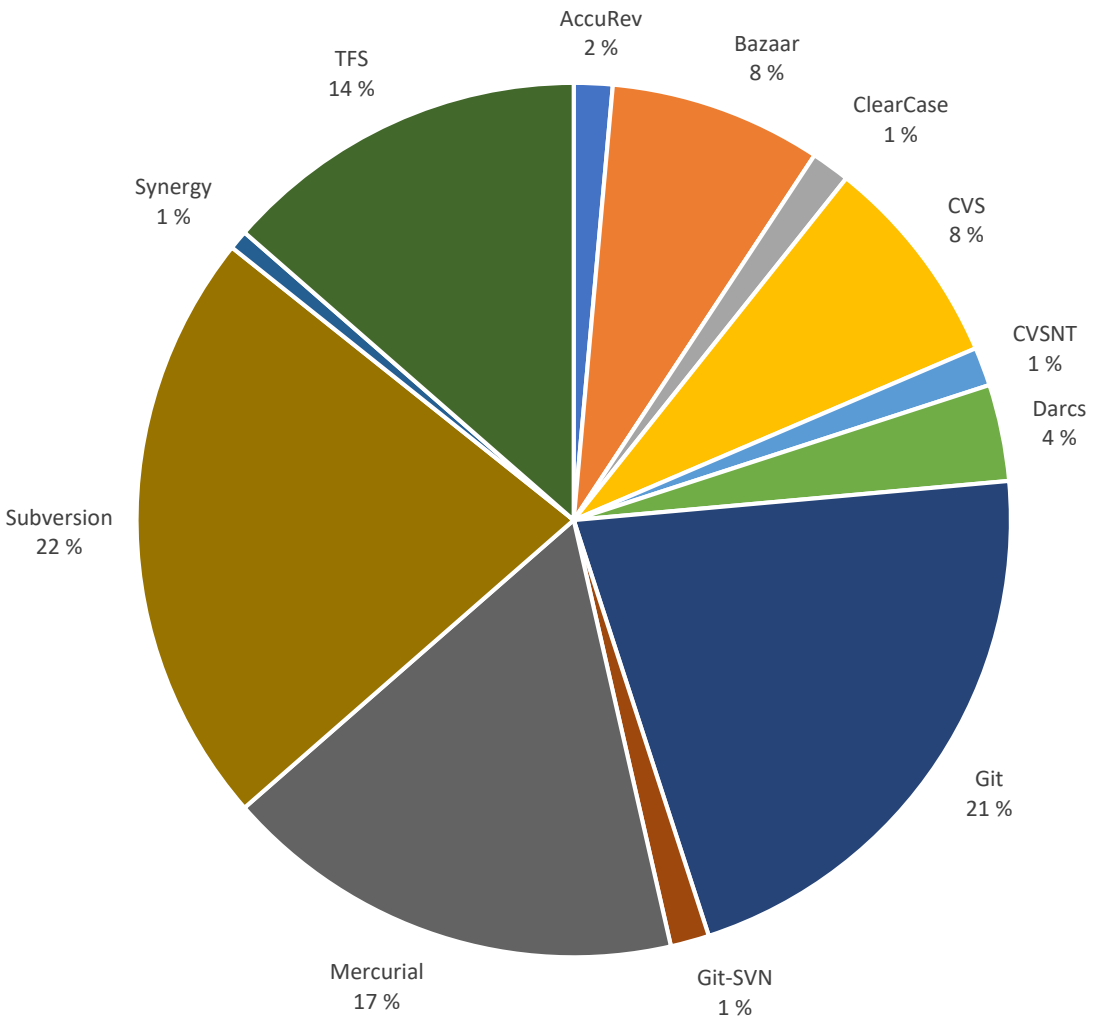
Yliopisto	Versionhallintajärjestelmä
Aalto-yliopisto	Git, Gitlab
Helsingin yliopisto	Git, Gitlab
Itä-Suomen yliopisto	Näyttäisi olevan Git ja Gitlab
Jyväskylän yliopisto	Git, Gitlab, YouSource, (SVN, CVS, Trac)
LUT-yliopisto	SVN, SVN palvelin, (Gitlab)
Lapin yliopisto	Ei tietoa
Maanpuolustuskorkeakoulu	Ei tietoa
Oulun yliopisto	Näyttäisi olevan Git
Svenska handelshögskolan	Ei tietoa
Taideyliopisto	Maininta versionhallinnan opetuksesta
Tampereen yliopisto	Git, Gitlab
Turun yliopisto	Git, Gitlab
Vaasan yliopisto	Ei tietoa
Åbo Akademi	Git, Gitlab, SVN, SVN frontend

3.3 Versionhallintajärjestelmien suosio

Jotta saadaan selville mitä versionhallintajärjestelmää kannattaisi käyttää opetuksessa on syytä tutkia mitä versionhallintajärjestelmää käytetään eniten teollisuudessa. Tästä on saatavilla tietoa osana eri asioita tutkivien tutkimusten kyselyitä ja myös erinäisistä verkkokyselyistä.

Vuonna 2011 julkaistussa tutkimuksessa [33] järjestettiin kysely johon vastasi 140 henkilöä. Kyselyssä selvitettiin mitä versionhallintajärjestelmää vastanneiden kehitystiimeissä käytetään. Nämä tulokset on nähtävissä Kuvassa 1. Käytetyimmät versionhallintajärjestelmät olivat Subversion, Git ja Mercurial, joista kaksi ensimmäistä olivat melkein yhtä suosittuja.

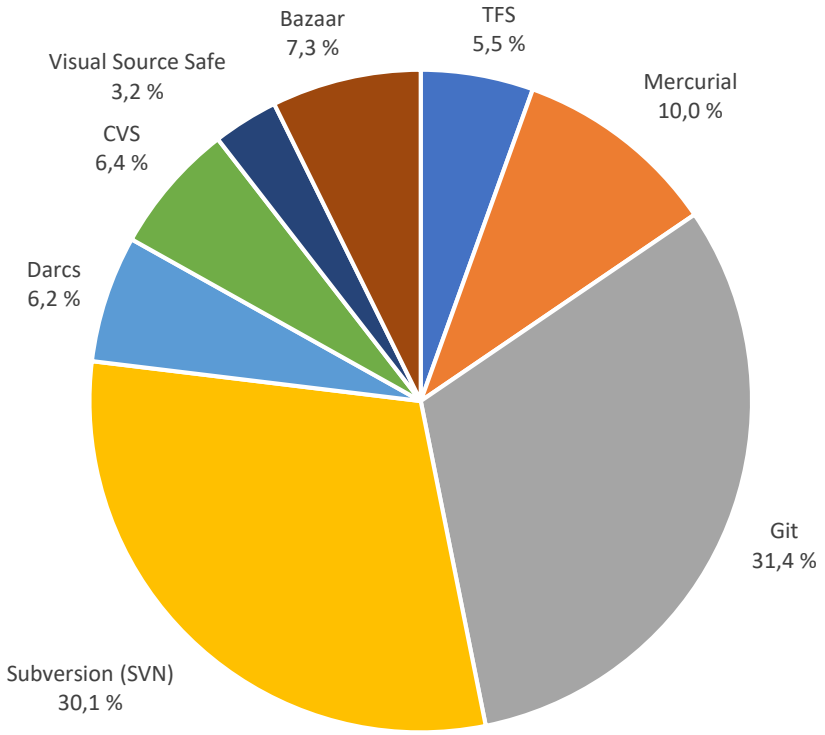
Vuonna 2013 julkaistussa tutkimuksessa [34] järjestettiin kysely, johon vastasi 42 henkilöä. Kyselystä saatiin Kuvan 2 mukainen versionhallintajärjestelmien käyttöjakauma.



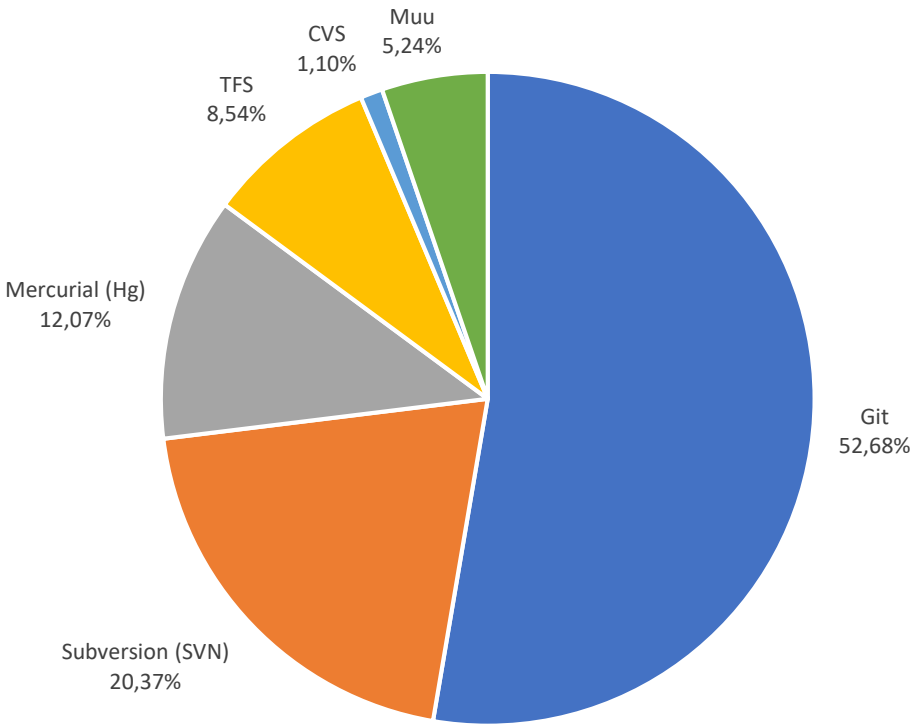
Kuva 1. Versionhallintajärjestelmien käyttö 2011 (n = 140)

Vuonna 2014 julkaistussa tutkimuksessa [10] järjestettiin kysely, johon vastasi 820 henkilöä. Kyselyn tuloksena saatiin Kuvan 3 mukainen versionhallintajärjestelmien käyttöjakauma. Vuonna 2014 julkaistiin myös Eclipsen käyttäjäkysely [35], jossa oli 876 vastaajaa. Kyselystä selviää versionhallintajärjestelmien käyttöprosentti Eclipsen käyttäjäkunnassa. Vuoden 2014 tulosten lisäksi on listattu myös vuosien 2011-2013 tulokset, mistä huomataan suosion muutos. Nämä tulokset ovat Kuvassa 4.

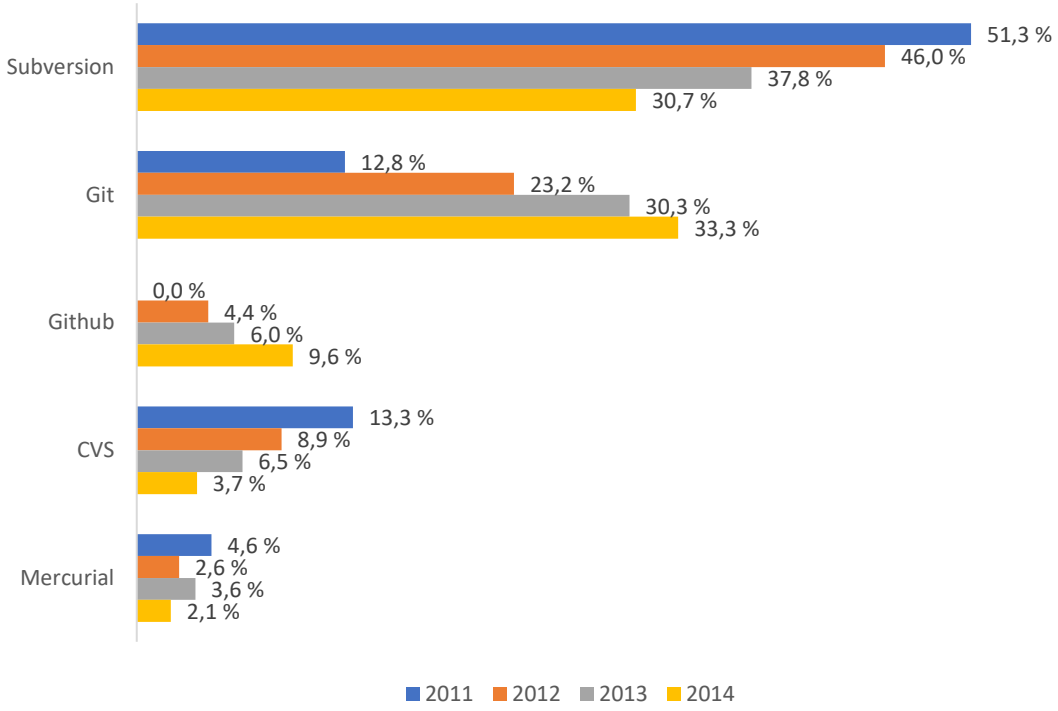
Vuonna 2017 julkaistussa tutkimuksessa [36] suoritetussa kyselyssä SVN on hieman suositumpi kuin Git. Tutkimuksessa kuitenkin huomioidaan, että vastausten perusteella Git olisi valtaamassa alaa. Vuonna 2017 suoritetussa Stack Oveflown kehittäjä kyselyssä [37] Git on kuitenkin selvässä johtoasemassa. Nämä tulokset ovat Kuvassa 5.



