Kim Kohvakka

Versionhallinta osana projektinhallintaa

Opinnäytetyö

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

2024

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Insinööri (AMK), sähkö- ja automaatiotekniikka
Tekijä/Tekijät	Kim Kohvakka
Työn nimi	Versionhallinta osana projektinhallintaa
Toimeksiantaja	Kaukosäätö Mikkeli Oy
Vuosi	2024
Sivut	60 sivua, liitteitä 4 sivua
Työn ohjaaja(t)	Jyrki Liikanen, Teemu Manninen

TIIVISTELMÄ

Opinnäytteen aiheena oli tutustua versionhallintaohjelmistoon ja ottaa sellainen käyttöön ja testata sen toimivuutta Fidelix-automaatiojärjestelmässä. Käyttöönottovaiheesta koottiin selkeä ohjeistus tähän opinnäytetyöhön, jonka ohjeistusta voidaan hyödyntää esimerkiksi uuden työntekijän tullessa taloon.

Ensimmäisenä opinnäytetyössä tutustuttiin versionhallinnan historiaan ja nykytilanteeseen, millaisia ohjelmistoja versionhallintaan on tarjolla ja mitä pilvitallennuspalvelua valitun ohjelmiston käytössä kannattaa käyttää. Tutustumis- ja selvitystyön jälkeen valikoitiin versionhallintaohjelmistoksi työhön ilmaisjakelussa oleva ja laajasti suosittu Git-ohjelmisto, jonka käyttö aloitettiin versionhallinnan perusteet opettelemalla. Git-ohjelmistolle tarjotaan myös etäkäyttö- ja tallennusmahdollisuus muun muassa GitHub-nimisellä pilvitallennuspalvelulla.

Git-ohjelmiston perusteiden ollessa hallussa siirryttiin ohjelmiston testaamiseen ja käyttöönottoon itse Fidelix-automaatiojärjestelmässä. Ohjelmiston testauksen jälkeen todettiin selväksi, että Git-ohjelmisto sopii versionhallintaan käytössä olevan Fidelix-järjestelmän kanssa. Viimeisenä työssä suoritettiin GitHubin käyttöönotto Git-ohjelmistoon, jotta versionhallintaa voidaan käyttää etäpalvelimelta käsin ja versiointia voidaan toteuttaa etänä sekä paikallisesti.

Työn toteuttaminen onnistui ongelmitta, ja työn haluttuun tavoitetilanteeseen päästiin kohtuullisessa ajassa. Mikäli työ olisi toteutettu työelämässä, olisi ohjelmiston käyttöönotto- ja selvitystyö tullut viemään useamman päivän yhdeltä ihmiseltä, vaikkakin työn toteuttamisella tullaan tulevaisuudessa säästämään paljon aikaa sekä ylimääräistä päänvaivaa ohjelmistoversioiden hallinnan osalta. Lisäksi ohjelmiston käyttöönotto mahdollistaa useammalle eri ohjelmoitsijalle ohjelmistokoodin viimeisimmän version hallussapidon, joka on erittäin toivottua ohjelmointityön osalta.

Asiasanat: versionhallinta, projektinhallinta, ohjelmointi, automaatio, Fidelix

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering, electrical and automation engineering
Author (authors)	Kim Kohvakka
Thesis title	Version control as part of project management
Commissioned by	Kaukosäätö Mikkeli Oy
Time	2024
Pages	60 pages, 4 pages of appendices
Supervisor	Jyrki Liikanen, Teemu Manninen

ABSTRACT

The purpose of this thesis was to get familiarized with a version control software, to implement it and test its functionality within Fidelix automation systems. This study presents clear instructions of the implementation phase to be later utilized, for example, when a new employee is recruited in a company.

The first phase of this study was getting to know the history of version control and present situation regarding what kind of software for version control are currently available and which cloud saving service with selected version control software is the best to use. Version control software was selected after the familiarizing and surveying phase. The selected software was the freely distributed and widely used Git, and its usage was initiated by learning the fundamentals of the version control. Remote control and saving functions are also offered for example in GitHub cloud saving service.

After learning the fundamentals, the next step was to test and launch the version control software in Fidelix automation system. The testing showed clearly that Git is suitable software for the version control of Fidelix automation system. Finally, GitHub was implemented in Git software so that the version control can be used remotely, and versioning can be managed locally and remotely.

This study succeeded without any major problems and the desired objective was reached within a reasonable time. Should this study have been a real working-life project, the commission of the software and surveying would have taken a couple of days from one employee, although the implementation will save a lot of time and unnecessary effort with managing the software versions. Furthermore, the implementation of the software allows several programmers to possess the latest software code, which is a strong asset in programming.

Keywords: version control, project management, programming, automation, Fidelix

SISÄLLYS

LYHENTEET

Branch	Nimike, jota Git-ohjelmisto käyttää versiohaarasta.
Commit	Komento, jota käytetään tiedostojen lähettämiseksi haaraan.
CVCS	Centralized Version Control System. Keskitetty versionhallintajärjestelmä. Järjestelmä, jossa versiointi toteutetaan yhdelle palvelimelle.
DVCS	Distributed Version Control System. Hajautettu versionhallintajärjestelmä. Järjestelmä, joka ei riipu yksittäisestä palvelimesta.
Fetch	Komento, jota käytetään toisen haaran tiedostojen vertailemiseksi nykyiseen haaraan nähden.
LVCS	Local Version Control System. Paikallinen versionhallintajärjestelmä. Järjestelmä, jossa versiointi toteutetaan vain paikallisesti.
Repository	Tietovarasto. Nimike, jota Git-ohjelmisto käyttää projektihakemistosta.
Merge	Komento, jota käytetään haarojen tiedostojen yhdistämiseen.
Grafiikkakuva	Graafinen kuva, joka sisältää VAKin pistetunnuksia ja erilaisia symboleita. Yleensä sisältää kokonaiskuvan ohjattavasta prosessista.

Pistetunnus	Ohjelmistopiste, joka voi olla fiktiivinen tai fyysinen. Esimerkiksi lämpötilamittaus sidotaan omaksi pistetunnukseksi. Lämpötilan mittaus kirjoitetaan omaan pistetunnukseen ja pistetietokantaan, josta ohjelmisto voi sitä kysellä.
VAK	Valvonta-alakeskus. Nimitys, joksi kiinteistöautomaatiokeskuksia yleensä kutsutaan. Sisältää ohjelmistoprosessorin ja mittaus-, säätö- ja ohjauspiirikortteja.

1 JOHDANTO

Opinnäytetyö toteutettiin toimeksiantona kiinteistöautomaation parissa toimivalle Kaukosäätö Mikkeli Oy:lle. Tavoitteena työssä oli tutustua markkinoilla oleviin versionhallintaohjelmistoihin ja selvittää ja käyttöönottaa toimiva ohjelmisto versionhallintaan Fidelix-automaatiojärjestelmissä.

Aluksi kerrotaan versionhallinnan historiasta ja projektinhallinnan nykytilanteesta sekä miten pyrittyyn tavoitteeseen opinnäytetyössä päästään. Seuraavaksi kerrotaan versionhallinnan toteuttamiseen tarjolla olevista versionhallintaohjelmistoista sekä Fidelix-järjestelmän työhön tarvittavista työkaluista ja resursseista.

Työn loppuosiossa kerrotaan, kuinka versionhallintaohjelmistoa käytetään sellaisenaan ja testataan sen toimivuus Fidelix-käyttöympäristössä selittäen hieman käyttöympäristön sisältämien ohjelmistojen toiminnasta, jotta voidaan varmistua siitä, ettei versionhallintaohjelmisto tee jotakin epätoivottua.

2 VERSIONHALLINNAN HISTORIA

Versionhallintaohjelmistojen esiasteena voidaan pitää IBM:n vuonna 1962 OS/360-käyttöjärjestelmälle kehittämän IEBUPDTE-ohjelmiston päivitystyökalua. Kuitenkin ensimmäinen varsinainen versionhallintaohjelmisto Source Code Control System (SCCS) sai alkunsa vuonna 1972 samaiselle IBM:n OS/360-käyttöjärjestelmälle. Siinä oli jo useita samoja toimintoja kuin nykypäivän versionhallintaohjelmistoissakin on kuten esimerkiksi kyky luoda tai muokata tiedostoja sekä seurata tiedostojen muutoksia. Se ei kuitenkaan tukenut useamman ohjelmoijan samanaikaista työskentelyä. (McMillan 2021.)

Tämän jälkeen markkinoille tuli vuonna 1982 Revision Control System (RCS), joka kilpaili silloin suositun SCCS:n kanssa. RCS toi uutena ominaisuutena markkinoille tiedostojen käsittelytavan, jossa ohjelmisto otti pohjakseen viimeisimmän tiedoston, josta muut versiot luotiin sen sijasta, että jokainen versio tiedostosta tallennettiin. Silti RCS ei vielä tukenut kuin yhden ohjelmoijan työskentelyn versionhallinnan parissa. (McMillan 2021.)

Muutama vuosi RCS:n julkaisun jälkeen julkaistiin Concurrent Versions System (CVS), joka mahdollisti ensimmäisenä useamman ohjelmistokehittäjän työskentelyn samanaikaisesti. Sen asiakasserverimalli ja haarajärjestelmä muodosti huomattavasti nykyaikaisemman ohjelmistomallin. (McMillan 2021.)

Sittemmin SCCS on jo kuopattu vuosituhannen vaihteessa, toisinkuin RCS, jonka viimeisin versio on julkaistu vuonna 2022 (GNU RCS 2022).

2000-luvun alussa seuraava kehitysaskel versionhallintaohjelmistoista tuli CollabNet-nimiseltä yritykseltä, joka kehitti Subversion (SVN) -ohjelmiston CVS:n entisiä käyttäjiä ajatellen. Ohjelmisto sisälsi useita toimintoja CVS:stä, jotta CVS:n käyttäjien olisi helppo siirtyä Subversioniin. SVN on hyvä esimerkki keskitetystä versionhallintajärjestelmästä (CVCS), jossa muutokset tehdään yhteen palvelimella sijaitsevaan projektiin, jonka jokainen käyttäjä voi ladata tehdäkseen omat muutoksensa. (McMillan 2021.)

Myöhemmin 2005 Linus Torvalds julkaisi suositun Git-ohjelmiston, joka on hajautettu versionhallintajärjestelmä (DVCS). Siinä jokainen käyttäjä lataa pilvestä tietohakemiston historioineen, eli tietokannat sijaitsevat useilla eri palvelimilla. (McMillan 2021.)

Nykyään versionhallintaohjelmistoja on laajasti tarjolla eri ohjelmistoyrityksiltä ja -tahoilta, ja niiden sopivuus käytettävän järjestelmän mukaan määrittää suurimmaksi osaksi ohjelmiston valinnan.

3 VERSIONHALLINNAN MERKITYS FIDELIX-JÄRJESTELMISSÄ

Kiinteistöautomaatiojärjestelmät sisältävät yleensä paljon ohjelmakoodia sekä pistetunnuksia sekä grafiikkoja, jotka ovat hyvin työläitä luoda. Uutta projektia aloitettaessa menee grafiikkakuvien sekä pistetunnuksien ja ohjelmien tekoon isossa projektissa jopa viikko. Nämä pistetunnukset itsestään eivät vielä ohjaa prosessia, vaan luodut pistetunnukset ovat ohjelmoitava ja määriteltävä tekemään tiettyjä prosesseja. Ohjelmointi on aikaa vievää, ja se vaatii huolellisuutta, jotta prosessit toimivat halutunlaisesti.

Fidelix-järjestelmä on helppo ohjelmoida tekstipohjaisen ST-koodikielen takia, jossa kaikki komennot kirjoitetaan tekstinä eli erillistä graafista ohjelmointi-ikkunaa ei tässä koodikielessä tarvita. Ohjelmistolla kirjoitetut ohjelmakoodit tallennetaan raakatiedostoina ST-tiedostoformaatissa.

Kun koko kiinteistöautomaatiojärjestelmä on ohjelmoitu eli kaikki yksittäiset ST-tiedostoformaatissa tallennetut ohjelmakoodit ovat kirjoitettu valmiiksi, se käännetään IEC 61131-3-standardin konekieliseksi yhdeksi tiedostoksi DAT-tiedostomuotoon, jota Fidelix-ala-asema lukee ja suorittaa.

Fidelixin haittapuolena on se, että vaikka tämä DAT-tiedosto voidaan ladata ala-asemasta suoraan tietokoneelle TelNet-komentoa hyväksikäyttäen, ei pakattua IECProg.dat-tiedostoa voida enää kääntää takaisin sen sisältämiksi ST-tiedostoiksi, jolloin ohjelmakoodin tallessa pito ja sen hyvä varmuuskopiointi korostuu suuresti. Grafiikkakuvat, pistetunnukset sekä muut tietokannat saadaan ala-asemasta sentään ladattua muokkausta varten.

Fidelix-järjestelmät eivät tue online-ohjelmointia eli reaaliaikaista ohjelmointia, jolloin ohjelmoinnin lähdekoodi on oltava tallessa, mikäli ohjelmamuutoksia halutaan tehdä. Mikäli ohjelmistokoodi häviää, ei sitä voida palauttaa millään, vaan järjestelmä on ohjelmoitava uudestaan. Lisäksi tässä tapauksessa ei voida tarkalleen tietää, mitä ohjelma on pitänyt sisällään. Vaikka kohteesta olisi olemassa säätökaaviot, jossa on selitetty toimintaselostukset järjestelmän toiminnasta, on monesti kohteeseen tehty pieniä muutoksia, jota ei ole välttämättä dokumentoitu ylös mihinkään.