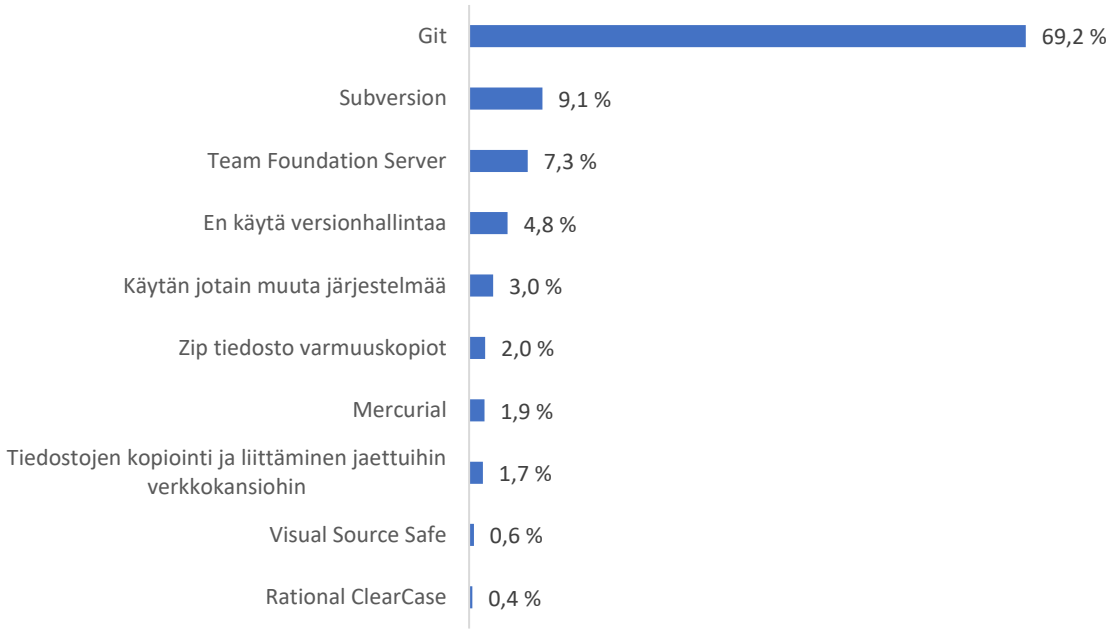
Kuva 2. Versionhallintajärjestelmien käyttö 2013 [34] (n = 42)



Kuva 3. Versionhallintajärjestelmien käyttö 2014 (n = 820)

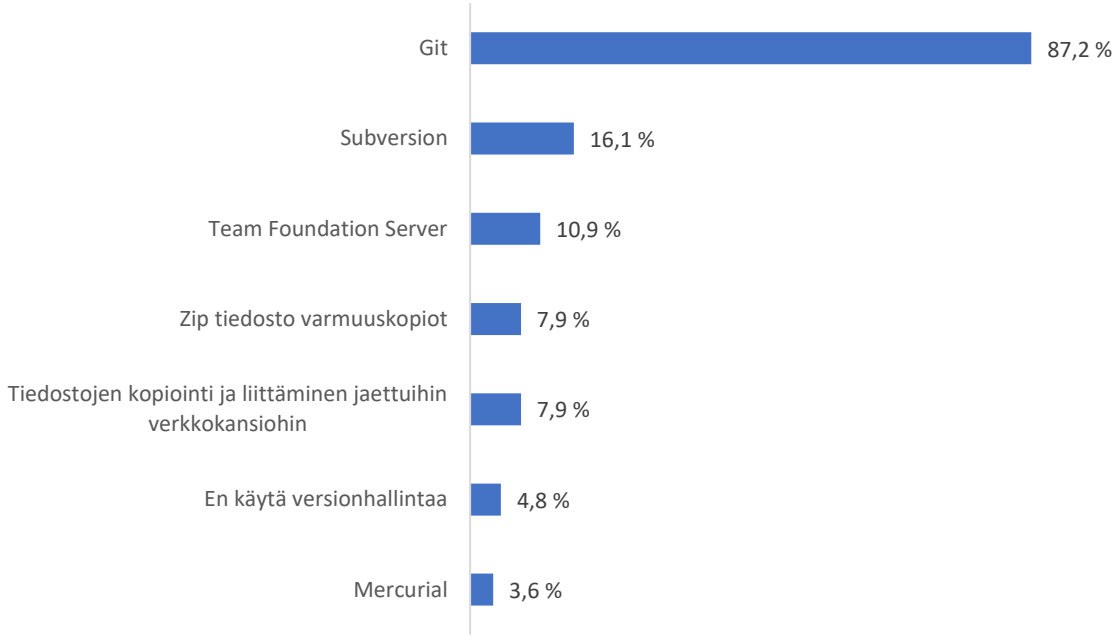


Kuva 4. Versionhallintajärjestelmien käyttö 2011-2014 [35] (vuonna 2014 n = 876)



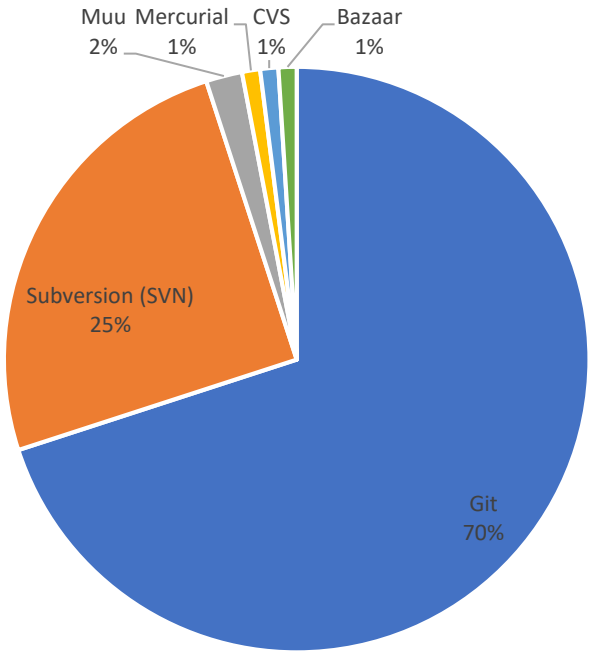
Kuva 5. Versionhallintajärjestelmien käyttö 2017 [37] (n = 30730)

Vuonna 2018 suoritetussa Stack Overflown kehittäjä kyselyssä [38] on Git edelleen todella suosittu kuten kuvasta 6 näkyy. Vuoden 2017 ja 2018 arvoja ei kuitenkaan voi suoraan verrata keskenään, koska niissä ei ole kaikkia samoja vaihtoehtoja.



Kuva 6. Versionhallintajärjestelmien käyttö 2018 [38] (n = 74298)

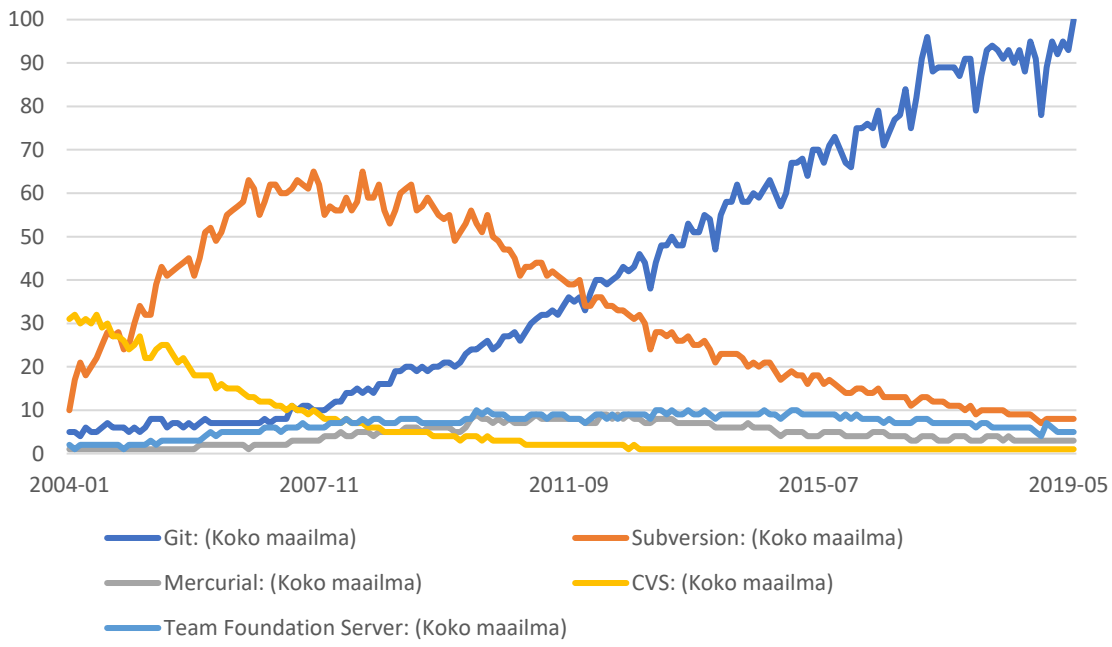
Open hub sivusto kerää tietoja avoimen lähdekoodin projekteista. Saatavissa on muun muassa tieto projektien käyttämistä versionhallintajärjestelmistä [39]. Kuvasta 7 näkee tämän jakauman.



Kuva 7. Versionhallintajärjestelmien käyttö 2019 [39]

Google Trends sivulta pystyy hakemaan tietoja siitä, kuinka paljon jotain asiaa on haettu Googlesta. Kuvassa 8 on viiden versionhallintajärjestelmän hakujen määrä vuodesta 2004 vuoteen 2019 asti. Siitä huomataan, että kiinnostus Gitiin on kasvanut koko ajan ja SVN:än suosio on laskenut siitä asti, kun Gitin suosio alkoi kasvaa.

Näistä tuloksista voidaan huomata, että Gitin suosio ohitti SVN:än noin 2013-2014 ja tämän jälkeen se on ollut huimassa kasvussa. Vain yhdessä lähteessä [36] oli poikkeavaa tietoa. Kuvasta 8 huomataan, että kiinnostus Gitiin ohitti SVN:än jo 2011-2012. Voidaan siis todeta, että Git on tällä hetkellä selvästi suosituin versionhallintajärjestelmä. Tästä seuraa kysymys, miksi Git on niin suosittu?



Kuva 8. Versionhallintajärjestelmien haku määrät [40]

3.4 Syitä Gitin suosioon

Keskitetyistä versionhallintajärjestelmistä siirtymistä hajautettuihin ei ole tutkittu paljon. Tästä aiheesta löytyi vain kaksi tieteellistä artikkelia. Alwis ja Silito [11] tutkivat neljän avoimen lähdekoodin projektin syitä siirtyä hajautettuun versionhallintajärjestelmään. Nämä neljä projektia olivat Perl, OpenOffice, NetBSD ja Python. He tutkivat projekteista julkisesti saatavilla olevia lähteitä, joissa keskusteltiin versionhallintajärjestelmän vaihdon syistä.

Muşlu et al. [9] tutkivat siirtymisen syitä, esteitä ja seuraamuksia Microsoftilla. He suorittivat haastattelun kymmenelle henkilölle ja loivat vastausten perusteella kyselyn, johon vastasi 70 henkilöä. Brindescu et al. [10] tutkivat miten hajautetut versionhallintajärjestelmät vaikuttavat ohjelmisto muutoksiin. He suorittivat kyselyn, johon vastasi 820 henkilöä. Vaikka heidän tutkimuksensa ei liitykään siirtymiseen versionhallintajärjestelmien välillä, oli yksi heidän tutkimuskysymyksistä ”Miksi kehittäjät suosivat tiettyä versionhallintajärjestelmää?”.

Alwis ja Silito [11] , Muşlu et al. [9] ja Brindescu et al. [10] kaikki havaitsivat, että työskentely ilman yhteyttä keskustietovarastoon on yksi siirtymisen syistä. Brindescu et al. [10] havaitsivat pääsyyn suosia hajautettuja versionhallintajärjestelmiä olevan jokin niille ominainen ominaisuus. Muşlu et al. [9] ja Alwis ja Silito [11] havaitsivat myös, että mahdollisuus tehdä kokeellista koodia tehokkaasti hyödyntämällä haarautumista oli yksi syy. Muşlu et al. [9] havaitsivat, että mahdollisuus vaihtaa työskentelytilaa tehokkaasti käyttämällä haaroja oli siirtymisen syy. He havaitsivat mahdollisuuden työskennellä pikkuhiljaa tehden useita committeja olevan toinen syy. Alwis ja Silito [11] havaitsivat mahdollisuuden tarjota kaikille kehittäjille yhtäläinen versionhallinnan käyttökokemus, tuen atomisille muutoksille ja yksinkertaisen automaattisen yhdistämisen olevan syitä siirtymiseen.

Tältä pohjalta voidaan sanoa, että pääsyy siirtyä hajautettuihin versionhallinjärjestelmiin, ja täten myös Gitiin, on mahdollisuus työskennellä ilman yhteyttä keskustietovarastoon. Myös hajautetun versionhallintajärjestelmän tehokkaat haarat ja niiden tuomat kehitystavat ovat tärkeitä. Yleisesti voidaan sanoa, että hajautettuja versionhallintajärjestelmiä suositaan niiden ominaisuuksien takia.

3.5 Versionhallintajärjestelmien testausta

Testattaviksi versionhallintajärjestelmiksi valittiin vain Git ja SVN, koska ne ovat selvästi suositumpia kuin muut. Ne ovat myös ilmaisia mikä mahdollistaa testauksen ilman rahallista panostusta. Testauksen kohteina olivat käyttöönoton ja käytön helppous sekä vaadittujen ominaisuuksien löytyminen. Vaadittuina ominaisuuksina olivat LUTissa käytetyn C-kielen

oppaan versionhallintaosuuden ohjeen mukaisten toimintojen suorittaminen [14]. Koska oppaassa käytetään SVN:ää on selvää, että se täyttää vaatimukset, mutta testauksessa verrataan sen helppokäyttöisyyttä Gitiin. Testaus suoritettiin sekä Windows 10 että Ubuntu 19.04 käyttöjärjestelmillä.

Kummankin versionhallintajärjestelmän asentaminen oli hyvin helppoa niin Windowsilla kuin Ubuntullakin. Ubuntulla SVN asennettiin syöttämällä komento ”sudo apt install subversion” terminaaliin. Tämän jälkeen asentaminen pitää hyväksyä ja asennuksen loputtua SVN on valmis käytettäväksi. Gitin asentamisprosessi on muuten sama, mutta komento on ”sudo apt install git”. Windowsissa molempien ohjelmien asennus on hieman monimutkaisempaa, mutta ei juuri vaikeampaa. Koska Windowsissa ei ole keskitettyä ohjelmistovarastoa kuten Ubuntussa, joutuu ohjelmat lataamaan niiden kotisivuilta [41]. Koska SVN:stä ei ole saatavilla virallista asennusohjelmaa, valittiin SVN asiakasohjelmaksi SVN:än kotisivuilta löytyvästä kolmannen osapuolen ohjelmalistasta ensimmäinen, joka ei vaatinut rekisteröitymistä eli SlikSVN. SlikSVN ladattiin sen kotisivuilta [42]. Asennusohjelma tuli pakattuna tiedostona, joka purettiin ja ohjelma asennettiin oletusasetuksilla. Asennus oli hyvin yksinkertainen eikä käyttäjän tarvitse kuin painaa jatka painiketta tarvittaessa. Gitistä on saatavissa virallinen asennusohjelma Windowsille, mikä ladattiin Gitin kotisivuilta [43]. Gitin asennus oli aivan yhtä helppo kuin SlikSVN:än ja asennettiin myös oletusasetuksilla.

Versionhallintajärjestelmien käytöstä testataan seuraavaa:

1. Uuden versionhallinta tietovaraston teko
2. Olemassa olevan tietovaraston haku
3. Tiedoston lisääminen tietovarastoon
4. Paikallisten tiedostojen päivitys
5. Tiedostomuutosten lisääminen tietovarastoon
6. Muutosten katselu

SVN:llä uusi tietovarasto luodaan ensin SVN palvelimelle ja miten tämä tehdään, riippuu siitä minkälaista SVN palvelinta käyttää. Tätä vaihetta ei siis lasketa asiakasohjelman käyttökokemukseen. Tehty tietovarasto on nyt siis olemassa, jolloin sen käyttö hoituu samalla komennolla kuin olemassa olevan tietovaraston haku. Myös Gitillä uuden

tietovaraston teko riippuu palvelinohjelmistosta. Riippuen miten tietovaraston alustaa, vaihtelee sen käyttöönotto hieman. Jos luo tyhjän tietovaraston esimerkiksi, koska käyttäjällä on jo paikallinen tietovarasto, joutuu paikalliseen tietovarastoon lisäämään palvelimella olevan tietovaraston osoitteen. Tässä tietovarasto luotiin kuitenkin niin, että siellä on valmiiksi ”lue minut” tiedosto. Tämä mahdollistaa tietovaraston käyttöönoton samalla komennolla kuin olemassa olevan tietovaraston haku tapahtuu.

Olemassa oleva tietovarasto haetaan SVN:llä komennolla ”svn checkout osoite”, jossa osoite on palvelimella olevan tietovaraston osoite. Tässä vaiheessa pitää myös tunnistautua palvelimelle asetetulla tietovaraston käyttäjätunnuksella ja salasanalla. Gitillä tietovarasto haetaan komennolla ”git clone osoite”, jossa osoite on myös palvelimella olevan tietovaraston osoite. Myös tässä pitää tunnistautua.
SVN:llä uuden tiedoston voi lisätä versionhallintaan komennolla ”svn add polku”, jossa polku on kyseisen tiedoston tiedostopolku. Tämä ei kuitenkaan vielä lähetä sitä palvelimelle, vaan vasta valmistelee sen. Lähetystä varten pitää käyttää komentoa ”svn commit -m ’viesti’”, jossa viesti on käyttäjän ilmoittama informaatio siitä mitä tehtiin. Gitillä tämä toimii pitkälti samalla tavalla, mutta kommennot alkavat ”git” ei ”svn”. Gitin rakenteen vuoksi tämä ei kuitenkaan lähetä tiedostoa vielä palvelimelle vaan paikalliseen tietovarastoon. Jotta se saataisiin myös palvelimelle pitää käyttää komentoa ”git push”. Gitillä joutuu siis tekemään yhden vaiheen lisää.

Paikalliset tiedostot saa SVN:llä helposti päivitettyä komennolla ”svn update”. Gitillä vastaava komento on ”git pull”. Tässä tulee ilmi, onko paikallisella versiolla ja palvelimen versiolla ristiriitoja, jotka pitäisi ratkoa. Sitä ei kuitenkaan testattu, koska LUTissa käytetyssä C-kielen oppaassa ei ole myöskään ristiriitojen ratkaisua, joten se ei ole vaatimus.

Tiedoston muutokset saa SVN:llä lähetetty suoraan komennolla ”svn commit -m ’viesti’”. Gitillä suoritetaan samat toimet kuin uuden tiedoston lisäämisessä, koska ensin valitaan mitkä muutokset halutaan lähettää komennolla ”git add polku”. SVN puolestaan lähettää automaattisesti kaikki muutokset. Ennen muutosten lähettämistä pystyy niitä myös katselemaan komennoilla ”svn diff” ja ”git diff”. Jos haluaa vain yleiskuvan siitä mitä tiedostoja on muutettu, onnistuu se komennoilla ”svn status” ja ”git status”.

Gitin ja SVN:än käyttö näissä perustoiminnoissa on siis melkein yhtä helppoa. Gitissä on ylimääräisiä vaiheita johtuen sen rakenteesta, mutta se ei tee käytöstä vaikeampaa.

3.6 Web-käyttöliittymät

Jotta yhteistyö Gittiä käytettäessä olisi helpompaa ja etätietovarastojen hallinta ja käyttö olisi helpompaa, on kehitetty web-käyttöliittymiä. Näitä käyttöliittymiä on monia kuten GitHub, GitLab, Bitbucket, Azure DevOps Services, jne. Gitin mukana tulee myös sen oma web-käyttöliittymä gitweb, mutta tämä ei tarjoa käyttäjien hallintaa tai muita edistyneempiä ominaisuuksia vaan mahdollistaa vain tietovarastojen tutkimisen [44]. Tutkittaviksi valittiin kolme käyttöliittymää: GitHub, GitLab ja Azure DevOps Services. GitHub valittiin, koska se on hyvin suosittu Git web-käyttöliittymä yli 36 miljoonalla käyttäjällä ja 100 miljoonalla tietovarastolla [45]. GitLab valittiin, koska kaikissa Suomen yliopistoissa, joista löytyi tietoa Gitin käytöstä, käytetään sitä. Microsoftin Azure DevOps valittiin, koska LUTilla on käytössä Microsoftin käyttäjätunnukset ja täten se on ainoa web-käyttöliittymä, johon pääsee kirjautumaan suoraan LUTin käyttäjätunnuksilla.

GitHub on ilmainen yksityisille käyttäjille ja siitä on myös maksullinen versio lisäominaisuuksilla. Tiimeille on vielä hieman kalliimpi versio ja jos GitHubin haluaa itse omille palvelimille itseylläpidettäväksi pitää hinta erikseen neuvotella.[46] Oppilaitoksille on kuitenkin saatavilla ilmainen lisenssi sekä tiimi että itseylläpidettävään versioon, mutta sitä voi käyttää ainoastaan opetukseen ja ei-kaupalliseen tutkimustyöhön [47]. Lisäksi jotta voidaan käyttää LUTin käyttäjätunnuksia pitäisi käyttää itseylläpidettyä versiota, joka siis rajoittaa käyttötarkoituksia tai josta joutuu maksamaan erikseen neuvoteltavan hinnan. GitHubissa tietovarastot voidaan organisoida organisaation alle [48]. Tämä mahdollistaisi itseylläpidetyn version avulla esimerkiksi rakenteen, jossa jokaiselle LUTin kurssille tai muulle Gitiä käyttävälle kokonaisuudelle olisi oma organisaatio, jossa olisi opiskelijoille tietovarastoja tehtävien tekoon. Ilmaiskäyttäjillä taas on rajoituksena yksityisissä tietovarastoissa yhteistyöntekijät, jotka on rajattu vain kolmeen [46]. GitHubissa itsessään ei ole jatkuvaa integraatiota, mutta siihen saa integroitua monia ulkoisia jatkuvan integraation palveluita [49].