4 VERSIONHALLINNAN NYKYINEN TILANNE

Työn aloitusvaiheessa varsinaista versionhallintaa ei ole ollut käytössä. Versiointi on toteutettu lähinnä käsin ”leikkaa ja liitä”-menetelmällä, eli projektin kansiorakenne on kopioitu kahteen kertaan. Toinen kansio toimii työversiona, ja toinen on alkuperäinen lähdekoodi. Kun muutokset ovat valmiita ja ohjelmisto ajettu ala-asemaan ja todettu toimivaksi, on projektikansio varmuuskopioitu.

Tällainen versiointi ei mahdollista ensinnäkään useamman ohjelmoitsijan työskentelyä samanaikaisesti, ja projektia käsiteltäessä on hyvin vaikea muistuttaa itseään tallentamaan muutokset ja kopioimaan nykyinen versio nykyisillä muutoksilla hakemistoon yhdeksi versioksi, mikäli tulisi tarve esimerkiksi nyt poistaa tiedostoja, joita tarvittaisiin myöhemmin työn edetessä. Myös loki on itse manuaalisesti kirjoitettava esimerkiksi tekstitiedostoon, että mitä muutoksia kyseiseen versioon on tehty ja kenen toimesta.

Tällainen menetelmä vaatii siis käyttäjältään paljon huolellisuutta, jotta versiointi olisi toimivaa ja järkevää, sekä tehtyjen muutosten loki pysyisi yllä.

5 VERSIONHALLINNAN TAVOITETILANNE

Työn tavoitteena on ottaa käyttöön versionhallintaohjelmisto Fidelix-käyttöympäristössä. Versionhallintaohjelmistolla voidaan projektista luoda tasaisin väliajoin eri versioita, jotka tallennetaan paikallisesti tietokoneelle, kunnes versiot ajetaan päähakemistoon, joka yleensä on palvelimella.

Suurin ongelma työn aloitustilanteessa on projektin jälkeisen ohjelmistokoodin hallinta. Yleensä viimeisin versio on viimeisimpänä muutoksia tehneellä hallussa, on seuraavan ohjelmoijan aina pyydettävä nykyinen versio edelliseltä. Kun ohjelmoijia on useita ja ohjelmoijat voivat olla eri yrityksistä, tuottaa se ylimääräistä työtä.

Versionhallinnan suurin etu tulee siinä, että projektin pääversio pystytään pitämään pilvessä, jolloin projektin viimeisin versio on jatkuvasti kaikkien ohjelmoijien hallinnassa. Jokainen ohjelmoija voi ottaa nykyisen projektiversion työn alle, ja tehdä tarvittavat muutokset tiettyyn asiaan, vaikka samaan aikaan, kun toinen ohjelmoija tekee samaan projektiin jotakin toista asiaa samanaikaisesti. Vaikka muutoksia tehdään yhtä aikaa, voidaan versiointi suorittaa hallitusti, kun ohjelmisto pitää kirjaa mitä tiedostoja on muutettu. Kumpikin ohjelmoija voi ladata tekemänsä muutokset tähän pääversioon ilman, että projektin tiedostot menisivät sekaisin. Tällaisia yhtäaikaisesti tehtäviä muutoksia ei Fidelix-käyttöympäristössä juurikaan ilmaannu, mutta ohjelmiston ominaisuudesta ei haittaakaan ole.

Jokaiseen versioon voidaan kirjoittaa seliteteksti, jotta ohjelmoijan on helppo ymmärtää, mitä muutoksia projektiin on kyseisenä hetkenä tehty. Valmiit projektiversiot voidaan ladata pilveen, jolloin ne ovat varmuuskopioitu ja pysyvät tallessa.

Versionhallinta helpottaa huomattavasti projektin parissa työskentelyä, sillä se pitää kirjaa koko ajan sille määritetyn kansion sisältämistä tiedostoista, jolloin nähdään suoraan, mitä muutoksia projektiin on vaikka yhden päivän aikana tehty useamman ohjelmoijan toimesta.

Versionhallintaohjelmistolla voidaan myös palauttaa aikaisempien versioiden tiedostoja tarvittaessa, mikäli projektia toteutettaessa tulisi jokin virhe, joka vaatii aikaisempaa tiedostoversiota. Tällainen tilanne voisi olla esimerkiksi ohjelmointimuutos, joka ei toiminutkaan. Tällöin olisi helppoa palauttaa edellinen toimiva ohjelmaversio ilman, että ohjelmoijan tarvitsee itse manuaalisesti kopioida ja siirrellä tiedostoja Windowsin resurssienhallinnan kautta.

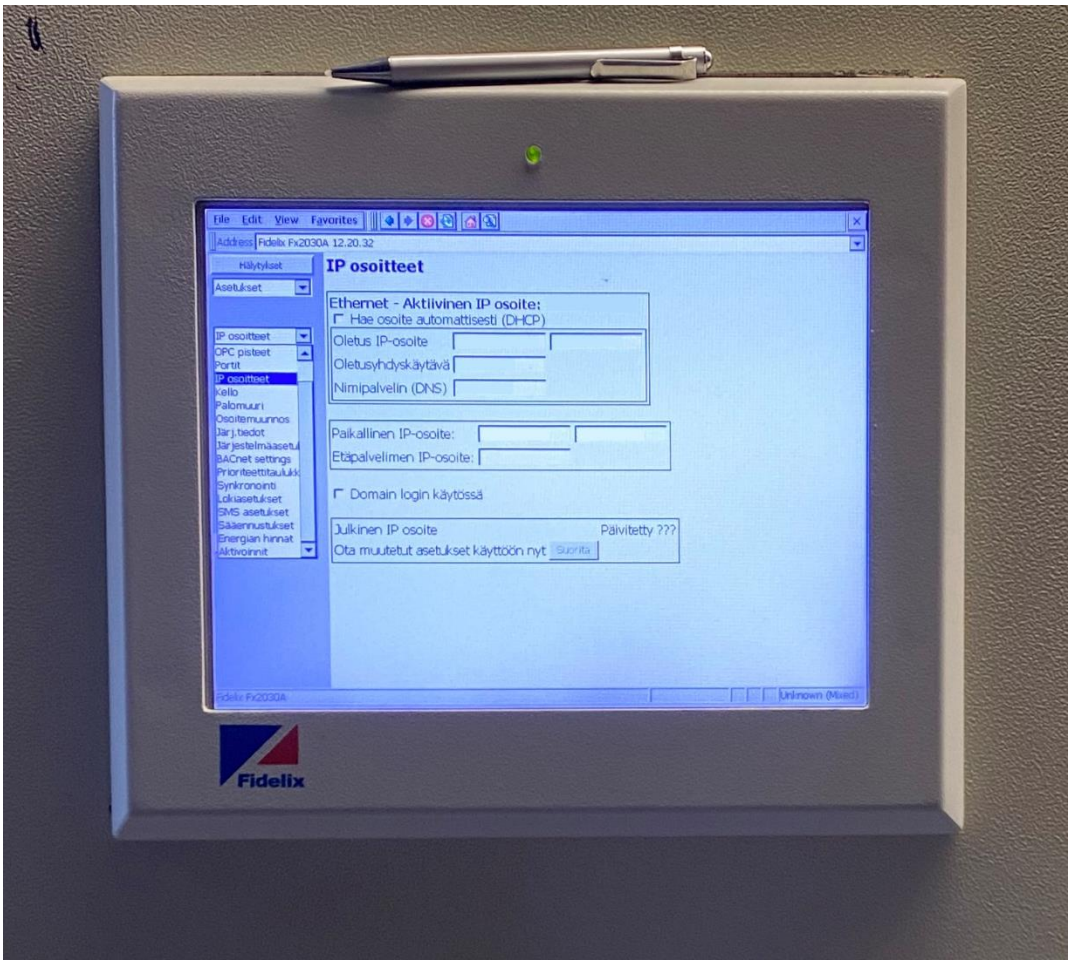
6 MENETELMÄT VERSIONHALLINNAN TOTEUTTAMISEEN

Työn toteuttamiseen vaadittiin pelkästään kaksi tietokonetta sekä Fidelixin valvonta-ala-asema FX-2030A. Tietokoneisiin oli asennettu Fidelix-järjestelmien käyttöönottoon vaadittavat ohjelmistot: Fidelix FX-Editor-projektiohjelmisto ja infoTeam OpenPCS-ohjelmointiohjelmisto sekä versionhallintaohjelmisto Git.

6.1 Valvonta-ala-asema Fidelix FX-2030A

Fidelix on vuonna 2002 Suomessa perustettu yhtiö, jonka nykyään omistaa ruotsalainen Assemblin Group AB. Yritys tuottaa tuotteita kiinteistöautomaation kokonaisuuden hallitsemiseen prosessoivasta älystä itse ohjauskortteihin saakka. Fidelix tarjoaa myös urakointi- ja huoltopalveluita järjestelmilleen. (Fidelix Oy s.a.)

Fidelix julkaisi vuonna 2013 valvonta-ala-aseman FX-2030 (Alakeskus versiot, Fidelix s.a.). Tästä tuli markkinoille päivitetty versio vuonna 2014 FX-2030A, jossa muutoksena oli COM4- ja COM5-lisäporttien muuttuminen omille lisäkorteilleen aikaisemman FX-2030-version erillisen USB-väylämuuntimen sijasta (Tekninen tiedote 025 2014).



Kuva 1. Fidelix FX-2030A-valvonta-ala-asema

Tämä kuvassa 1 näkyvä FX-2030A valvonta-ala-asema on jo poistunut tuote, ja markkinoille korvaavaksi tuotteeksi on julkaistu vuonna 2018 FX-3000C (kuva 2), joka on huomattavasti kehittyneempi versio edelliseen nähden, jonka suurimpana erona on erillinen Android-pohjainen näyttöpaneeli (kuva 3) ja erillinen keskusyksikkö (kuva 2). Käyttö on myös huomattavasti sulavampaa vanhempiin keskusyksikköihin verrattuna.

Syyskuussa 2024 julkaistiin FX-3000C:n seuraaja FX-3000X, joka on aikaisemmista keskusyksiköistä poiketen nyt Linux-pohjainen (FX-3000-X 2024).

Kuva 2. Fidelix FX-3000-C-valvonta-ala-asema

Kuva 3. Erillinen näyttöpaneeli Fidelix Visio-15-X

Tähän työhön valikoitu testikeskusyksiköksi FX-2030A, koska kyseinen toimiva yksikkö oli aikaisemmin saneeratusta kiinteistöstä jäänyt yli, ja täten oli helppo valita kustannustehokas vaihtoehto, kun työssä vain testataan versionhallinnan toimivuus ala-asemassa, ettei ohjelmisto vioita ohjelmistokoodia versioinnin yhteydessä.

6.2 Versionhallintaohjelmistot

Versionhallintaohjelmistoja on useita erilaisia, ja ne eroavat toisistaan hintojen sekä käytettävyyden ja eri käyttöympäristöjen integroinnin osalta. Koska työ

toteutetaan Fidelix-käyttöympäristöön, ei työn toteuttamiseen vaadita kallista kuukausimaksullista ohjelmistoa suurella käyttöympäristötuella.

Järjestelmävaihtoehtoja ovat hajautettu versionhallintajärjestelmä (DVCS), keskitetty versionhallintajärjestelmä (CVCS) ja paikallinen versionhallintajärjestelmä (LVCS). Vaihtoehtoina esitellään kustakin järjestelmätyypistä yksi esimerkkiohjelmisto.

Versionhallintaohjelmistoksi valikoitui ilmainen avoimeen lähdekoodiin perustuva Git, johtuen sen suosittavuudesta, ohjelmistoversioiden päivitysten säännöllisyydestä ja hyvästä ohjeistuksesta, jota Internetistä löytyy paljon. Se on myös hajautettu versionhallintajärjestelmä, jolloin tiedostot pysyvät varmasti tallessa verrattuna esimerkiksi keskitettyyn versionhallintajärjestelmään nähden, jossa yhden ja ainoan palvelimen hajoaminen tarkoittaa kaikkien tietojen menettämistä.

6.2.1 Git

Git on Linux-käyttöjärjestelmästään tutuksi tulleen Linus Torvaldsin ja muiden ohjelmistokehittäjien vuonna 2005 luoma ilmainen ja avoimeen lähdekoodiin perustuva versionhallintaohjelmisto, joka on suunniteltu käsittelemään pieniä ja isoja projekteja. Sen etuina on helppous ja nopeus. (McMillan 2021; Atlassian s.a.)

Git on hajautettu versionhallintajärjestelmä (DVCS), jossa versiointi toteutetaan useille eri palvelimille (McMillan 2021.)