Azure DevOps Services on myös ilmainen yksityisille käyttäjille ja tätä ominaisuutta pystyy suoraan käyttämään LUTin käyttäjätunnuksilla [50]. Yhteistyöntekijöiden määrä on kuitenkin rajattu viiteen per organisaatio [51], joka sisältää projekteja, jotka sisältävät tietovarastoja. Tämä hankaloittaa yhteistyön tekemistä, koska käyttäjärajoite ei ole per tietovarasto. Lisää yhteistyöntekijöitä saa kuitenkin lisää maksamalla noin viisi euroa per käyttäjä [50]. Koska Azure DevOpsiin pääsee suoraan kirjautumaan LUTin käyttäjätunnuksilla ja se on Microsoftin ylläpitämä, ei siitä koituisi käyttäjien lisäämisen ja ohjelmiston päivittämisen osalta ylläpitotoimia LUTin tietohallinnolle. Tietohallinto joutuisi ainoastaan lisäämään kursseille projekteja, jollei luennoitsijoille anneta tähän oikeuksia. Azure DevOpsissa on kolme tasoa: organisaatio, projekti ja tietovarasto. Jos LUTilla olisi käyttäjä, jolla on organisaatio, johon on ostettu käyttäjiä koko yliopiston tarpeisiin, voitaisiin jokaiselle kurssille luoda oma projekti, johon voidaan luoda opiskelijoille tietovarastoja tehtävien tekoon. Tämä mahdollistaisi helpon ryhmätyöskentelyn. Mahdollisena ongelmana on kuitenkin, että tietovarastojen työtehtävät (work items) näkyvät kaikille, vaikka itse tietovarasto olisi asetettu näkyväksi vain joillekin. Käyttöoikeuksien hallinta on myös hieman monimutkaista. Azure DevOpsissa on jatkuvan integraation tuki [52].

GitLab on myös ilmainen yksityisille käyttäjille ja siitä on saatavilla avoimen lähdekoodin ilmainen itseylläpidettävä versio MIT lisenssillä [53]. Kummassakaan ei ole rajoituksia yhteistyötekijöiden suhteen [54]. Oppilaitoksille on myös olemassa ilmainen lisenssi, jolla saa kaikki ominaisuudet käyttöön sekä GitLabin ylläpitämässä että itseylläpidetyssä versiossa. Tätä lisenssiä saa kuitenkin käyttää vain opetuskäyttöön ja ei-kaupalliseen tutkimukseen.[55] Jotta GitLabiin voidaan kirjautua LUTin käyttäjätunnuksilla, on käytettävä itseylläpidettyä versiota, johon avoimen lähdekoodin versio on riittävä. GitLabista julkaistaan uusi versio kerran kuukaudessa [56]. Tietohallinnon ylläpitotoimia olisivat siis GitLabin päivittäminen kerran kuukaudessa ja tietovarasto rakenteen alustaminen. GitLabissa on mahdollista tehdä ryhmiä ja niille aliryhmiä kaksikymmentä tasoa [57], [58]. Tämä mahdollistaa käyttäjäoikeuksien hierarkkisen jaottelun. Rakenteena voisi olla esimerkiksi ryhmät koulutusohjelmille, joissa on ryhmät kursseille, joissa on ryhmät toteutuskerroille. Toteutuskerroissa olisi opiskelijoille joko ryhmät, joissa on tietovarastoja tai tietovarastot, riippuen halutaanko monta tietovarastoa per opiskelija. Tällä rakenteella tietohallinnon ei välttämättä tarvitse lisätä kursseja tai tietovarastoja, jos

aliryhmille on asetettu koulutusohjelmien henkilökuntaa tarvittavilla oikeuksilla. GitLabissa on jatkuvan integraation tuki [59].

Kaikissa näissä web-käyttöliittymissä on kuitenkin ongelmana suurien käyttäjämäärien lisääminen kursseille. Tätä varten on kehitetty monia työkaluja [60]–[64]. Esimerkiksi Tampereen yliopistolla on käytössä heidän itse kehittämä Repolainen, jolla voi hallita kursseja GitLabissa [60]. Tässä työssä ei kuitenkaan tutkita tarkemmin tätä aihetta, koska se ei suoraan liity versionhallintaa.

Luottamuksellisen datan käsittely on myös yksi ongelma. Tätä varten täytyy käytännössä käyttää itse ylläpidettyä web-käyttöliittymää tai olla erillinen sopimus palveluntarjoajan kanssa [65].

Näistä web-käyttöliittymistä paras LUTin käyttöön olisi GitLab, koska se on ilmainen eikä aseta rajoituksia käytölle. Siinä ei myöskään ole rajoituksia yhteistyöntekijöille tietovarastoissa ja se tukee jatkuvaa integraatiota. Tietovarastojen organisointi on myös joustavaa. Vaikka se on itseylläpidettävä ei siitä koidu käyttöönoton jälkeen välttämättä paljoa ylläpitotoimia tietohallinnolle ja koska se on itse ylläpidetty, onnistuu sillä luottamuksellisen datan käsittely. GitLabia myös käytetään muissa yliopistoissa, joten voidaan olettaa sen olevan opetukseen soveltuva ohjelmisto. GitHubin ilmainen opetus versio asettaa käytölle rajoituksia ja maksullisen version hinnasta ei ole tietoa. Lisäksi tietovarastojen organisointi ei ole kovin joustavaa. Azure DevOpsissa olisi etuna käyttöönoton helppous, mutta se maksaa noin viisi euroa per käyttäjä. Lisäksi tietovarastojen organisointi ei ole yhtä joustavaa kuin GitLabissa.

4 TULOKSET

Tutkimuksessa selvisi, että Gitin suosio on kasvanut nopeasti ja tällä hetkellä se on ylivoimaisesti suosituin versionhallintajärjestelmä. Sitä käytetään myös monissa Suomen yliopistoissa. Suurin syy Gitin suosioon on tutkimuksen mukaan sen ominaisuudet ja varsinkin mahdollisuus työskennellä ilman verkkoyhteyttä. Versionhallintajärjestelmien testauksessa huomattiin, että sen käyttöönotto ja käyttö ovat suurin piirtein yhtä helppoja kuin SVN:llä. Muissa tutkimuksissa on havaittu, että Gitin tai yleisesti hajautettujen versionhallintajärjestelmien opettelu on vaikeampaa, mutta tässä tutkimuksessa testatuilla hyvin yksinkertaisilla vaatimuksilla ei havaittu eroa SVN:ään. Git myös vaatisi tietohallinnolta vähemmän ylläpitoa kuin SVN. Ylläpidon määrä kuitenkin vaihtelee riippuen minkä web-käyttöliittymän Gitille valitsee. LUTin ylläpitämä GitLab palvelin olisi paras vaihtoehto, koska GitLab on ilmainen MIT-lisenssillä eli sillä saa tehdä mitä tahansa. Sen päivittäminenkin kerran kuukaudessa on helppoa. LUTin ylläpitämänä se myös mahdollistaa luottamuksellisen datan käsittelyn. GitLab on käytössä muissa Suomen yliopistoissa, joten voidaan olettaa, että se toimii opetuksessa. GitLabia varten on kehitetty työkaluja, jotka helpottavat kurssien hallintaa ja ylläpitoa, kun kurssilla on paljon opiskelijoita. Voidaan siis todeta, että LUTissa kannattaisi käyttää Gitiä ja GitLabia.

Koska tutkimuksessa havaittiin Git ja GitLab parhaiksi työkaluiksi versionhallintaan, tehtiin niille opetusmateriaali ja harjoitustehtäviä. Opetusmateriaalia varten tutkittiin, minkälaisia ohjeita muista yliopistoista on saatavilla. Tutkittiin myös Version Control with Git [66] ja Pro Git [6] kirjoja. Havaittiin, että Gitin virallinen ohjekirja Pro Git sisältää kattavasti kaiken tarvittavan ja näin ollen sitä käytettiin pohjana opetusmateriaalin teossa. Se on myös saatavilla ilmaiseksi Gitin sivuilta, joten opiskelijat voivat tutustua siihen opiskellakseen Gitin käyttöä syvällisemmin. GitLabin ohjeiden tekemisen materiaalina käytettiin GitLabin dokumentaatiota [67].

5 POHDINTA JA TULEVAISUUS

Osa lähteistä, joissa on versionhallintajärjestelmien suosiota osoittavia lukuja, on hyvin rajoittuneesta ryhmästä kuten esimerkiksi vain Microsoftin työntekijöistä. Tämä voisi johtaa tulosten vääristymiseen, mutta koska eri lähteistä saadaan hyvin samanlaisia tuloksia, voidaan niiden olettaa olevan päteviä. Suomen yliopistoissa käytettyjä versionhallintajärjestelmiä tutkittiin vain etsimällä tietoa hakukoneella ja yliopistojen sivuilta. Paremmin tietoa olisi voinut ehkä saada ottamalla yhteyttä yliopistojen henkilökuntaan. Versionhallintajärjestelmien suosion syitä ei ole tutkittu paljon, mutta tutkimuksissa havaitut syyt ovat hyvin vakuuttavia. Versionhallintajärjestelmien testauksessa ei havaittu suuria eroja versionhallintajärjestelmien välillä, mutta tämä johtuu pitkälti testien yksinkertaisuudesta. Jos olisi testattu monimutkaisempia asioita, eroja olisi varmasti syntynyt, mutta nämä asiat menisivät yli vaaditun kurssisisällön. Web-käyttöliittymistä ei etsitty tutkimuksia, koska niissä vaikuttaa enemmän LUTin tarpeet kuin mitä muualla käytetään. Versionhallintajärjestelmistä tutkimuksia etsiessä kuitenkin huomattiin GitHubista löytyvän paljon tutkimuksia todennäköisesti sen vuoksi, että se on hyvin suosittu. Tässä tutkimuksessa ei tutkittu miten suurien opiskelijamäärien hallinnointi versionhallintajärjestelmässä tapahtuisi. Huomioitiin ainoastaan, että se on mahdollista. Tästä aiheesta olisi mahdollista tehdä jatkotutkimus, jossa tutkittaisiin, voitaisiinko LUTissa käyttää jotain jo olemassa olevaa järjestelmää hallinnointitehtävien automatisointiin vai olisiko mahdollisesti järkevintä tehdä LUTille oma järjestelmä.

6 YHTEENVETO

Tässä työssä tutkittiin, mikä versionhallintajärjestelmä olisi paras opetuskäyttöön LUTissa. Versionhallintajärjestelmiä huomattiin olevan paljon erilaisia ja havaittiin, että muissa Suomen yliopistoissa on käytössä Git ja GitLab. Versionhallintajärjestelmien suosiosta maailmalla yrityksissä ja avoimen lähdekoodin projekteissa huomattiin, että Gitin suosio on kasvanut nopeasti ja tällä hetkellä se on selvästi suosituin versionhallintajärjestelmä. Sitä ennen SVN oli suosituin. SVN:ää ja Gitiä testatessa ei huomattu suuria eroja. Todettiin, että Git on opetuskäyttöön sopivin, koska se on yleisin eikä sillä ole merkittäviä haittapuolia. Gitille web-käyttöliittymäksi todettiin itse ylläpidetyn GitLabin olevan paras, koska se mahdollistaa mukautuvimman käytön ja on ilmainen. Gitille ja GitLabille tehtiin tiivis oppimateriaali niiden perusteista.

LÄHTEET

[1] Spinellis, D., ”Version control systems”, *IEEE Softw.*, vol. 22, nro 5, s. 108–109, 2005.

[2] Rochkind, M.J., ”The source code control system”, *IEEE Trans. Softw. Eng.*, vol. SE-1, nro 4, s. 364–370, 1975.