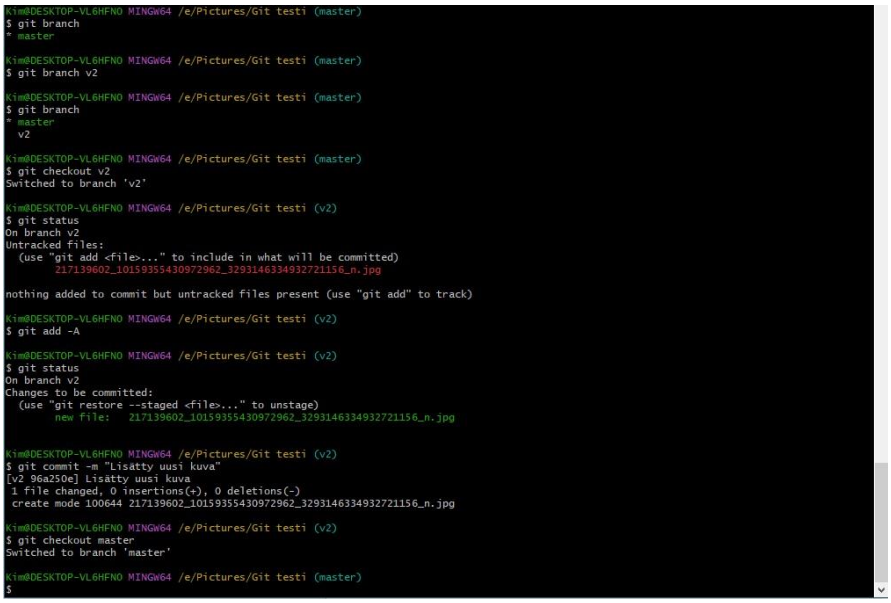
Suurista teknologiajäteistä ainakin Microsoft ja IBM käyttävät Git-ohjelmistoa (TheirStack s.a.) Lisäksi ohjelmistoon tulee useasti vuodessa uusia päivityksiä, joten käyttäjän ei pitäisi kohdata ohjelmistoa käyttäessä ongelmiin, jota ei ole korjattu (Download mirrors index 2024).

Ohjelmistosta löytyy kaksi eri versiota; Git Bash ja Git GUI. Git GUI on graafisen käyttöliittymän sisältävä versio komentopohjaisesta Git Bashista.

6.2.1.1 Git Bash

Git Bash on Gitin kevyempi versio. Sen käyttöliittymä ei turhia kaunistele, koska se pohjautuu suoraan Windowsin omaan komentokehotteeseen (kuva 4). Siinä käyttökomennot annetaan tekstikomentoina, ja se vaatii käyttäjältään hieman enemmän, koska komennot on muistettava ulkoa, jotta ohjelmiston käyttäminen olisi sujuvaa.

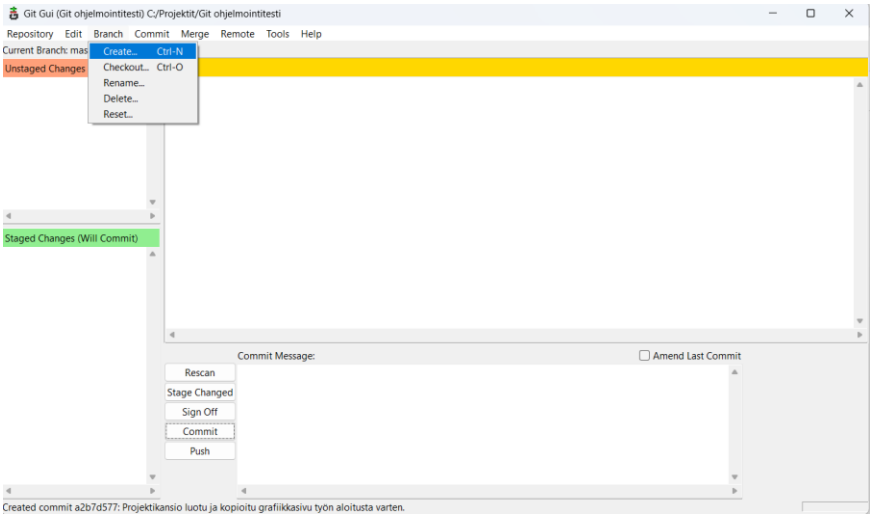
Komentopohjainen ohjelmisto on kuitenkin kokeneemmalle käyttäjälle tehokkaampi työkalu versionhallinnan käyttöön.



Kuva 4. Git Bash-käyttöliittymä

6.2.1.2 Git GUI

Gitistä löytyy myös GUI-versio, joka on graafisella käyttöliittymällä (kuva 5). Graafisen käyttöliittymän käyttö on huomattavasti käyttäjäystävällisempi, kun kaikki komennot näkyvät näytöllä erilaisien painikkeiden alla, jolloin käyttäjän ei tarvitse osata komentoja ulkoa versionhallinnan käyttämiseen.



Kuva 5. Git GUI-käyttöliittymä

6.2.2 Apache Subversion

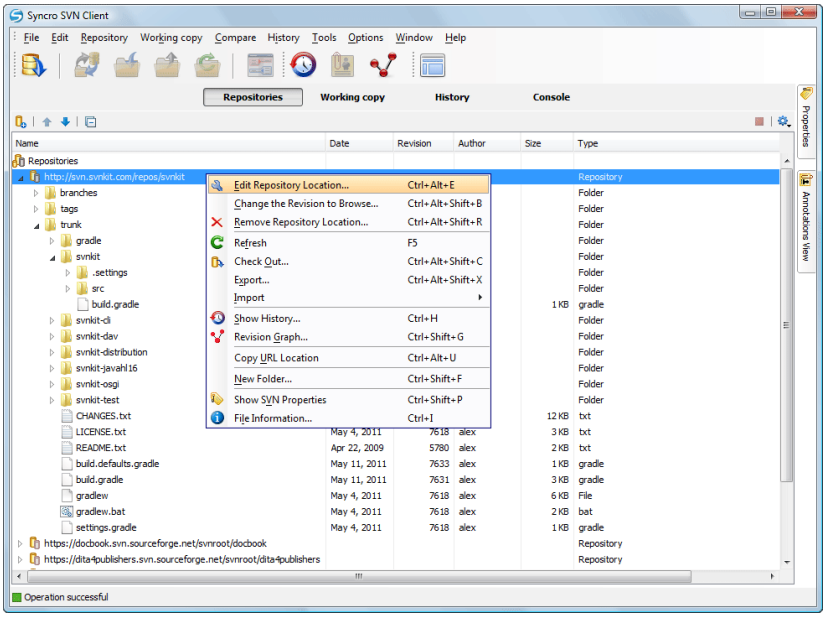
Toinen versionhallintaohjelmistovaihtoehto löytyy CollabNet-ohjelmistoyrityksestä, jonka versionhallintaohjelmisto kulkee nimellä Apache Subversion. Se on ollut kehityksessä vuodesta 2000 lähtien.

Subversionin tavoitteena on olla suurimmaksi osaksi yhteensopiva seuraaja laajasti käytetylle Concurrent Version Systems-nimiselle versionhallintaohjelmistolle, jonka ensimmäinen versio julkaistiin vuonna 1986. (McMillan 2021).

Subversion on keskitetty versionhallintajärjestelmä (CVCS), jonka toiminta perustuu yhdelle keskitetylle palvelimelle, johon versiointi toteutetaan (McMillan 2021.)

Tämäkin ohjelmisto on avoimeen lähdekoodiin perustuva ja ilmainen käyttäjälleen.

Käyttöliittymältään Subversion on GUI-pohjainen ja täten käyttäjäystävällinen vaihtoehto (kuva 6).



Kuva 6. Apachen käyttöliittymä

Viimeisin versio kirjoitushetkellä on 1.14.3, joka on julkaistu 28.12.2023. Edellinen versio ohjelmistosta on julkaistu 2022, eli ohjelmistoa päivitetään vähän hitaammin, jolloin käyttäjä saattaa kohdata toistuvasti samoja ongelmia ohjelmistoa käyttäessä, jotka korjataan vähän myöhemmin kuin esimerkiksi Gitiin nähden. (News Apache Subversion 2024).

6.2.3 Revision Control System

Revision Control System on näistä kolmesta vaihtoehdosta kaikista vanhin. Se julkaistiin jo vuonna 1982 silloin suositun Source Code Control System-versionhallintaohjelmiston kilpailijaksi. Sen kehitti alunperin Walter F. Tichy Purduen yliopistosta. Sittemmin GNU Project- ja OpenBSD Project-nimiset yhteistyöohjelmointiryhmät ovat jatkokehittäneet ohjelmistoa. (GNU RCS 2022).

Ohjelmiston pitkä ikä näkyy ohjelmiston ominaisuuksissa suuresti. Ohjelmisto ei edes mahdollista palvelimelle versiointia, koska se on paikallinen versionhallintajärjestelmä (LVCS). Edelliset mainitut versionhallintaohjelmistot tukevat useamman ohjelmoijan samanaikaisesti työskentelemisen. RCS ei tätä tue, koska ohjelmisto tukee vain yhden käyttäjän muokkausta samanaikaisesti. Lisäksi ohjelmisto on tarkoitettu yksittäisten tiedostojen versionhallintaan, eikä kokonaisien projektien versionhallintaan. (McMillan 2021.)

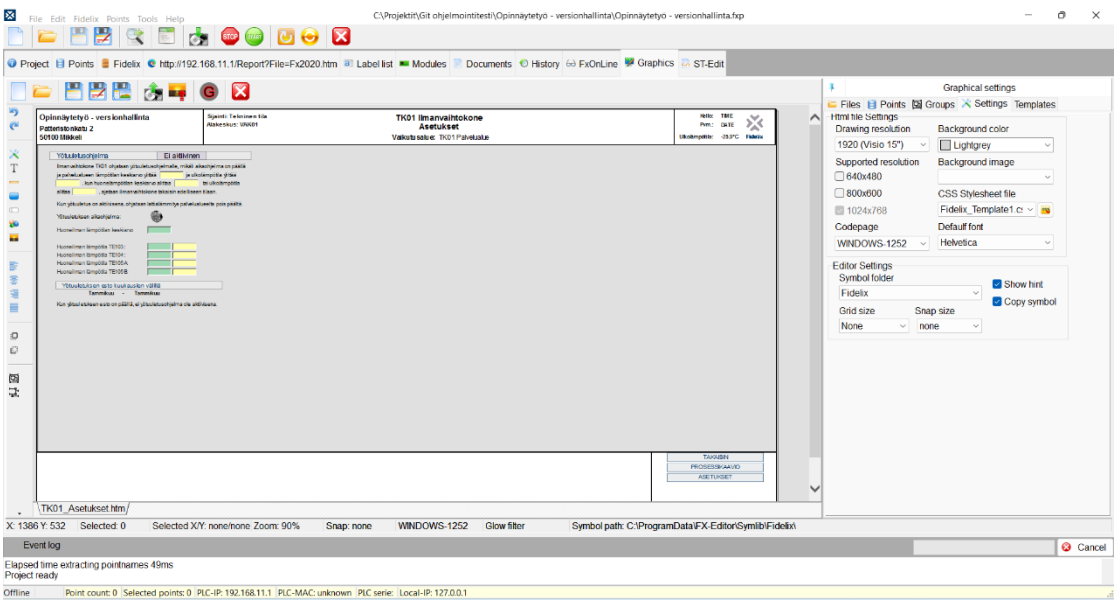
Viimeisin versio ohjelmistosta on kirjoitushetkellä julkaistu kuitenkin 2.2.2022, eli ohjelmistoa päivitetään vielä säännöllisesti (GNU RCS, 2022).

Työhön tätä ohjelmistoa ei valittu johtuen puhtaasti sen vanhuudesta sekä kömpelyydestä useiden tiedostojen käsittelyyn.

6.3 Fidelix-käyttöympäristön ohjelmistot

6.3.1 Fidelix FX-Editor

Fidelix FX-Editor on Fidelix-järjestelmille yksinomaan kehitetty projektiohjelmisto, jolla voidaan hoitaa koko Fidelix-järjestelmän ohjelmointi. Se sisältää grafiikkaeditorin, jolla piirretään grafiikkakuvat ala-asemiin. Siitä löytyy pistetunnuslistaus, jossa määritellään ala-aseman kaikki pisteet ja niiden selitetekstit ja esimerkiksi ohjausviiveet tai raja-arvot. Nämä pisteet voidaan linkittää ohjelmistossa suoraan fyysisille mittaus-, säätö-, tai ohjauskorteille, jotka nekin voidaan ohjelmistossa luoda.



Kuva 7. Fidelix FX-Editor 2.0.2.14

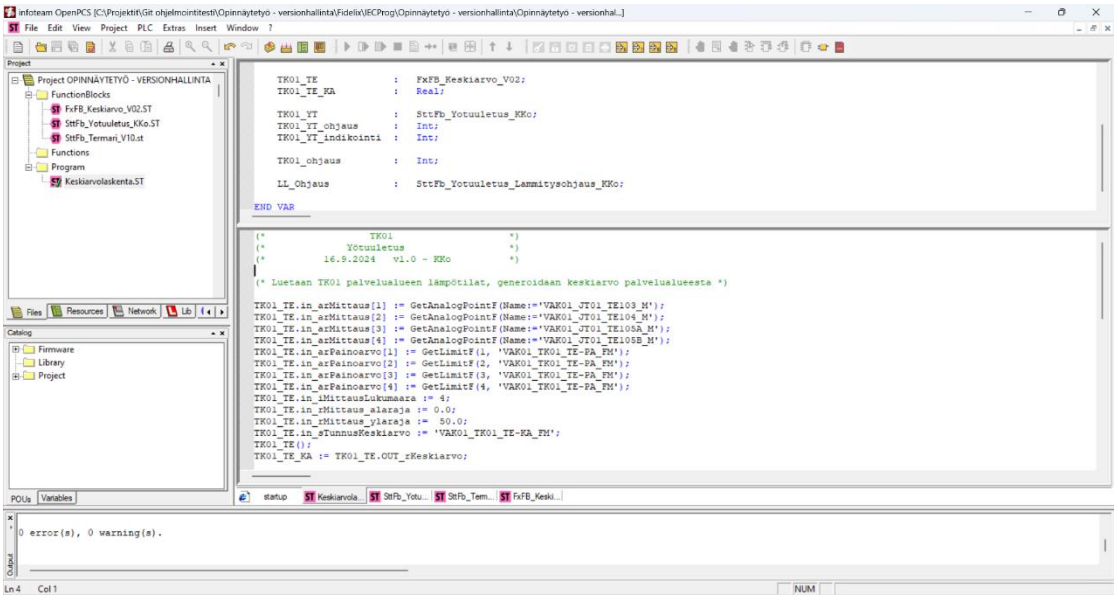
Lisäksi ohjelmistolla voidaan ladata esimerkiksi varmuuskopio ala-asemasta suoraan tai ladata ala-asemasta tiedostoja sen oman resurssienhallintaikkunan kautta.