[3] Tichy, W.F., ”Design, Implementation, and Evaluation of a Revision Control System”, *Proceedings of the 6th International Conference on Software Engineering*, Los Alamitos, CA, USA, 1982, s. 58–67 [Verkossa]. Saatavissa: http://dl.acm.org/citation.cfm?id=800254.807748. [Viitattu: 31.07.2019]

[4] ”CVS--Concurrent Versions System v1.12.12: 1. Overview”, 15-huhti-2012. [Verkossa]. Saatavissa: https://web.archive.org/web/20120415051926/http://ximbiot.com/cvs/manual/cvs-1.12.12/cvs_1.html. [Viitattu: 31.07.2019]

[5] ”subversion 1.0 is released [LWN.net]”, Saatavissa: https://lwn.net/Articles/72498/. [Viitattu: 31.07.2019]

[6] ”Git - Book”, Saatavissa: https://git-scm.com/book/en/v2. [Viitattu: 01.07.2019]

[7] ”Comparison of version-control software”, *Wikipedia*. 18-kesä-2019 [Verkossa]. Saatavissa: https://en.wikipedia.org/w/index.php?title=Comparison_of_version-control_software&oldid=902443527. [Viitattu: 20.06.2019]

[8] Rao, N.R. ja Sekharaiah, K.C., ”A Methodological Review Based Version Control System with Evolutionary Research for Software Processes”, *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, New York, NY, USA, 2016, s. 14:1–14:6 [Verkossa]. Saatavissa: http://doi.acm.org/10.1145/2905055.2905072. [Viitattu: 20.06.2019]

[9] Muşlu, K., Bird, C., Nagappan, N., ja Czerwonka, J., ”Transition from Centralized to Decentralized Version Control Systems: A Case Study on Reasons, Barriers, and Outcomes”, *Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA, 2014, s. 334–344 [Verkossa]. Saatavissa: http://doi.acm.org/10.1145/2568225.2568284. [Viitattu: 20.06.2019]

[10] Brindescu, C., Codoban, M., Shmarkatiuk, S., ja Dig, D., ”How Do Centralized and Distributed Version Control Systems Impact Software Changes?”, *Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA, 2014, s. 322–333 [Verkossa]. Saatavissa: http://doi.acm.org/10.1145/2568225.2568322. [Viitattu: 20.06.2019]

[11] Alwis, B. de ja Sillito, J., ”Why are software projects moving from centralized to decentralized version control systems?”, *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, 2009, s. 36–39.

[12] Atlassian, ”Git LFS - large file storage | Atlassian Git Tutorial”, *Atlassian*. [Verkossa]. Saatavissa: https://www.atlassian.com/git/tutorials/git-lfs. [Viitattu: 31.07.2019]

[13] ”Opasraportti - Tekniikan kandidaatti Tietotekniikka - LUT School of Business and Management (23E1)”, 2018 [Verkossa]. Saatavissa: https://uni.lut.fi/fi/c/document_library/get_file?uuid=5834ab02-81a9-4810-bed4-c95ace9f8b74&groupId=10304. [Viitattu: 24.06.2019]

[14] Kasurinen, J. ja Uolevi, N., *C-kieli ja käytännön ohjelmointi osa1, versio 2*. LUT Scientific and Expertise Publications, 2013 [Verkossa]. Saatavissa: https://docplayer.fi/8593797-C-kieli-ja-kaytannon-ohjelmointi-osa-1-versio-2-lappeenrannan-teknillinen-yliopisto-2013-jussi-kasurinen-uolevi-nikula.html
[15] ”GitLab Enterprise Edition - LUT”, *GitLab*. [Verkossa]. Saatavissa: https://git.it.lut.fi/users/sign_in. [Viitattu: 24.06.2019]
[16] ”Luennot – TIEA2.1B”, 30-maalis-2019 [Verkossa]. Saatavissa: https://coursepages.uta.fi/tiea2-1b/kevat-2019/luennot/. [Viitattu: 23.06.2019]
[17] ”TUNI Course Gitlab”, *GitLab*. [Verkossa]. Saatavissa: https://course-gitlab.tuni.fi/users/sign_in. [Viitattu: 23.06.2019]
[18] Pirkkanen, K., ”Musiikkiteknologia” [Verkossa]. Saatavissa: http://www.uniarts.fi/sites/default/files/Musiikkiteknologia_5.pdf
[19] ”Ohjelmisto tutkimuksen tuotoksena | Aalto-yliopisto”, Saatavissa: https://www.aalto.fi/fi/palvelut/ohjelmisto-tutkimuksen-tuotoksena. [Viitattu: 23.06.2019]
[20] ”Aalto Version Control System”, Saatavissa: https://version.aalto.fi/. [Viitattu: 23.06.2019]
[21] ”Bioinformatics Center Gitlab”, Saatavissa: https://bioinformatics.uef.fi/gitlab/users/sign_in. [Viitattu: 23.06.2019]
[22] Hedberg, H., ”Hajautettu versionhallinta Gitillä”, 2013 [Verkossa]. Saatavissa: https://noppa.oulu.fi/noppa/kurssi/811346a/luennot/811346A_git_versionhallinta.pdf
[23] ”Gitlab - University of Jyväskylä”, *GitLab*. [Verkossa]. Saatavissa: https://gitlab.jyu.fi/users/sign_in. [Viitattu: 23.06.2019]
[24] ”YouSource”, Saatavissa: https://yousource.it.jyu.fi/. [Viitattu: 23.06.2019]
[25] ”JYU Trac Help”, Saatavissa: https://trac.cc.jyu.fi/projects/help. [Viitattu: 23.06.2019]
[26] ”Gitlab for UTU”, *GitLab*. [Verkossa]. Saatavissa: https://gitlab.utu.fi/users/sign_in. [Viitattu: 23.06.2019]
[27] ”Åbo Akademi University - Gitlab”, *GitLab*. [Verkossa]. Saatavissa: https://gitlab.abo.fi/users/sign_in. [Viitattu: 23.06.2019]
[28] ”[svnfrontend] Login”, Saatavissa: https://www.cs.abo.fi/svnfrontend/admin_login.php. [Viitattu: 23.06.2019]
[29] Maisala, S., ”Versionhallinta on tutkimuksen välttämätön työkalu – Gitlab tukee tutkijan työtä Helsingin yliopistossa”, *Think Open*. 09-tammi-2019 [Verkossa]. Saatavissa: https://blogs.helsinki.fi/thinkopen/versionhallinta-on-valttamaton-tyokalu-tutkimukselle/. [Viitattu: 23.06.2019]
[30] ”University of Helsinki Version”, *GitLab*. [Verkossa]. Saatavissa: https://version.helsinki.fi/users/sign_in. [Viitattu: 23.06.2019]
[31] ”GitLab.org / GitLab Community Edition”, *GitLab*. [Verkossa]. Saatavissa: https://gitlab.com/gitlab-org/gitlab-ce. [Viitattu: 23.06.2019]
[32] ”GitLab for education”, *GitLab*. [Verkossa]. Saatavissa: https://about.gitlab.com/solutions/education/. [Viitattu: 23.06.2019]
[33] Phillips, S., Sillito, J., ja Walker, R., ”Branching and Merging: An Investigation into Current Version Control Practices”, *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, New York, NY, USA, 2011, s. 9–15 [Verkossa]. Saatavissa: http://doi.acm.org/10.1145/1984642.1984645. [Viitattu: 20.06.2019]

[34] Matos, A., de Leon, M.P., Ferreira, R., ja Barraca, J.P., ”An Open Source Software Forge for European Projects”, *Proceedings of the Workshop on Open Source and Design of Communication*, New York, NY, USA, 2013, s. 41–45 [Verkossa]. Saatavissa: http://doi.acm.org/10.1145/2503848.2503857. [Viitattu: 20.06.2019]
[35] Ian Skerrett, ”Eclipse community survey 2014 v2”, 11:53:33 UTC [Verkossa]. Saatavissa: https://www.slideshare.net/IanSkerrett/eclipse-community-survey-2014. [Viitattu: 18.06.2019]
[36] Raunak, M.S. ja Binkley, D., ”Agile and other trends in software engineering”, *2017 IEEE 28th Annual Software Technology Conference (STC)*, 2017, s. 1–7.
[37] ”Stack Overflow Developer Survey 2017”, *Stack Overflow*. [Verkossa]. Saatavissa: https://insights.stackoverflow.com/survey/2017/?utm_source=so-owned&utm_medium=social&utm_campaign=dev-survey-2017&utm_content=social-share. [Viitattu: 18.06.2019]
[38] ”Stack Overflow Developer Survey 2018”, *Stack Overflow*. [Verkossa]. Saatavissa: https://insights.stackoverflow.com/survey/2018/?utm_source=so-owned&utm_medium=social&utm_campaign=dev-survey-2018&utm_content=social-share. [Viitattu: 18.06.2019]
[39] ”Compare Repositories - Open Hub”, Saatavissa: https://www.openhub.net/repositories/compare. [Viitattu: 27.06.2019]
[40] ”Google Trends”, *Google Trends*. [Verkossa]. Saatavissa: https://trends.google.fi/trends/explore?date=all&q=%2Fm%2F05vqwg,%2Fm%2F012ct9,%2Fm%2F08441_,%2Fm%2F09d6g,%2Fm%2F02rvgkm. [Viitattu: 27.06.2019]
[41] ”Apache Subversion Binary Packages”, Saatavissa: https://subversion.apache.org/packages.html. [Viitattu: 24.06.2019]
[42] ”Download SlikSVN Subversion client”, Saatavissa: https://sliksvn.com/download/. [Viitattu: 24.06.2019]
[43] ”Git - Downloads”, Saatavissa: https://git-scm.com/downloads. [Viitattu: 24.06.2019]
[44] ”Git - gitweb Documentation”, Saatavissa: https://git-scm.com/docs/gitweb. [Viitattu: 31.07.2019]
[45] ”Build software better, together”, *GitHub*. [Verkossa]. Saatavissa: https://github.com. [Viitattu: 31.07.2019]
[46] ”Pricing · Plans for every developer”, *GitHub*. [Verkossa]. Saatavissa: https://github.com/pricing. [Viitattu: 31.07.2019]
[47] ”Github Educational Use Agreement”, *GitHub Education*. [Verkossa]. Saatavissa: http://education.github.com/partners/schools/terms. [Viitattu: 02.07.2019]
[48] ”About organizations - GitHub Help”, Saatavissa: https://help.github.com/en/articles/about-organizations. [Viitattu: 31.07.2019]
[49] ”GitHub welcomes all CI tools”, *The GitHub Blog*. 08-marras-2017 [Verkossa]. Saatavissa: https://github.blog/2017-11-07-github-welcomes-all-ci-tools/. [Viitattu: 31.07.2019]
[50] ”Azure DevOps Services | Microsoft Azure”, Saatavissa: https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/. [Viitattu: 31.07.2019]
[51] chcomley, ”Add new users to your organization or project - Azure DevOps Services”. [Verkossa]. Saatavissa: https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/add-organization-users. [Viitattu: 31.07.2019]

[52] CamSoper, ”Continuous integration and deployment - DevOps with ASP.NET Core and Azure”. [Verkossa]. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/azure/devops/cicd. [Viitattu: 31.07.2019]
[53] ”LICENSE · master · GitLab.org / GitLab Community Edition”, *GitLab*. [Verkossa]. Saatavissa: https://gitlab.com/gitlab-org/gitlab-ce/blob/master/LICENSE. [Viitattu: 31.07.2019]
[54] ”GitLab Pricing”, *GitLab*. [Verkossa]. Saatavissa: https://about.gitlab.com/pricing/. [Viitattu: 31.07.2019]
[55] ”GitlabTerms of Use”, *GitLab*. [Verkossa]. Saatavissa: https://about.gitlab.com/terms/#edu-oss. [Viitattu: 02.07.2019]
[56] ”GitLab releases”, *GitLab*. [Verkossa]. Saatavissa: https://about.gitlab.com/releases/. [Viitattu: 31.07.2019]
[57] ”Groups | GitLab”, Saatavissa: https://docs.gitlab.com/ee/user/group/. [Viitattu: 31.07.2019]
[58] ”Subgroups | GitLab”, Saatavissa: https://docs.gitlab.com/ee/user/group/subgroups/. [Viitattu: 31.07.2019]
[59] ”GitLab Continuous Integration & Delivery”, *GitLab*. [Verkossa]. Saatavissa: https://about.gitlab.com/product/continuous-integration/. [Viitattu: 31.07.2019]
[60] Venttola, J.-P., ”Versionhallinta opetuskäytössä”, 2018 [Verkossa]. Saatavissa: http://dspace.cc.tut.fi/dpub/handle/123456789/25749. [Viitattu: 20.06.2019]
[61] Glassey, R., ”Adopting Git/Github Within Teaching: A Survey of Tool Support”, *Proceedings of the ACM Conference on Global Computing Education*, New York, NY, USA, 2019, s. 143–149 [Verkossa]. Saatavissa: http://doi.acm.org/10.1145/3300115.3309518. [Viitattu: 18.06.2019]
[62] Chen, H., Chen, W., Hsueh, N., Lee, C., ja Li, C., ”ProgEdu - an automatic assessment platform for programming courses”, *2017 International Conference on Applied System Innovation (ICASI)*, 2017, s. 173–176.
[63] Ohtsuki, M., Ohta, K., ja Kakeshita, T., ”Software Engineer Education Support System ALECSS Utilizing DevOps Tools”, *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, New York, NY, USA, 2016, s. 209–213 [Verkossa]. Saatavissa: http://doi.acm.org/10.1145/3011141.3011200. [Viitattu: 19.06.2019]
[64] Heckman, S. ja King, J., ”Developing Software Engineering Skills Using Real Tools for Automated Grading”, *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, New York, NY, USA, 2018, s. 794–799 [Verkossa]. Saatavissa: http://doi.acm.org/10.1145/3159450.3159595. [Viitattu: 19.06.2019]
[65] ”LUT_tietoaineistojen-kasittelyohje.pdf”, 08-marras-2018 [Verkossa]. Saatavissa: https://intranet.lut.fi/expertandsupportservices/tietohallinto/tietoturva/Documents/LUT_tietoaineistojen-kasittelyohje.pdf. [Viitattu: 02.07.2019]
[66] Loeliger, J., *Version Control with Git*. 1005 Gravenstein Highway North, Sebastopol, CA 95472.: O’Reilley Media, Inc., 2009.
[67] ”GitLab Documentation | GitLab”, Saatavissa: https://docs.gitlab.com/ee/README.html. [Viitattu: 31.07.2019]