FX-Editorista löytyy myös sisäänrakennettuna ST-ohjelmointitoiminto, joka on hieman kömpelö käyttää verrattuna seuraavaksi esiteltävään erilliseen ohjelmointityökaluun. Ohjelmointitoiminto suorittaa taustalla seuraavaksi esiteltävää OpenPCS-ohjelmistoa, joka suorittaa ohjelmakoodin kääntämisen konekielelle.

6.3.2 InfoTeam OpenPCS

Fidelix-järjestelmien ohjelmoinnissa käytetään saksalaisen infoTeamin kehittämää ohjelmointiohjelmistoa, johon on lisätty Fidelixin kehittämät laiteajurit, jotta yhteys voidaan ala-asemaan muodostaa OpenPCS-ohjelmiston kautta.

OpenPCS on IEC 61131-3-standardin mukainen ohjelmistoympäristö. Sillä voidaan ohjelmoida monia IEC-koodikieliä: ST-, IL-, SFC-, FBD-, LD-, ja CFC-kieliä. (PLC Programming Systems s.a.) Fidelix-järjestelmiä ohjelmoitaessa ei ohjelmassa tarvitse käyttää kuin tekstipohjaista ST-koodikieltä.



Kuva 8. infoTeam OpenPCS-ohjelmisto

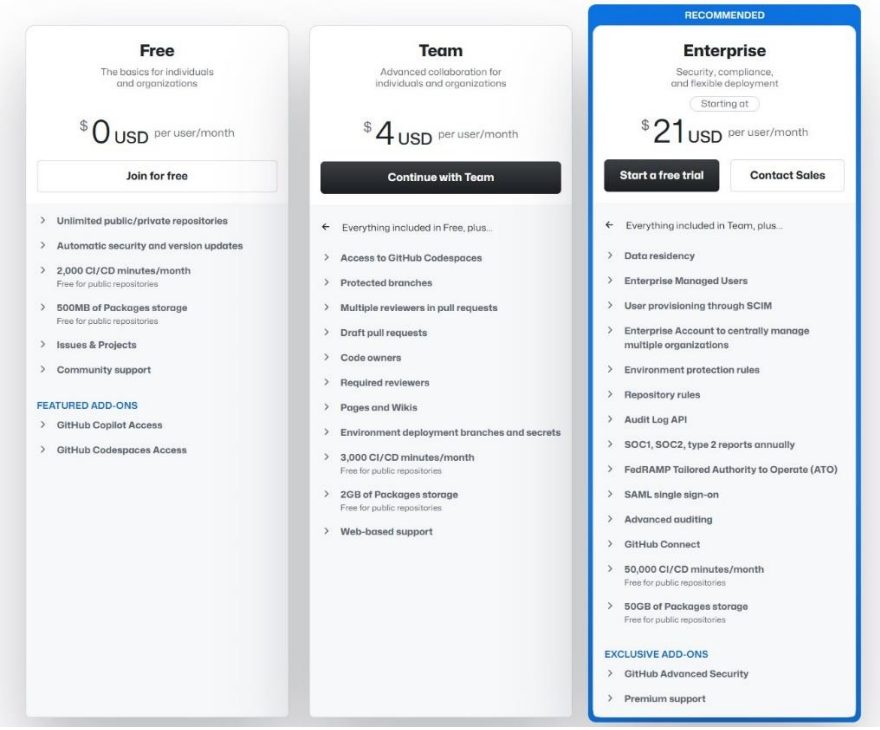
6.4 Pilvipalvelu

Versionhallintaohjelmistosta saadaan paras hyöty irti, kun sitä käytetään pilvipalvelun kautta, jolloin muutokset näkyvät reaaliajassa kaikille ohjelmoitsijoille ja uusimmat haarat ovat jokaisen ulottuvissa.

Työssä oli alun perin ajatuksena hyödyntää yrityksen tarjoamaa jo käytössä olevaa Microsoft OneDriveä, mutta kun selvisi se, että OneDrive ei ole kovin hyvä palvelu versionhallinnan pilveen versioinnin toteuttamiseen johtuen sen tavasta käsitellä tiedostoja päädyttiin valitsemaan palveluksi erittäin suosittu GitHub. Toinen hyvä vaihtoehto olisi ollut myös BitBucket.

6.4.1 GitHub

GitHub on ilmainen ohjelmistokehittäjien alusta, johon voidaan ladata ilmaiseksi ladattavia ohjelmistoja. Se on ilmainen tiettyyn pisteeseen saakka, mutta ilmaisversion ominaisuudet riittävät tämän työn testaamiseen hyvin. Työkäytössä maksullinen versio olisi kuitenkin hyvä ilmaisversion rajoituksien takia.



Kuva 9. GitHub. Lisenssivaihtoehdot (GitHub 2024)

Ilmaisversio mahdollistaa projektin jakamisen kolmen ohjelmoijan kesken ja 2000 minuuttia toimintojen tekemiseen kuukaudessa. Mikäli jakaminen pitää saada useammalle, vaaditaan Team-jäsenyys, joka maksaa 4 dollaria kuussa. Lisäksi sillä saa suojauksen versiointiin, mikäli jokin kriittinen virhe sattuisi sekä 3000 minuuttia toimintojen tekemiseen. Vielä suuremmalla 21 dollarin maksulla saa muita ominaisuuksia, kuten haarojen käyttöoikeuksien suojauksen. Versioinnin suojausominaisuutta ilmaisversio ei tarjoa, eli mikäli pilveen

ajetaan viallinen versio, on muutokset peruuttamattomia ja myöskään haaroja ei voida suojata. Projektikansion maksimitallennusrajaa ei GitHubilla ole. (GitHub pricing 2024.)

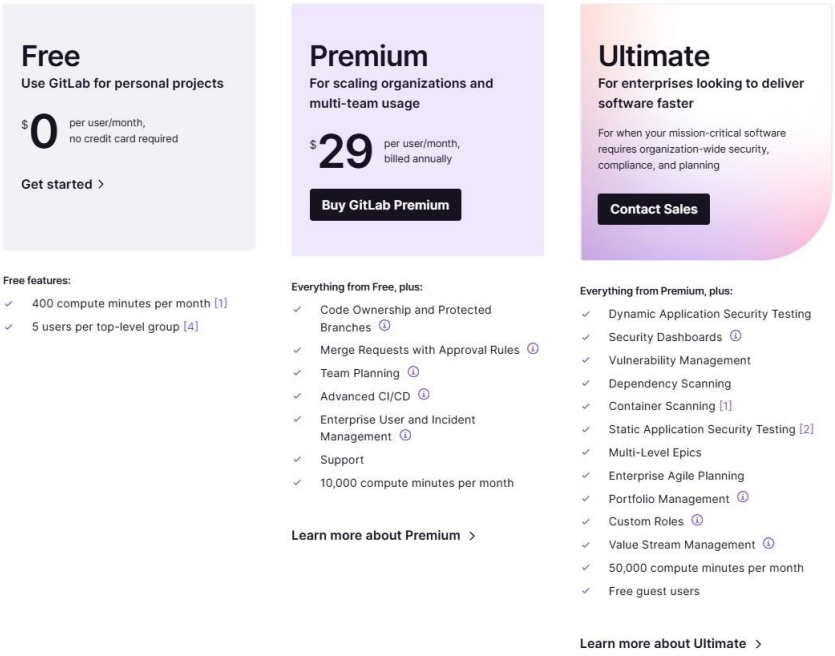
Toiminnallisuuden kannalta ajatellen GitHub on kaikista loogisin vaihtoehto versionhallinnan pilvipalveluksi. Se on kevyt, nopea, sekä edullinen ja sisältää hyvän integraation itse versionhallintaohjelmiston kanssa.

6.4.2 GitLab

Toinen vaihtoehto Gitin versionhallinnan pilvihakemiston toteuttamiseen on GitLab. Se on Git-pohjaiseen versionhallintaan sopiva palvelu ja tukee useista eri lähteistä koodin tuomisen.

Ilmaisversio on hyvin rajoittunut GitHubiin nähden, koska se tarjoaa ilmaiskäyttäjälle vain 400 minuuttia aikaa kuukaudessa toimintojen tekemiseen. Ilmaisversio mahdollistaa kuitenkin projektin jakamisen viiden ohjelmoijan kesken. Mikäli aikaa ja jakamisoikeuksia tarvitaan enemmän, seuraava vaihtoehto ilmaisjäsenyyden jälkeen onkin 29 dollaria kuussa maksava Premium-jäsenyys. Se on siis huomattavasti kalliimpi vaihtoehto GitHubiin nähden. Projektikansion maksimitallennusraja GitLabissa on 10 gigatavua. (GitLab pricing 2024.)

Ilmaisversion ominaisuudet kuitenkin riittäisivät työn testaamiseen varsin hyvin ja saattaisivat jopa riittää myös työkäytössä, mikäli jakamista ei tarvitse suorittaa useille henkilöille.



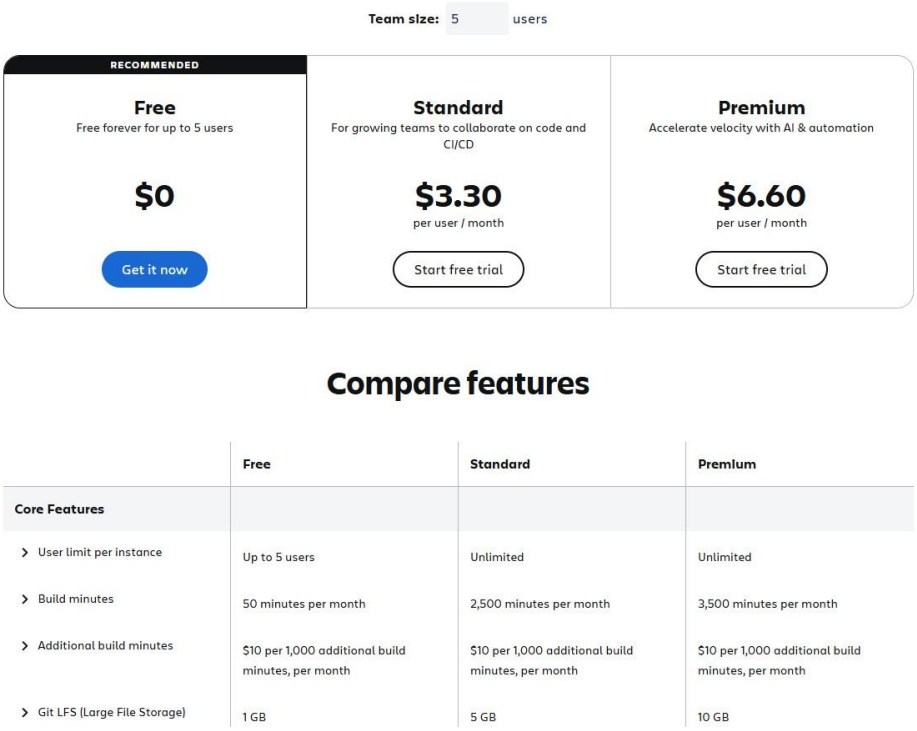
Kuva 10. GitLab. Lisenssivaihtoehdot (GitLab 2024)

6.4.3 BitBucket

BitBucket on toisiksi edullisin vaihtoehto Gitin pilvihakemistojen toteuttamiseksi. Se tarjoaa käyttäjälleen rajattomat piilotetut projektit, joiden jokaisen tallennusraja on jo ilmaisversiollakin yhden gigatavun verran.

Ilmaisversio tarjoaa jakamisen piilotetuille projekteille maksimissaan viiden ohjelmoijan kesken. Toimintoaika kuukaudessa on vain 50 minuuttia ilmaisversiossa, eli palvelu käytännössä pakottaa hankkimaan Standard-jäsenyyden, jolla aikaa saadaan 2500 minuuttia per kuukausi sekä rajattoman käyttäjämäärän per projekti. Sen hinta on kuitenkin yhdellekin käyttäjälle vain 16,5 dollaria, eli huomattavasti huokeampi kuin edellisten palveluiden halvin maksullinen jäsenyys. (BitBucket pricing 2024.)

Ilmaisversio riittäisi työn testaamiseen, mutta työkäytössä Standard-jäsenyys olisi pakollinen.



Kuva 11. BitBucket. Lisenssivaihtoehdot (BitBucket 2024)

6.4.4 Microsoft OneDrive

OneDrive on todennäköisesti käytetyin pilvipalvelu yrityskäytössä johtuen helppokäyttöisyydestä ja hyvästä integroinnista käytettyjen Microsoft Windows -käyttöjärjestelmien kanssa.

Sen sopivuus projektinhallintaan on todettu kuitenkin huonoksi, jos kansiorakenteet sisältävät paljon tiedostoja. Kansiot itsessään siirtyvät hyvin pilveen, mutta itse kansion sisältämät tiedostot eivät siirry pilveen välttämättä lainkaan ja se on myös erittäin hidas kopioimaan tiedostoja pilveen. Työn aloitusvaiheessa projektien siirto pilveen on toteutettu pakkaamalla kansiot pakkausohjelmalla kuten esimerkiksi 7Zipillä, jolloin tiedostot pysyvät varmasti kansiossa tallessa ja kansion lataus onnistuu nopeasti.

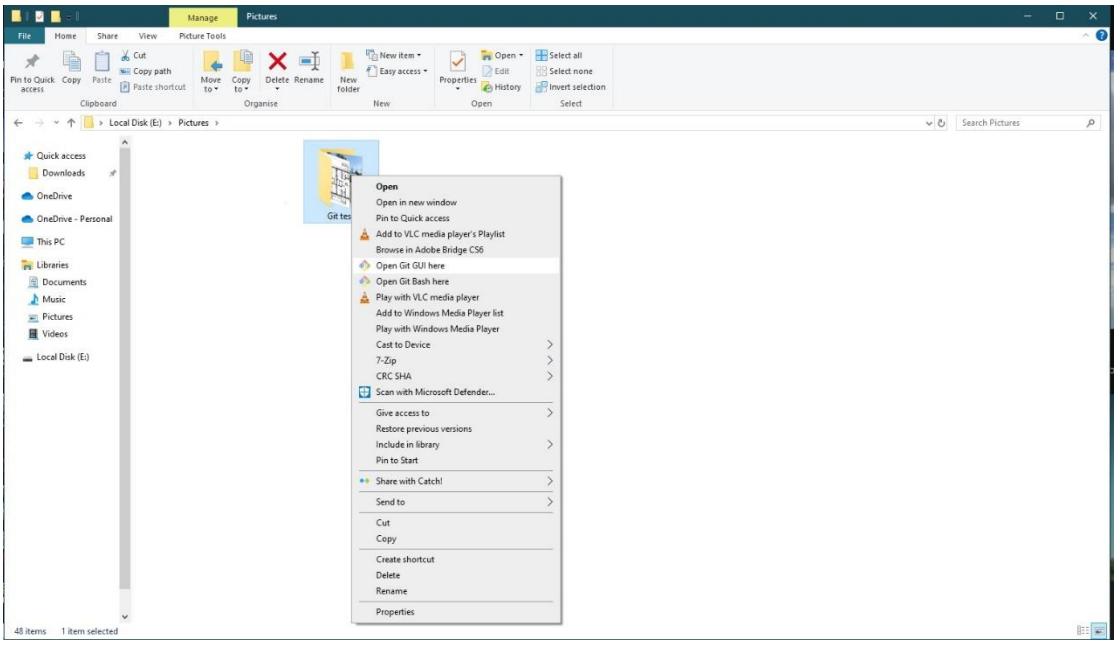
Useat lähteet kertovat, että OneDrive ei sovellu kovin hyvin Gitin pilvikansion säilytykseen johtuen sen tavasta käsitellä tiedostoja, mikäli tiedostoja ei käytetä pitkään aikaan ja varsinkin mikäli kyseessä on suuret tiedostomäärät. Lisäksi OneDriven epämääräinen oma tiedostosynkronointi ei mahdollista useamman ohjelmoitsijan yhtäaikaista työskentelyä ilman, että versiointi rikkoutuu. (Dustin Briles 2022; Rahul Bakale 2023, @Helloall_16 2024.)

7 VALMIIN TYÖN TESTAUS JA KÄYTTÖÖNOTTO

Versionhallintaohjelmiston käyttöönotto on hieman työlästä. Kun Git-ohjelmisto on ladattu, ei se tarjoa käyttäjälleen suoraan käyttöapua, saati mitään ohjeita miten versionhallintaa käytetään. Gitin mukana tuleva ohjeistus kertoo vain käytössä olevat komennot, eikä mitään esimerkiksi siitä, missä järjestyksessä käyttäjän kannattaisi ohjelmistoa käyttää. Työ testataan käyttämällä Git GUI:ta johtuen sen helposta lähestyttävyydestä selkeän käyttöliittymänsä takia verrattuna komentosarjapohjaiseen Bashiin.

7.1 Hakemiston luonti

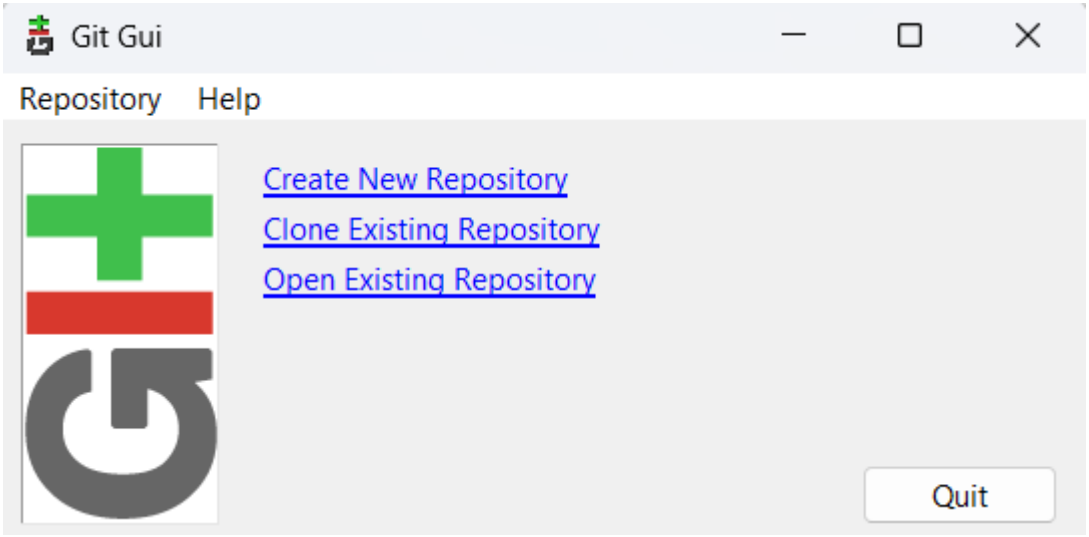
Versionhallinnan käyttöönoton ensimmäisenä vaiheena luodaan projektihakemisto, jonka muutoksia Git seuraa. Git GUI on todella helppo ottaa käyttöön halutussa kansiossa. Riittää, että siirtyy Windowsin resurssienhallinnassa haluttuun kansiorakenteeseen, jolloin halutun kansion kohdalla hiiren oikeaa painiketta painamalla aukeaa valikkorakenne, johon Git lisää itse Open Git GUI here-komennon (kuva 12). Vaihtoehtoisesti voi sovelluksen avata suoraan, jolloin Git avaa käynnistysvalikkonsa (kuva 13).



Kuva 12. Git GUI. Versionhallinnan avaaminen kansiohakemistoon

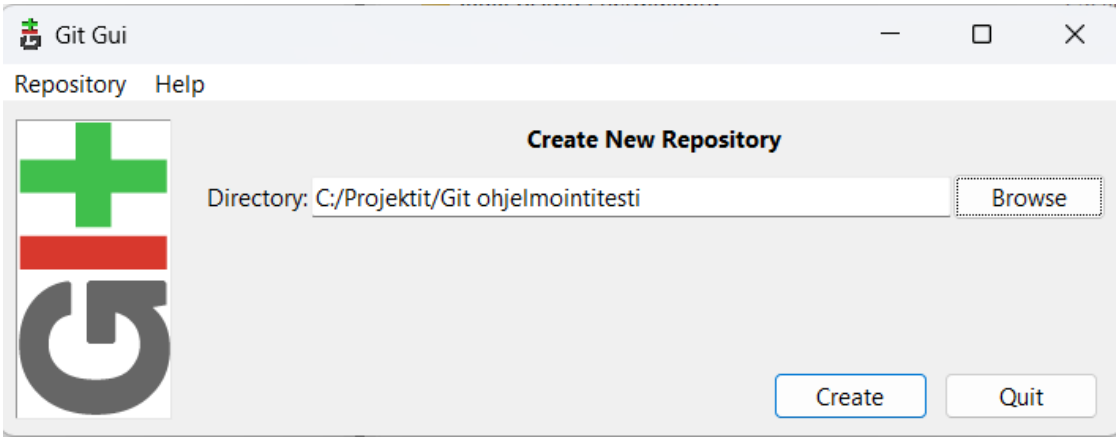
Versionhallinta kysyy seuraavana hakemistoa, johon versionhallinta otetaan käyttöön tai vaihtoehtoisesti voidaan avata jo olemassa oleva hakemisto (kuva 11). Koska versionhallintaa ei ole vielä otettu käyttöön kyseisessä kansiossa,

on se määriteltävä uudeksi tietovarastoksi (englanniksi Repository) painamalla Create New Repository -painiketta.



Kuva 13. Git GUI. Aloitusikkuna

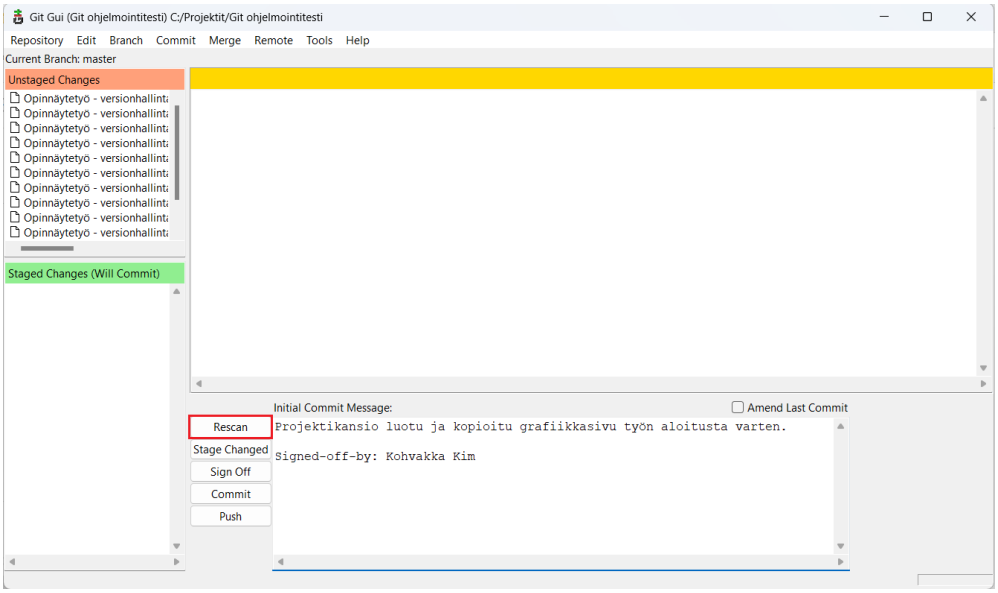
Kun tietovarasto on luotu, alkaa Git seuraamaan kansion sisällä olevia tiedostoja ja niiden muutoksia.



Kuva 14. Git GUI. Tietovaraston luonti

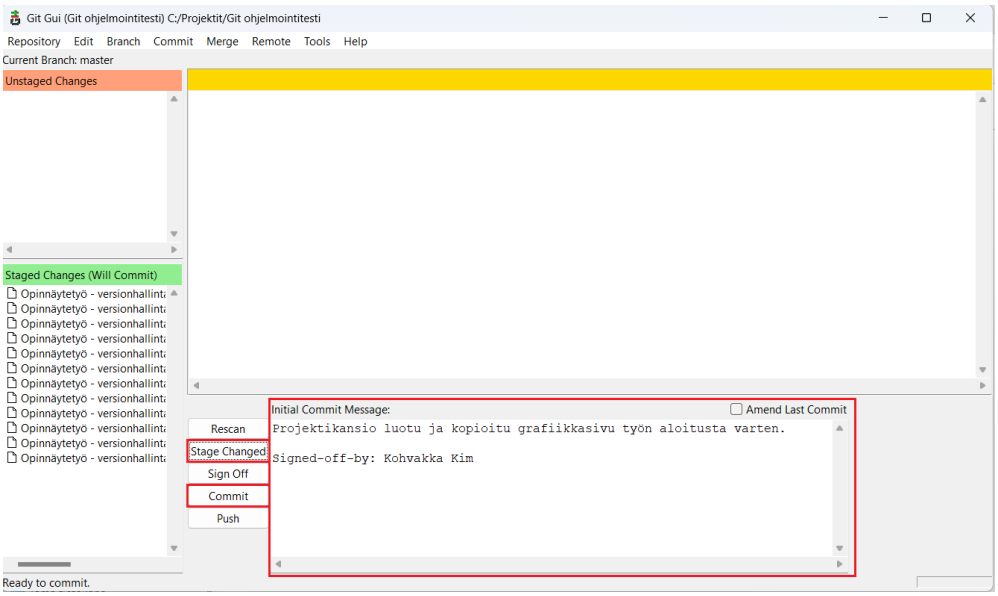
7.2 Haaran luonti

Seuraavaksi lisätään kansioon tarvittavia Fidelix-järjestelmän tiedostoja työn aloittamista varten. Tiedostot tulevat näkymään näkyviin, kun teksti-ikkunan vieressä olevaa Rescan -painiketta painetaan (kuva 15).