LIITE 1. Oppimateriaali

Lappeenrannan–Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma

Git ja GitLab oppimateriaali

Mikko Mustonen

(jatkuu)

LIITE 1. (jatkoa)

SISÄLLYSLUETTELO

(jatkuu)

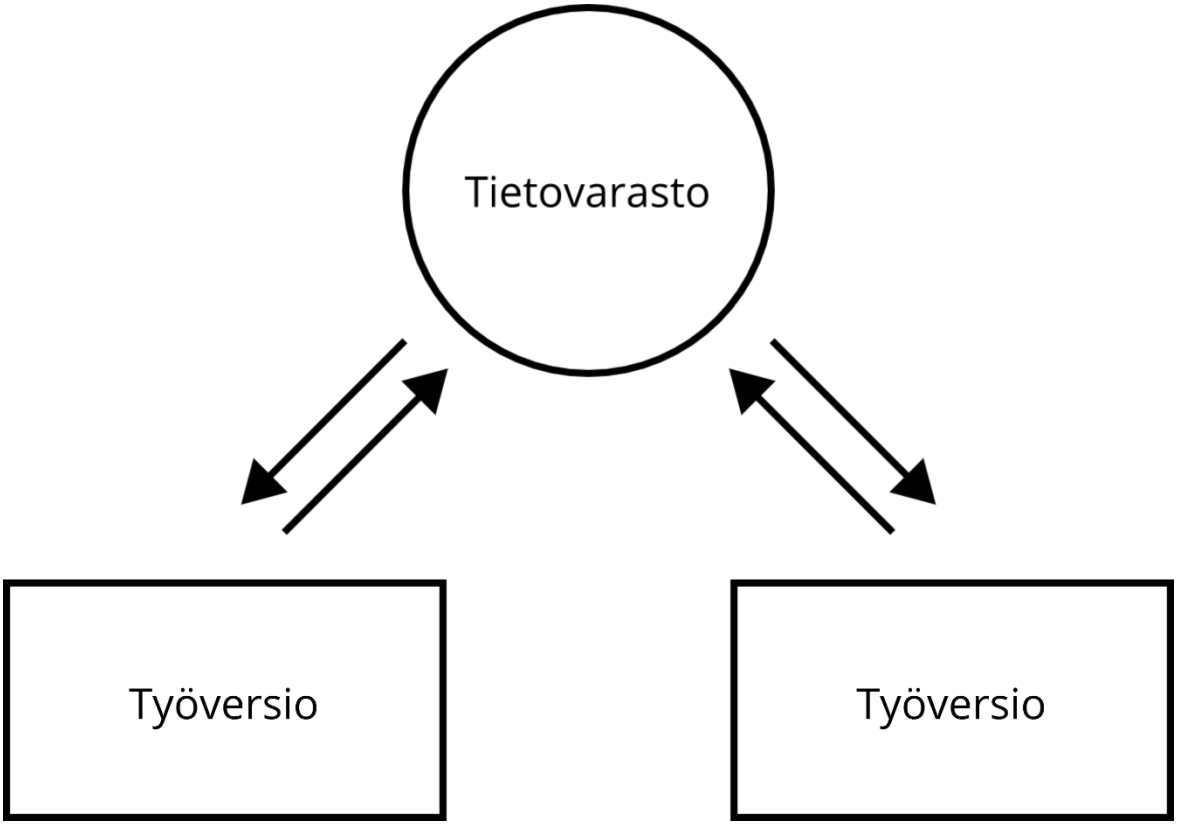
1 JOHDANTO

Versionhallinta on tärkeä osa ohjelmistokehitystä. Koska ohjelmia kehittäessä koodi muuttuu koko ajan, on sen hallintaan kehitetty versionhallintajärjestelmiä. Ensimmäinen versionhallintajärjestelmä SCCS (Source Code Control System) kehitettiin 1970-luvun alussa. Muita tunnettuja versionhallintajärjestelmiä ovat vuonna 1982 julkaistu RCS (Revision Control Sytem), vuonna 1986 julkaistu CVS (Concurrent Versions System), vuonna 2004 julkaistu SVN (Apache Subversion) ja vuonna 2005 julkaistu Git. Näiden lisäksi on kehitetty monia muita versionhallintajärjestelmiä. Versionhallintajärjestelmien tehtävänä on tehdä muutosten tekemisestä ja hallinnasta helpompaa varsinkin tiimityöskentelyssä. Versionhallintajärjestelmä pitää kirjaa kaikista tiedostomuutoksista ja muutosten tekijöistä. Tämä mahdollistaa aiempaan versioon palamisen ja muutosten tarkastelun. Versionhallintajärjestelmä takaa muutosten hallitun yhdistämisen, vaikka useampi henkilö olisi muokannut samaa tiedostoa samaan aikaan. Lisäksi se helpottaa eri versioiden välillä liikkumista haarojen avulla. Versionhallintajärjestelmiä on kahta tyyppiä, keskitettyjä ja hajautettuja. Nykyään käytetään pääasiassa hajautettuja versionhallintajärjestelmiä, mutta molemmilla on hyvät ja huonot puolensa.

1.1 Keskitetyt versionhallintajärjestelmät

Keskitetyt versionhallintajärjestelmät, kuten SVN ja CVS, tallentavat tiedostojen versiohistorian vain yhteen paikkaan palvelimelle. Tämän takia niitä kutsutaan keskitetyiksi. Kuvasta 1 näkee miten käyttäjien työversiot ovat suorassa yhteydessä palvelimen tietovarastoon. Versionhallintajärjestelmän käyttäjällä on vain yksi versio kerrallaan työhakemistossa. Tämän hyvänä puolena on binääritiedostojen käsittely. Koska käyttäjällä on vain yksi versio kerrallaan käsiteltävänä, ei haittaa, vaikka versionhallinnassa olisi monia versioita samasta binääritiedostosta, mikä saattaa viedä paljon tilaa. Jos käyttäjä kuitenkin haluaa tarkastella jotain toista versiota, tallentaa muutokset versiohistoriaan tai tehdä mitä tahansa muita versionhallinta toimia, joutuu hän pyytämään niitä palvelimelta. Tämän takia keskitettyjä versionhallintajärjestelmiä ei voi käyttää ilman yhteyttä versionhallinta palvelimeen ja yhteys palvelimeen on pullonkaulana versionhallintatoimenpiteille.

(jatkuu)



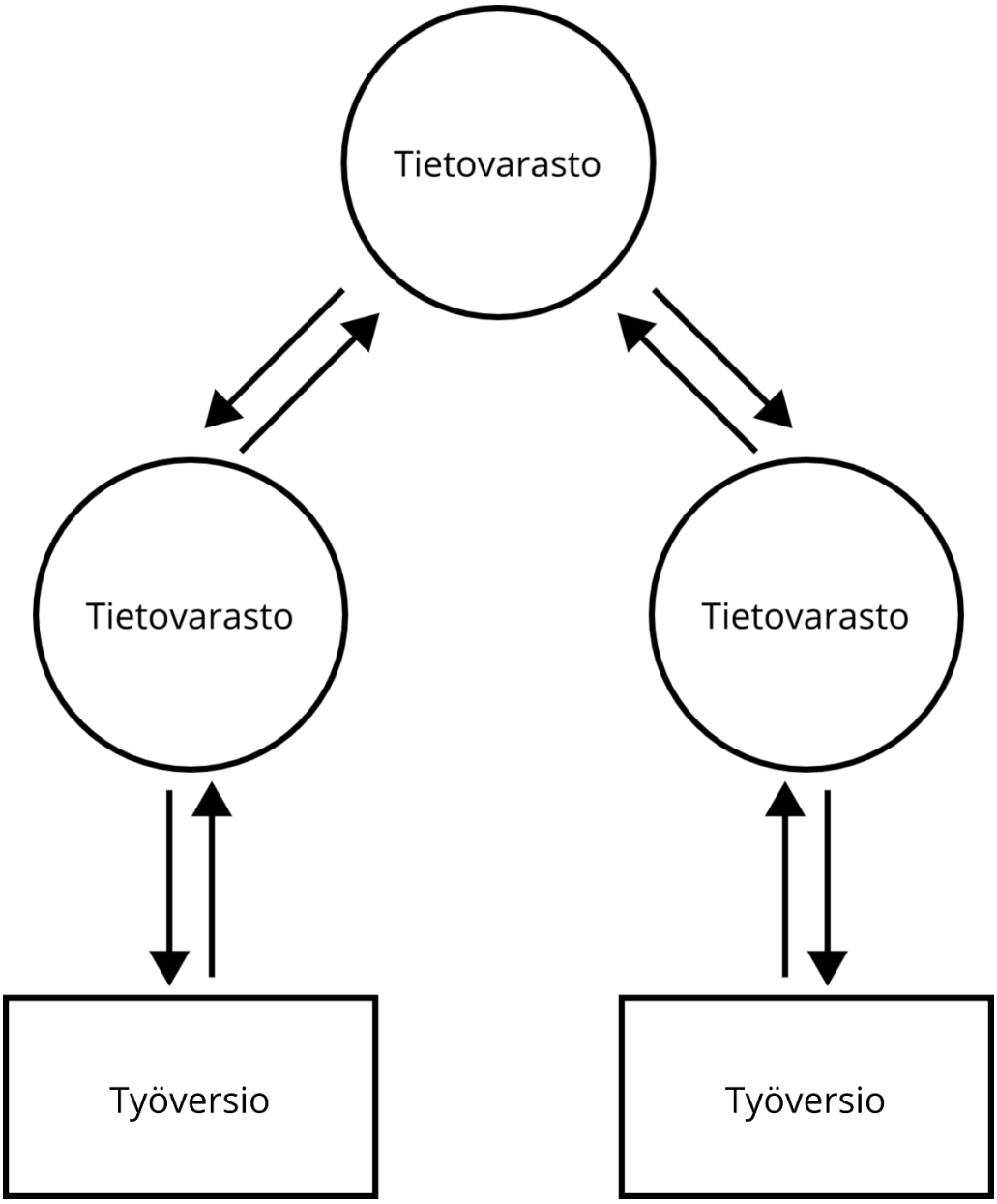
Kuva 9. Keskitetty versionhallinta

1.2 Hajautetut versionhallintajärjestelmät

Hajautetut versionhallintajärjestelmät, kuten Git ja Mercurial, tallentavat tiedostojen versiohistorian jokaisen käyttäjän tietokoneelle. Hajautetut versionhallintajärjestelmät eivät varsinaisesti tarvitse keskitettyä palvelinta, koska muutokset versiohistoriaan voidaan jakaa monella tavalla esimerkiksi sähköpostin avulla. Paljon kuitenkin käytetään keskitettyä palvelinta, jonne muutokset päivitetään, helpottamaan yhteistyötä. Kuvassa 2 näkyy rakenne, jossa tällainen keskitetty palvelin on. Keskitettyyn versionhallintaan verrattuna kuvasta näkee, että käyttäjillä on paikalliset tietovarastot. Koska jokaisella käyttäjällä on täysi versiohistoria omalla koneellaan ei versionhallintatoimenpiteisiin tarvita yhteyttä palvelimeen. Vain tietojen päivittäminen palvelimelta tai palvelimelle vaatii verkkoyhteyden. Tästä johtuen versionhallintatoimenpiteetkin ovat nopeita suorittaa. Version tai haaran vaihdot ja muutosten tallennus ovat välittömiä. Koko versiohistorian tallentamisella on hyvänä puolena myös automaattinen tiedon varmuuskopioituminen.