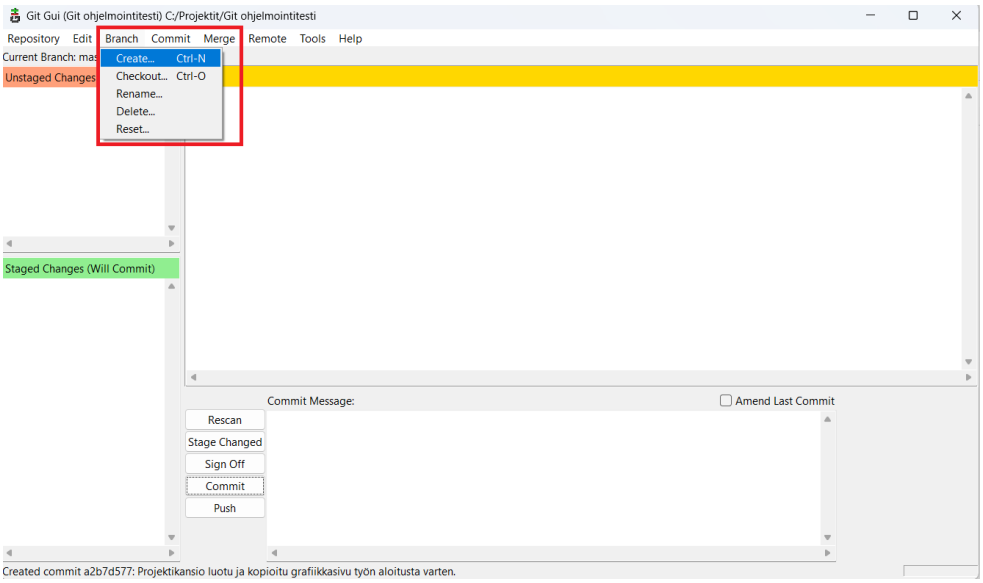
Kuva 15. Git GUI. Kansion sisällä tehty muutoksia, joita ei ole vielä tallennettu kyseiseen haaraan

Kun tarvittavat muutokset kansioon on tehty (kuva 16), voidaan muutokset tallentaa Stage Changed -painiketta painamalla. Tämän jälkeen on suositeltavaa kirjoittaa Initial Commit Message -ikkunaan seliteteksti, jotta on helppo pysyä perässä mitä projektiin tässä vaiheessa on tehty. Kun seliteteksti on kirjoitettu ja tiedostomuutokset valmiita, voidaan työ tallentaa painamalla Commit -painiketta painamalla. Nyt projektista on tehty ensimmäinen master-haaraversio, joka on tallennettu paikallisesti tietokoneelle.



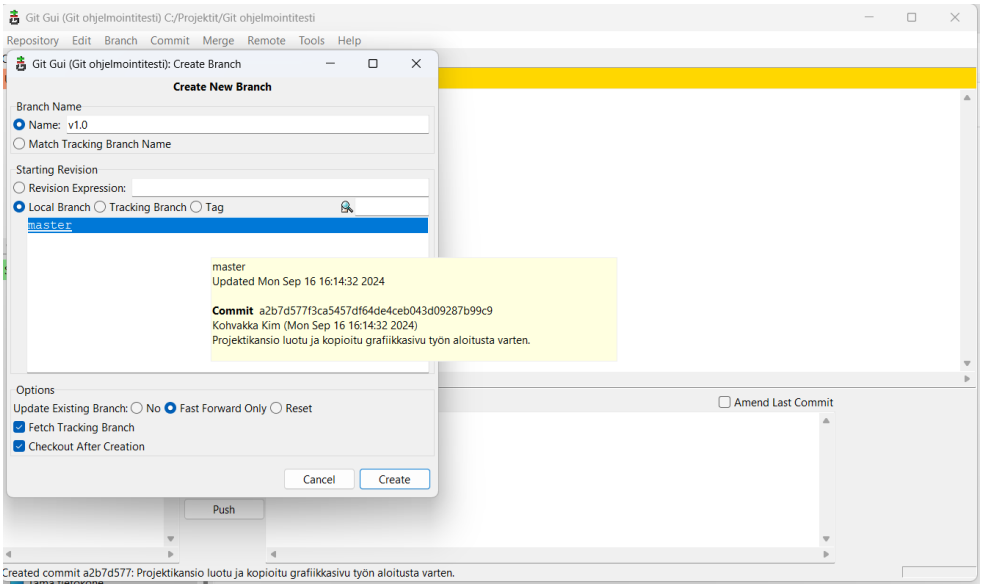
Kuva 16. Git GUI. Kansion sisällä tehty muutokset, jotka ovat tallentumassa paikallisesti nykyiseen haaraan

Seuraavaksi luodaan uusi haara (englanniksi branch) seuraavia muutoksia varten yläpalkista löytyvän Branch-valikon Create -painikkeen avulla (kuva 17).



Kuva 17. Git GUI. Uuden haaran luominen

Branch-valikossa määritellään, mitä haaraa uusi haara mukailee eli minkä haaraversion uusi haaraversio ottaa pohjakseen. Tässä ikkunassa (kuva 18) nähdään aikaisemmin jo tallennettu master-haara ja sen seliteteksti. Annetaan haaralle haluttu nimi kohtaan ”Name” ja valitaan aikaisempi paikallinen master-haara pohjaksi Local Branchista ja painetaan Create -painiketta.



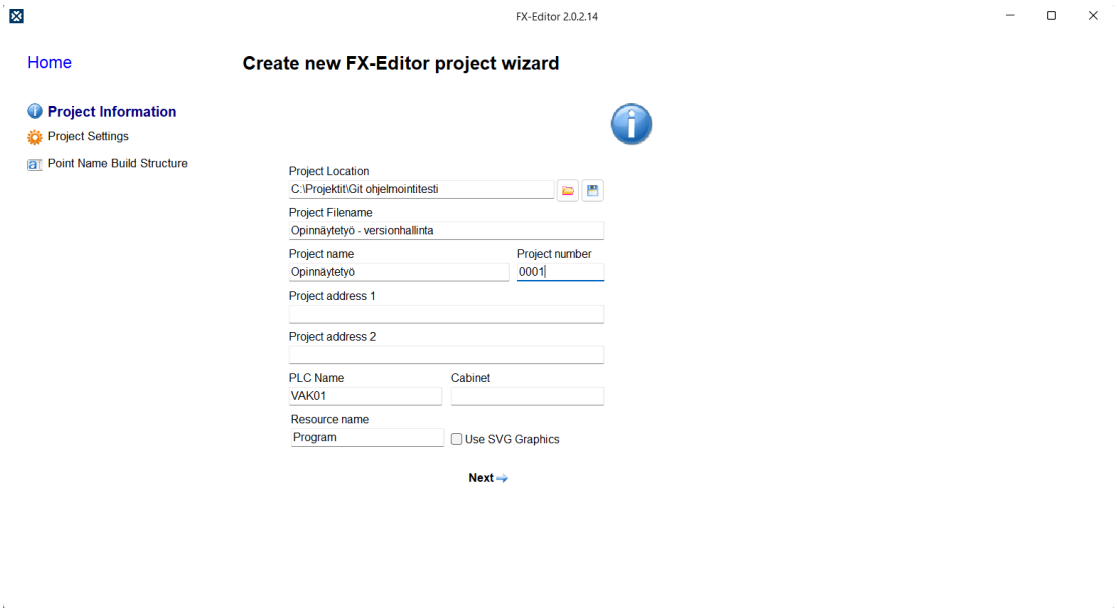
Kuva 18. Git GUI. Haaran luontivalikko

7.3 Versionhallintaohjelmiston testaus Fidelix-järjestelmässä

Nyt kun kansiorakenne on luotu ja tarvittavat tiedostot lisätty projektia varten sekä otettu Git-ohjelmisto kansiorakenteessa käyttöön ja tehty ensimmäinen haara, voidaan luoda Fidelix-käyttöympäristön vaatima projekti.

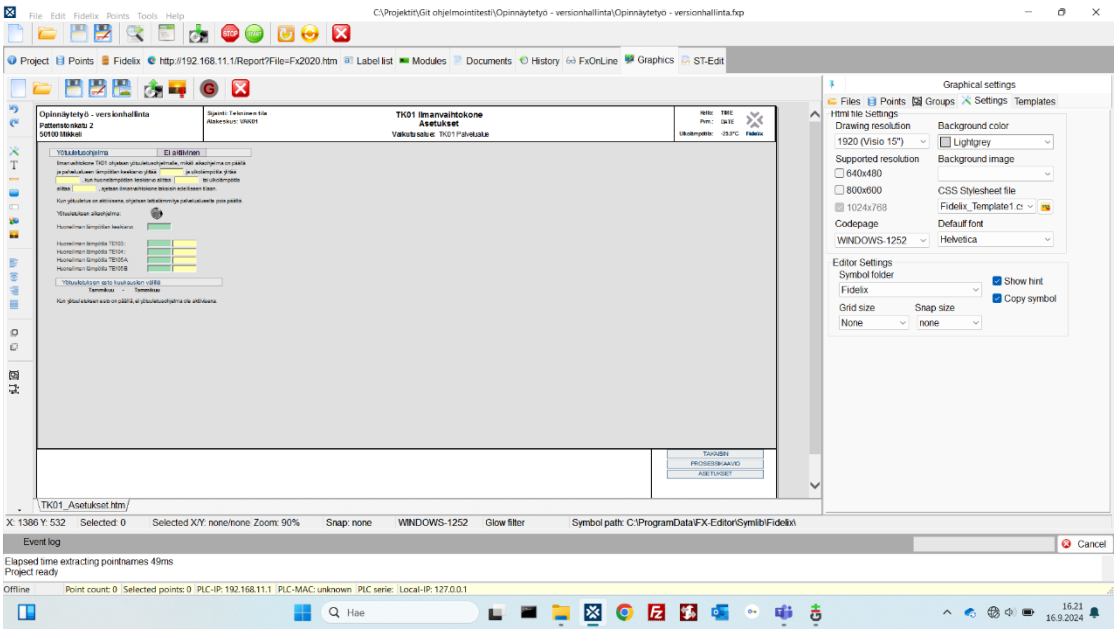
Projektityönä versionhallinnan testaamiseksi luodaan yötuuletusohjelma IV-koneeseen. IV-koneen palvelualueen huonelämpötiloja mitataan, ja näistä mittauksista lasketaan ohjelmallinen keskiarvo, joka ohjaa IV-koneen yötuuletusta palvelualueen huonelämpötilan keskiarvon ja ulkolämpötilan ollessa määritettyjen sallittujen rajojen sisässä.

Ensimmäiseksi luodaan uusi FX-Editor-projekti aikaisemmin luotuun kansiohakemistoon (kuva 19).

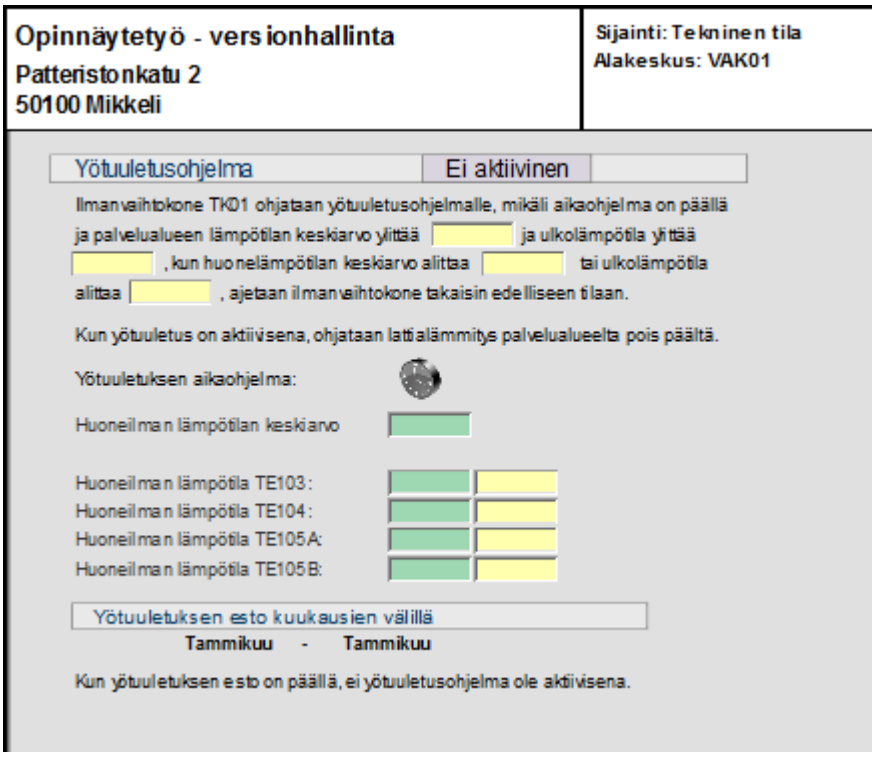


Kuva 19. Fidelix FX-Editor. Projektin luontivalikko

Seuraavana luodaan FX-Editorilla IV-koneen asetussivu, jossa näkyvät ulkolämpötila, palvelualueen keskiarvolämpötila, huonelämpötilamittaukset sekä niiden ohjelmalliset painoarvot laskennallista keskiarvoa varten sekä ohjelmalliset rajat yötuuletuksen aktivoimiseksi. Työhön kopioitiin grafiikkakuva edellisistä projekteista, johon tehtiin tarvittavat muutokset opinnäytetyön suorittamista varten.