(jatkuu)

Vaikka palvelin menisi rikki, saadaan tiedot todennäköisesti palautettua jonkun käyttäjän koneelta. Koko versiohistorian tallentamisella on myös huonot puolensa. Kaikkien tiedostoversioiden säilytys vie tilaa. Tämä ei ole ongelma tekstitiedostojen kohdalla, koska vain muutokset tallennetaan, mutta binääritiedostoista tallennetaan jokainen versio kokonaisena. Tämä johtaa nopeasti tietovaraston paisumiseen. Tähän on onneksi kuitenkin kehitetty ratkaisuja. Esimerkiksi Git-LFS (Large File Storage) tallentaa Git versiohistoriaan vain pienen pointteritiedoston, joka osoittaa mistä sitä vastaava binääritiedosto löytyy. Binääritiedostot tallennetaan eri paikkaan ja vain haluttu versio ladataan käyttäjän koneelle.



Kuva 10. Hajautettu versionhallinta

(jatkuu)

1.3 Git

Git on hajautettu versionhallintajärjestelmä, jonka on kehittänyt Linus Torvalds, Linuxin kehittäjä. Gitistä julkaistiin ensimmäinen version vuonna 2005. Tämä ohje opettaa Gitin perustoiminnot, mutta Gitillä voi tehdä paljon muita vaativampia asioita, joita ei tässä ohjeessa käsitellä. Jos haluaa perehtyä Gitiin syvällisemmin kannattaa tutustua Scott Chacon ja Ben Straubin kirjoittamaan Pro Git kirjaan, johon tämäkin ohje pitkälti pohjautuu. Pro Git löytyy osoitteesta https://git-scm.com/book/en/v2. Lisäksi osoitteesta https://git-scm.com/docs löytyy käsikirja Gitin komentojen käyttöön. Myös kirjoittamalla ”--help” Git komennon perään saa lisätietoja komennon käytöstä.

1.4 GitLab

GitLab on web-käyttöliittymä Gitin etätietovarastojen hallintaan. Myös muita palveluita on kuten GitHub, Bitbucket ja Azure DevOps Services. GitLab tarjoaa ilmaisen tietovarastojen tallennuspalvelun ja myös ilmaisen avoimen lähdekoodin version GitLabista, joka mahdollistaa itseylläpidetyt GitLab palvelimet. GitLabin dokumentaatio löytyy osoitteesta https://docs.gitlab.com/ee/README.html.

(jatkuu)

2 GITIN ASENNUS

2.1 Ubuntu

Gitin asennus Ubuntulla onnistuu hyvin helposti ajamalla terminaalissa komento:

```
$ sudo apt install git
```

Asennuksen jälkeen Git on käyttövalmis. Gitiä käytetään terminaalista.

Ubuntulla Git ei vakiona tallenna käyttäjätunnuksia. Jos haluaa, että käyttäjätunnukset tallennetaan automaattisesti, ettei niitä tarvitse syöttää joka kerta tietovarastoa päivittäessä pitää asentaa Gitin GNOME Keyring integraatio. Tämä onnistuu ajamalla seuraavat komennot terminaalissa:

```
$ sudo apt install libgnome-keyring-dev
$ sudo make -C /usr/share/doc/git/contrib/credential/gnome-
keyring
$ git config --global credential.helper
/usr/share/doc/git/contrib/credential/gnome-keyring/git-
credential-gnome-keyring
```

Nämä komennot asentavat integraation kääntämiseen tarvittavat tiedostot, kääntävät integraatio ohjelman ja asettavat sen käyttöön Gitissä.

2.2 Windows

Windowsille Gitin asennusohjelma löytyy osoitteesta https://git-scm.com/downloads. Asennuksessa voi käyttää vakioasetuksia, mutta kannattaa laittaa automaattinen päivitysten tarkistus päälle. Lisäksi jollei halua käyttää VIM tekstieditoria, voi asennettaessa valita haluamansa editorin. Tämän voi vaihtaa myöhemmin konfiguroinnissa. Asennuksen jälkeen Gitiä voi käyttää komentoriviltä, mutta on suositeltavaa käyttää Gitin mukana tulevaa Git Bash terminaaliohjelmaa.

2.3 Mac

Macilla Gitin asennus onnistuu ajamalla terminaalissa komento:

(jatkuu)

```
$ git --version
```

Jos Git on jo asennettuna tulostaa tämä komento asennetun Gitin versio tiedot, mutta jos Git ei ole vielä asennettuna, Mac kysyy, haluatko asentaa sen. Asennuksen jälkeen Git on käytettävissä terminaalista.

2.4 Konfigurointi

Jotta Git osaisi merkitä muutoksiin tekijän tiedot, pitää se kertoa Gitille. Tämä onnistuu helposti seuraavilla komennoilla:

```
$ git config --global user.name ’Etunimi Sukunimi’
$ git config --global user.email sähköposti@osoite.com
```

Näillä asetuksilla Git käyttää samaa tekijää kaikkiin tietovarastoihin, mutta se on mahdollista määrittää myös per tietovarasto vaihtamalla ”--global” argumentti ”--local” argumentiksi.

Gitin käyttämän tekstieditorin, yhdistysohjelman ja vertailuohjelman voi myös vaihtaa konfiguraatiossa. Esimerkkinä on komennot, joilla ne saa vaihdettua käyttämään Visual Studio Codea.

```
$ git config --global core.editor ’code --wait’

$ git config --global merge.tool vscode
$ git config --global mergetool.vscode.cmd ’code --wait $MERGED’

$ git config --global diff.tool vscode
$ git config --global difftool.vscode.cmd ’code --wait --diff
$LOCAL $REMOTE’
```


(jatkuu)

3 GITIN PERUSTEET

3.1 Tietovaraston luonti tai hankkiminen

Uuden tietovaraston saa luotua siirtymällä haluamaansa kansioon ja ajamalla komento:

```
$ git init
```

Jos sinulla on valmis Git tietovarasto jossain saatavilla ja haluaisit kopioida sen koneellesi, se onnistuu komennolla:

```
$ git clone <etätietovaraston osoite> <kansio>
```

Etätietovaraston osoitteesta lisää luvussa 4. Kansiota ei ole pakko määrittää. Silloin kansion nimeksi tulee etätietovaraston nimi.

3.2 Tiedostojen tai muutosten lisääminen

Gitissä versiot ovat committeja, jotka vastaavat koko tietovaraston tilaa commitin tekohetkellä. Tiedostoja tai muutoksia pystyy valmistelemaan seuraavaa committia varten komennolla:

```
$ git add <tiedosto>
```

Tiedostona voi käyttää myös esimerkiksi ’*’-merkkiä, jolloin kaikki kansiossa olevat tiedostot valmistellaan. Jos haluaa poistaa tiedostoja tietovarastosta, käytetään komentoa:

```
$ git rm <tiedosto>
```

Jos haluaa siirtää tai nimetä uudelleen tiedoston, onnistuu se komennolla:

```
$ git mv <vanha tiedosto> <uusi tiedosto>
```

3.3 Muutosten tarkastelu

Jos halutaan tarkastella tiedostojen tilaa tai Gitin tilaa yleisesti voidaan käyttää komentoa:

```
$ git status
```

Se näyttää onko tiedostoihin tehty muutoksia, onko niitä lisätty tai poistettu ja missä tilassa ne ovat. Lisäksi siitä näkee, onko Git jossain erikoistilassa kuten ”merging” ja se antaa usein

(jatkuu)

vihjeitä mitä komentoja saatat mahdollisesti haluta käyttää. Jos muutoksia halutaan tarkastella lähemmin, voidaan käyttää komentoa:

```
$ git diff
```

Se näyttää muutokset terminaalissa. Jos halutaan tarkastella muutoksia vertailutyökalulla, ja se on asetettu joksikin, käytetään komentoa:

```
$ git difftool
```

3.4 Muutosten tallentaminen tietovarastoon

Jotta muutokset saadaan tallennettua pysyvästi tietovarastoon, käytetään komentoa:

```
$ git commit -m ”Viesti tehdyistä muutoksista”
```

Tämä tallentaa muutokset vasta paikalliseen tietovarastoon. Muutosten lähettämisestä muiden saatavaksi kerrotaan luvussa 3.6.

3.5 Muutosten peruminen

Jos haluaa muokata edellistä committia, esimerkiksi lisätä tiedostoja committiin tai muuttaa sen viestiä, onnistuu se komennolla:

```
$ git commit --amend
```

Ennen kuin lähettää commitin, voi asioita poistaa commitista komennolla:

```
$ git reset HEAD <tiedosto>
```

Hylätäksesi muutokset johonkin tiedostoon aja komento:

```
$ git checkout -- <tiedosto>
```

3.6 Etätietovaraston käyttö

Jos hankit jo olemassa olevan tietovaraston komenolla ”git clone”, on sinun tietovarastossasi jo valmiiksi tieto etätietovarastosta. Sen sijaan, jos teit uuden tietovaraston komennolla ”git init” täytyy sinun tehdä uusi etätietovarasto, esimerkiksi GitLabissa, ja lisätä sen osoite tietovarastoosi, jotta Git tietää missä se on. Tämä onnistuu komennolla:

(jatkuu)

```
$ git remote add origin <etätietovaraston osoite>
```

Tämä lisää etätietovaraston osoitteen nimellä ”origin” tietovarastoosi. Nyt kun tietovarastossasi on tieto etätietovaraston osoitteesta, voit päivittää muutoksesi muiden saataville. Jos olet uudessa tietovarastossa tai uudessa haarassa, käytä komentoa:

```
$ git push -u origin paikallinen_master:eta_master
```

Tämä lähettää paikalliset muutoksesi haarasta ”paikallinen_master” etätietovaraston ”origin” haaraan ”eta_master”. Argumentti -u asettaa upstreamin, jota komennot jatkossa käyttävät, ettei etätietovarastoa ja haaraa välttämättä tarvitse kirjoittaa joka kerta. Jatkossa siis riittää komento:

```
$ git push
```

Kun haluat hakea muiden muutokset etätietovarastosta, onnistuu se komennolla:

```
$ git fetch
```

Muutosten yhdistäminen paikalliseen versioosi onnistuu komennolla:

```
$ git merge FETCH_HEAD
```

Jos paikallinen versio ja etäversio ovat ristiriidassa aiheutuu tästä ”merge commit”, jossa sinun pitää yhdistää muutokset toisiinsa sopiviksi. Tästä lisää luvussa 3.7. Jos haluaa pitää versiohistorian siistinä voi käyttää komentoa:

```
$ git rebase FETCH_HEAD
```

Tämä muuttaa versiohistoriaa niin, että sinun paikalliset muutokset siirtyvät uusimmiksi eli näyttää siltä kuin olisit tehnyt muutoksesi uusimman etäversion päälle. Rebasella pystyy tekemään hyvin monimutkaisia muutoksia, joita ei tässä ohjekirjassa käsitellä, mutta niistä voi lukea Pro Gitistä. Nämä operaatiot voi tehdä nopeammin, kun käyttää komentoa:

```
$ git pull
```

muutosten hakemiseen ja yhdistämiseen ja komentoa:

```
$ git pull --rebase
```

muutosten hakemiseen ja rebasetukseen.