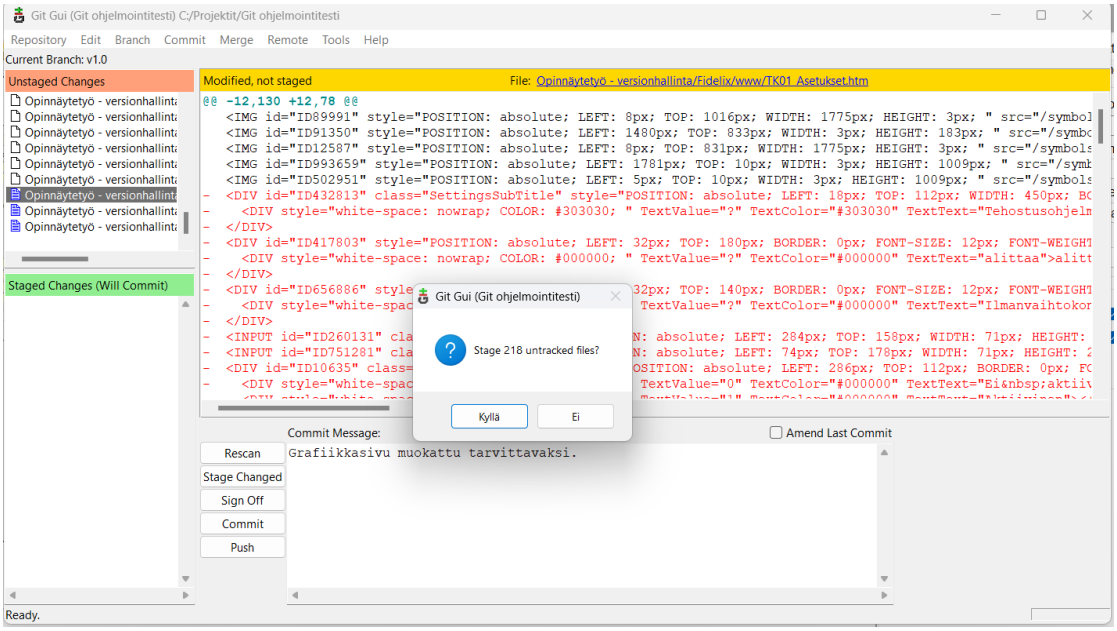


Kuva 20. Fidelix FX-Editor. Grafiikkaeditori



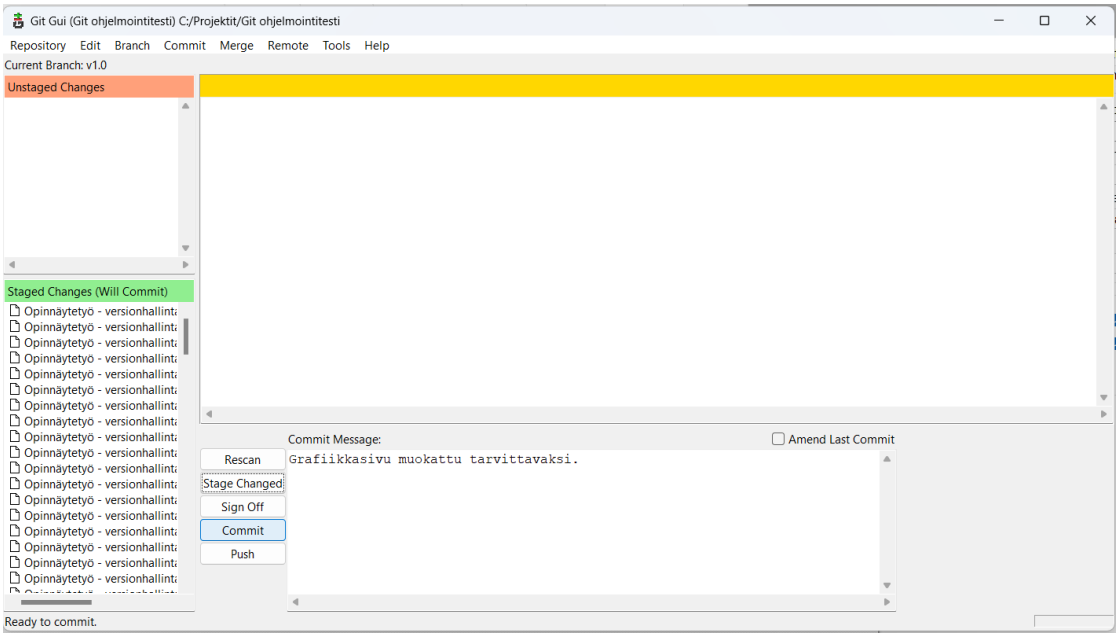
Kuva 21. FX-Editorissa luotu valmis grafiikkasivu

Seuraavana avataan Git ja skannataan kansio, kuten aikaisemmin tehtiin. Pelkästään grafiikkakuvaan koskemisesta koituu paljon muutoksia kansiorakenteen sisältämiin tiedostoihin, kuten kuvasta 22 nähdään.



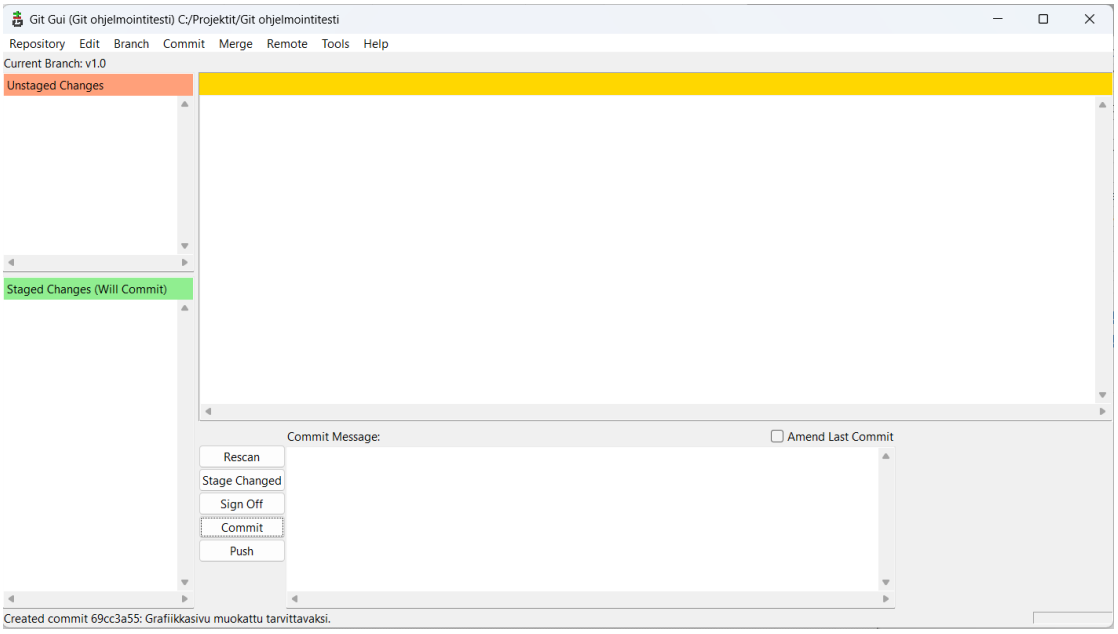
Kuva 22. Git GUI. Grafiikan muokkaamat tiedostot

Kirjoitetaan taas viesti-ikkunaan tarvittava seliteteksti, ja ajetaan muutokset nykyiseen v1.0-versioon Commit -painikkeen kautta (kuva 23).



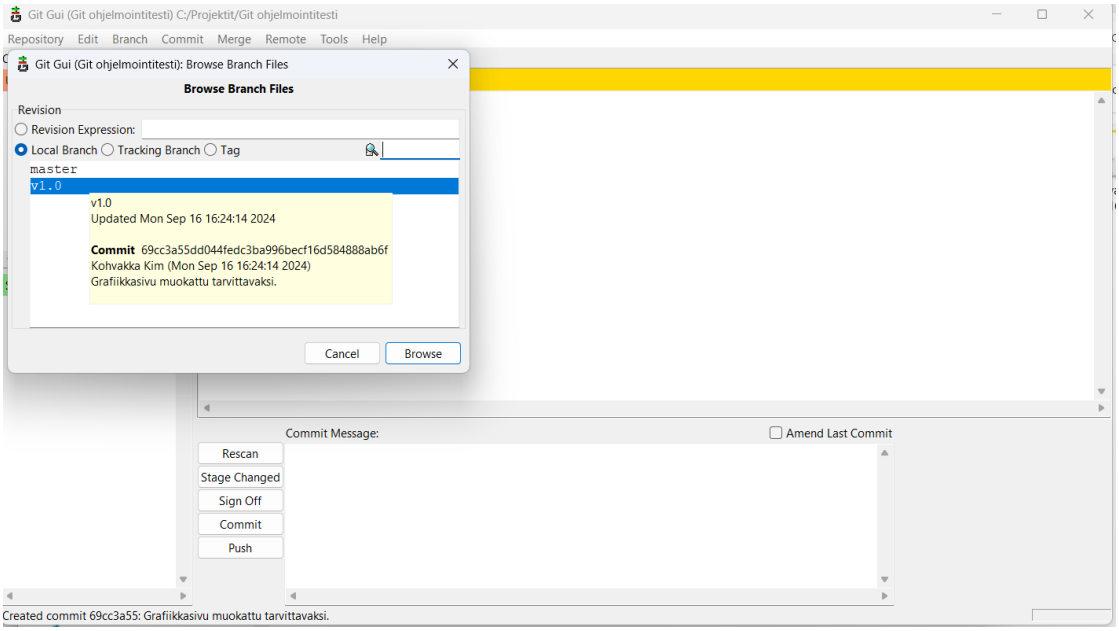
Kuva 23. Git GUI. Grafiikkakuvan tiedostot menossa tallennettavaksi haaraan selitetekstin kera

Kun tiedostot ovat ajettu haaraan, ne häviävät ikkunasta kokonaan, kunnes projektiin taas tehdään muutoksia ja skannataan kansiorakenne uudelleen (kuva 24).



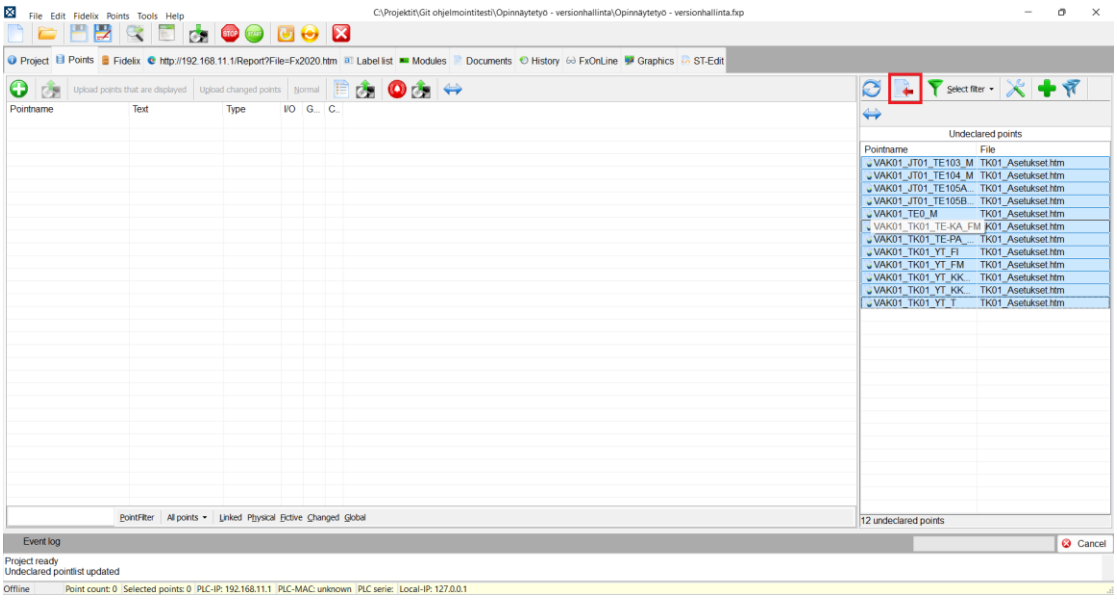
Kuva 24. Git GUI. Tiedostot tallennettu versioon 1.0

Seuraavaksi tarkastetaan, että juuri luotu ja tallennettu versio 1.0 on näkyvissä haaraikkunassa (kuva 25).