(jatkuu)

3.7 Yhdistäminen

Yhdistämistä käytetään paikallisten tiedostojen päivityksessä etätietovaraston versioon ja haarojen yhdistämisessä. Yhdistäminen tapahtuu siirtymällä haaraan, johon haluat yhdistää ja ajamalla komennon:

```
$ git merge <yhdistettävän haaran nimi>
```

Jos yhdistämisessä ilmeni ristiriitoja, joutuu ne ratkomaan jollain tavalla. Nähdäksesi missä tiedostoissa on ristiriita voit käyttää komentoa:

```
$ git status
```

Git merkkaa ristiriidat tiedostoihin seuraavalla tavalla:

```
<<<<<<< haara_johon_yhdistetään:tiedosto
Ristiriitainen sisältö haarassa, johon yhdistetään.
=======
Ristiriitainen sisältö haarassa, joka yhdistetään.
>>>>>>> yhdistettävä_haara:tiedosto
```

Näistä sinun pitää päättää kumpaa versiota käytetään, vai pitääkö mahdollisesti muokata niiden yhdistelmä. Sen jälkeen poistat ylimääräiset rivit, mukaan lukien ”<<<<<<<”, ”=======” ja >>>>>>>” rivit. Kun olet ratkaissut ristiriidat pitää tiedostot merkata ratkaistuiksi komennolla:

```
$ git add <tiedosto>
```

Yhdistämiseen voi käyttää myös yhdistämistyökalua, jos sellainen on asetettu, komennolla:

```
$ git mergetool
```

Muutosten tallentamisen jälkeen, kun yhdistämistyökalun sulkee, Git kysyy, onnistuiko ristiriitojen selvitys. Jos vastaa kyllä, Git merkkaa ristiriidan ratkaistuksi. Kun kaikki ristiriidat on ratkaistu ja merkattu ratkaistuiksi, viimeistellään yhdistäminen komennolla:

```
$ git commit
```


(jatkuu)

3.8 Haarat

Haarat ovat poikkeamia ohjelman pääkehityslinjasta. Ne mahdollistavat muutosten teon erillään sotkematta päähaaran koodia. Voit esimerkiksi tehdä haaran uuden ominaisuuden kehitystä tai kokeellisen koodin testausta varten ja päähaaran koodi pysyy käyttökelpoisena. Nyt jos esimerkiksi huomaisit jonkin virheen mikä pitää korjata, mutta ei liity kehittämääsi uuteen ominaisuuteen, siirtyisit päähaaraan ja tekisit uuden haaran virheen korjausta varten. Haarojen välillä on helppo liikkua mahdollistaen tehtävän vaihtamisen nopeasti. Kun työt haarassa on saatu valmiiksi, voidaan se yhdistää takaisin päähaaraan.

Uuden haaran voi tehdä komennolla:

```
$ git branch <haaran nimi>
```

ja haarojen välillä voidaan liikkua komennolla:

```
$ git checkout branch <haaran nimi>
```

Jos ei muista kaikkien haarojen nimiä voi ne tarkistaa komennolla:

```
$ git branch
```

Uuden haaran teon ja siihen siirtymisen voi tehdä myös samaan aikaan komennolla:

```
$ git checkout -b <haaran nimi>
```

Kun haaraa ei enää tarvitse, esimerkiksi jos se on yhdistetty päähaaraan, voidaan se poistaa komennolla:

```
$ git branch -d <haaran nimi>
```


(jatkuu)

4 GITLABIN PERUSTEET

4.1 Tietovaraston luonti

GitLabissa voi luoda uuden tietovaraston käyttöliittymän navigaatiopalkissa olevasta plus merkistä valitsemalla uusi projekti (new project). Seuraavaksi projektille pitää antaa nimi ja valita sen näkyvyystaso. Näkyvyystasoja on kolme yksityinen (private), sisäinen (internal) ja julkinen (public). Yksityinen projekti näkyy vain sen omistajalle ja siihen erikseen lisätyille käyttäjille. Sisäinen projekti näkyy kaikille palveluun kirjautuneille käyttäjille ja julkinen projekti näkyy kaikille käyttäjille. Lisäksi voidaan alustaa tietovarasto README tiedostolla. Sen alustaminen mahdollistaa etätietovaraston kloonauksen omalle koneelle, mutta jos sinulla on jo paikallinen tietovarasto, jonka haluat lähettää etätietovarastoon, älä tee tätä alustusta.

Nyt voit joko kloonata etätietovaraston koneellesi, jos alustit sen README tiedostolla tai lähettää paikallisen tietovarastosi etätietovarastoon, jos et alustanut. Kloonaus tapahtuu käyttämällä etätietovaraston käyttöliittymästä löytyvää kloonaa (clone) painiketta, josta kopioit HTTPS osoitteen ja käytät sitä Gitin kloonaus komennossa. Paikallisen tietovaraston tai kansion etätietovarastoon lähettämiselle löytyy komennot tyhjän etätietovaraston käyttöliittymästä.

Tietovaraston omistaja eli luoja voi lisätä siihen käyttäjiä jäseniksi. Tämä onnistuu käyttöliittymän vasemmasta laidasta menemällä asetukset (settings) valikkoon ja sieltä jäsenet (members) valikkoon. Siellä voit hallita käyttäjiä ja ryhmiä, joilla on oikeudet projektiisi. Uusille käyttäjille pitää valita jokin rooli neljästä vaihtoehdosta vierailija (guest), reportteri (reporter), kehittäjä (developer) ja ylläpitäjä (maintainer). Vierailijalla on vain katseluoikeudet, reportterilla on lisäksi joitain projektin hallintaoikeuksia, kehittäjällä on lisäksi oikeus puskea (push) dataa tietovarastoon ja tehdä muita muutoksia ja ylläpitäjällä on kaikki projektin sisäiset oikeudet.

4.2 Ryhmän luonti

GitLabissa voidaan luoda ryhmiä ja niille aliryhmiä kaksikymmentä tasoa. Ryhmien luonti onnistuu samasta paikasta kuin tietovaraston luontikin. Ryhmiin voi lisätä projekteja eli

(jatkuu)

tietovarastoja. Ryhmiin voi myös lisätä käyttäjiä, jolloin heillä on pääsy kaikkiin aliryhmiin tai projekteihin riippuen asetetuista oikeuksista.

4.3 Forkkaaminen

Forkkaaminen (forking) on etätietovarastojen kloonaamista. Voit forkata projektin, johon sinulla on katseluoikeudet. Näin sinun ei tarvitse olla projektin jäsen työskennelläksesi sen parissa. Tätä ominaisuutta käytetään paljon avoimen lähdekoodin ohjelmistokehityksessä. Forkattuasi projektin voit kehittää sitä omassa forkissasi ja jos saat jotain aikaan, voit tehdä yhdityspyynnön jonka avulla muutoksesi voidaan yhdistää alkuperäiseen projektiin, jos ne hyväksytään. Forkkaaminen tapahtuu painamalla fork painiketta projektin käyttöliittymässä.

4.4 Ongelmailmoitukset

Ongelmailmoitukset (issues) ovat esimerkiksi bugi-ilmoituksia tai keskustelua uuden ominaisuuden toteutuksesta. Niitä voi luoda ongelmailmoitus listan painikkeesta luo uusi ongelmailmoitus (new issue). Siihen täytetään kuvaava otsikko ja yksityiskohtainen kuvaus. Julkaistuja ongelmailmoituksia voi kommentoida.

4.5 Yhdistyspyynnöt

Kun sinulla on valmista koodia haarassa tai forkissa voit tehdä yhdistyspyynnön (merge request). Yksin työskennellessä tämä on melko turhaa, mutta jos työskentelet ryhmässä kannattaa käyttää yhdistyspyyntöjä, koska niitä käyttämällä ryhmäsi jäsenet voivat antaa palautetta koodista ennen kuin se yhdistetään. Yhdistyspyynnön voi tehdä haaran tai forkin käyttöliittymästä painamalla ”luo yhdistyspyyntö” (create merge request) painiketta. Siihen on täytettävä kuvaava otsikko ja kuvaus tehdyistä muutoksista. Yhdistyspyynnön voi hyväksyä joku, jolla siihen on oikeudet. Olisi hyvä, jos sen hyväksyy joku muu kuin sen tekijä.

LIITE 2. Harjoitustehtäviä

Git harjoitustehtäviä

Harjoitustehtävissä 2 ja 4 palautetaan samalla tehtävien 1 ja 3 tulokset.

Harjoitustehtävä 1.

Haluat tehdä uuden ohjelmointi projektin ja käyttää siinä versionhallintaa. Tehtävänäsi on siis tehdä uusi kansio ja luoda siihen uusi tietovarasto koodeineen. Suoritettavat vaiheet ovat seuraavat:

1. Luo uusi kansio.
2. Luo siihen Git tietovarasto.
3. Lisää kansioon jokin tekstitiedosto, jossa lukee ”Git harjoitustehtävä”.
4. Valmistele tiedosto committia varten.
5. Tallenna muutokset tietovarastoon.

Nyt sinulla on paikallinen Git tietovarasto, jossa on yksi tekstitiedosto.

Harjoitustehtävä 2.

Haluat jakaa edellisen tehtäväsi tiedoston muiden saatavaksi ja mahdollisesti muokattavaksi. Tehtävänäsi on tehdä uusi etätietovarasto GitLabiin ja yhdistää se paikalliseen tietovarastoosi. Lopuksi jaat paikallisen tiedostosi GitLabiin. Suoritettavat vaiheet ovat seuraavat:

1. Luo uusi julkinen etätietovarasto GitLabiin.
2. Lisää etätietovarastosi osoite paikalliseen tietovarastoosi.
3. Lähetä paikalliset muutoksesi etätietovarastoon.

Nyt tiedostosi on muiden saatavilla. Lisää tehtävänpalautukseen linkki tähän etätietovarastoon.

Harjoitustehtävä 3.

Haluat tehdä kokeellisia muutoksia edellisen tehtävän tiedostoosi. Tehtävänäsi on luoda uusi haara ja muokata siinä tiedostoasi ja tallentaa muutokset tietovarastoon. Suoritettavat vaiheet ovat seuraavat:

1. Luo uusi haara, mutta älä siirry siihen samalla.
2. Alkuperäisessä haarassasi poista tekstitiedostossa oleva tekstirivi ja kirjoita tilalle ”Alkuperäinen haara”. Tämä tehdään seuraavaa tehtävää varten.

(jatkuu)

3. Tallenna muutokset tietovarastoon.
4. Siirry äsken luomaasi haaraan.
5. Poista tekstitiedostossa oleva tekstirivi ja kirjoita tilalle ”Uusi haara”.
6. Tallenna muutokset tietovarastoon.

Harjoitustehtävä 4.

Tässä tehtävässä harjoitellaan ristiriitojen ratkomista yhdistämistilanteessa. Edellisessä tehtävässä luotiin tarkoituksella ristiriita, joka ratkaistaan tässä tehtävässä. Ensin kuitenkin harjoitellaan muutosten lähettämistä etätietovarastoon ja hakemista etätietovarastosta. Suoritettavat vaiheet ovat seuraavat:

1. Luomassasi haarassa lähetä muutokset etätietovaraston alkuperäiseen haaraan.
2. Siirry alkuperäiseen haaraasi.
3. Hae muutokset etätietovarastosta ja yritä yhdistää ne paikalliseen versioosi.
    - Voit joko hakea ja yhdistää erillisillä tai yhdellä komennolla.
    - Huomataan, että tulee ristiriita.
4. Korjaa ristiriita
    - Ristiriitaa korjatessa pitää valita mikä olisi paras vaihtoehto niin että korjattu tiedosto toimii.
    - Vaihtoehtoina on säilyttää paikalliset tai yhdistettävät muutokset tai tehdä täysin uusi muutos.
    - Tässä tapauksessa tehdään uusi muutos ja kirjoitetaan tekstin tilalle ”Yhdistetty haara”.
5. Lähetä muutokset etätietovarastoon.

Lisää tehtävänpalautukseen linkki tähän etätietovarastoon.