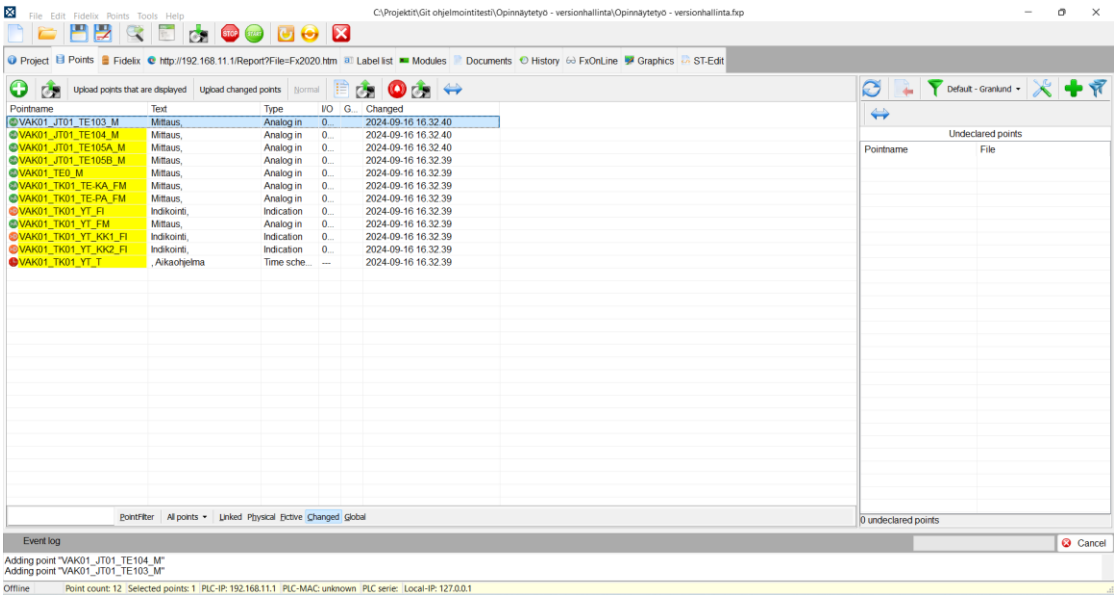
Kuva 25. Git GUI. Haaraikkuna version 1.0 jälkeen

Seuraavaksi luodaan FX-Editorissa pistetunnukset ohjelmointia varten. Näillä pistetunnuksilla IEC-ohjelmakoodi kysyy valvonta-ala-asemasta tietoja. Tuodaan grafiikalta pistetunnukset painamalla oikeassa reunassa olevaa painiketta (kuva 26).



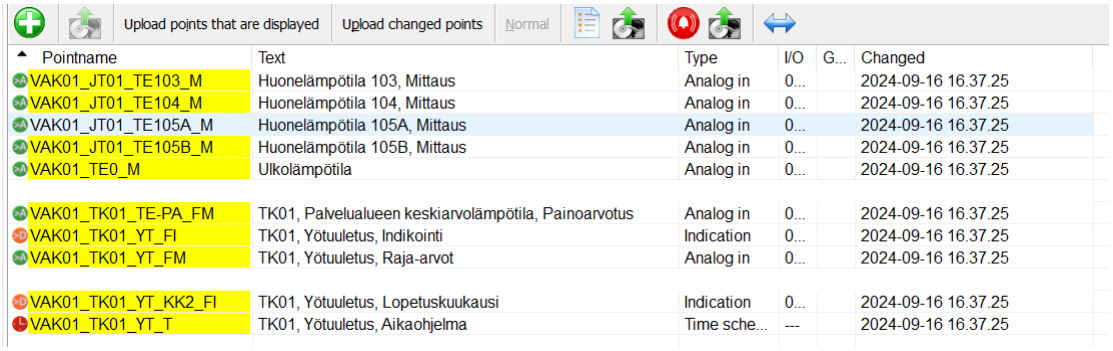
Kuva 26. Fidelix FX-Editor. Pistetunnuksien tuonti grafiikalta

Kuvasta 27 nähdään, että pisteet ovat siirtyneet nyt pistetunnussivulle, ja niiden selitetekstit ovat automaattisesti generoituneita. Mikäli Import-filtteriin olisi määritelty tarvittavat selitetekstit, tulisivat ne automaattisesti. Koska tähän filtteriin ei ollut määritelty pistetunnuksia tarkemmin, on ne tehtävä käsin.



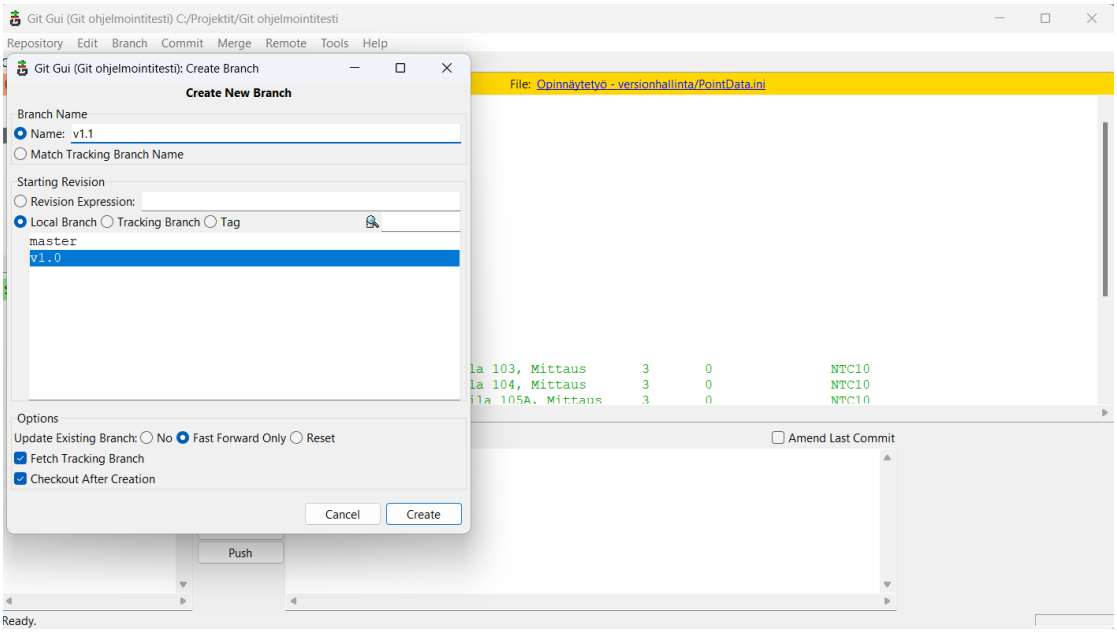
Kuva 27. Fidelix FX-Editor. Tuodut pistetunnukset

Valmiiden pistetunnuksien pitäisi kertoa selkeästi mitä ne ovat ja mikä niiden tarkoitus on kuten kuvasta 28 nähdään.



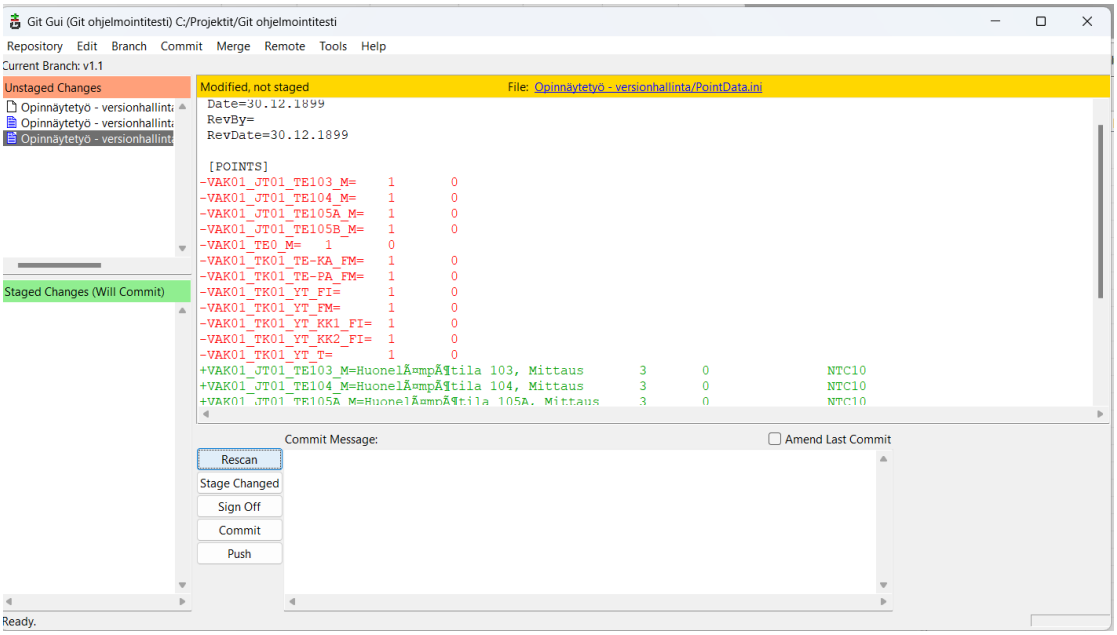
Kuva 28. Fidelix FX-Editor. Valmiit pistetunnukset

Nyt kun pistetunnukset on luotu, voidaan taas tehdä uusi versio projektista. Tällä kertaa annetaan versionumeroksi 1.1 (kuva 29).

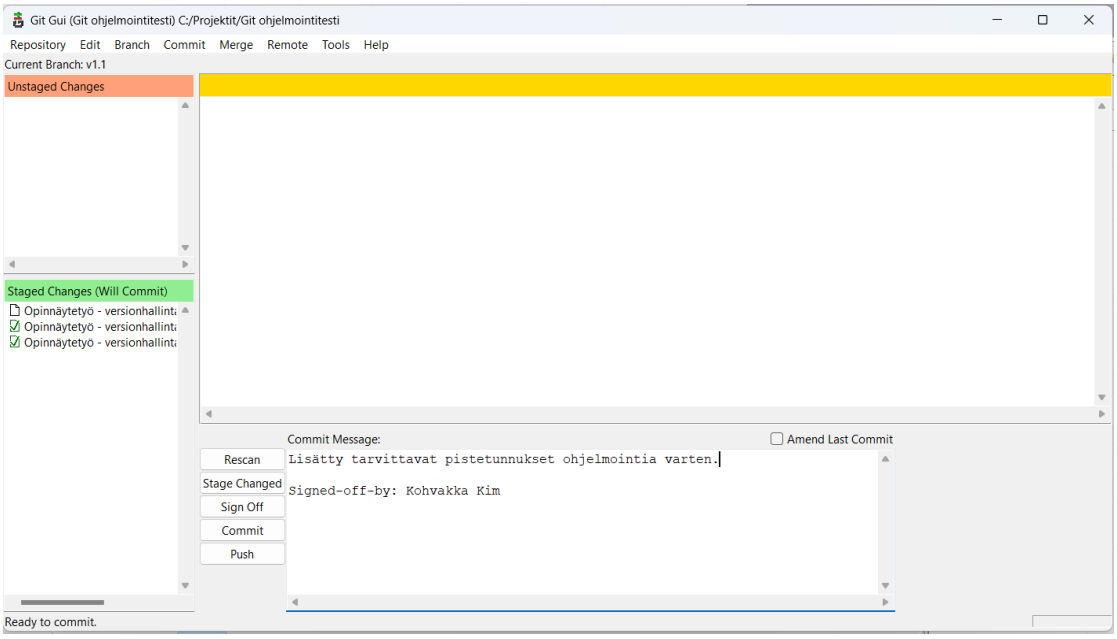


Kuva 29. Git GUI. Version 1.1 luominen

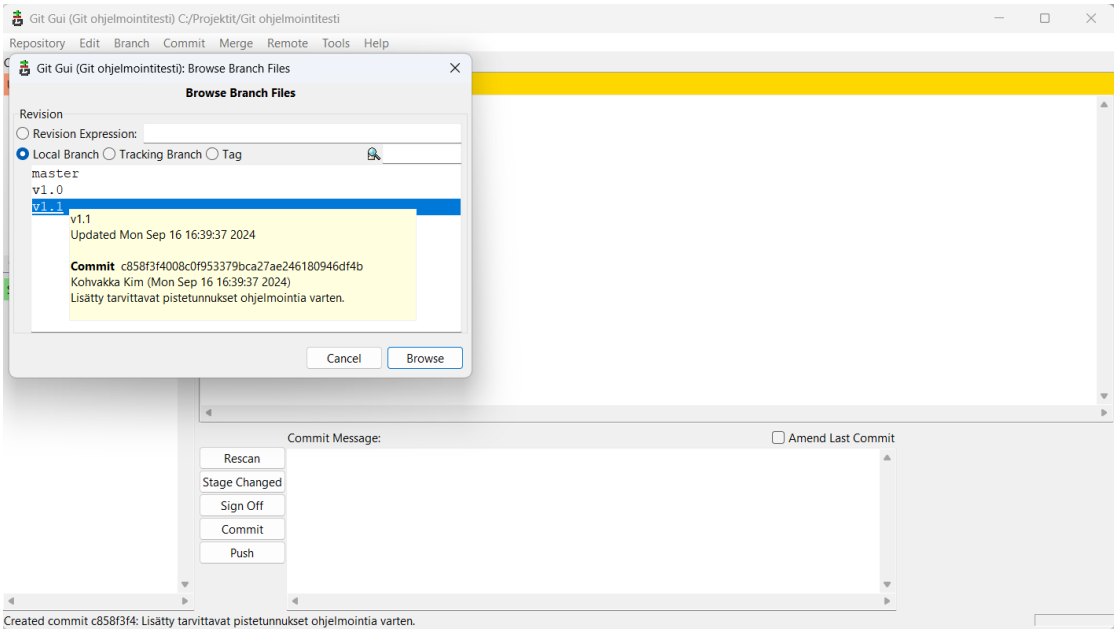
Käytetään tällä kertaa master-haaran sijasta versiota 1.0, jotta projektin aikaisemmin tehdyt muutokset tulevat uuteen haaraan mukaan. Skannataan taas kansiorakenne uudelleen ja tallennetaan muuttuneet tiedostot ja ajetaan ne versioon 1.1 (kuvat 30 ja 31).



Kuva 30. Git GUI. Version 1.1 tallentamattomat muutokset



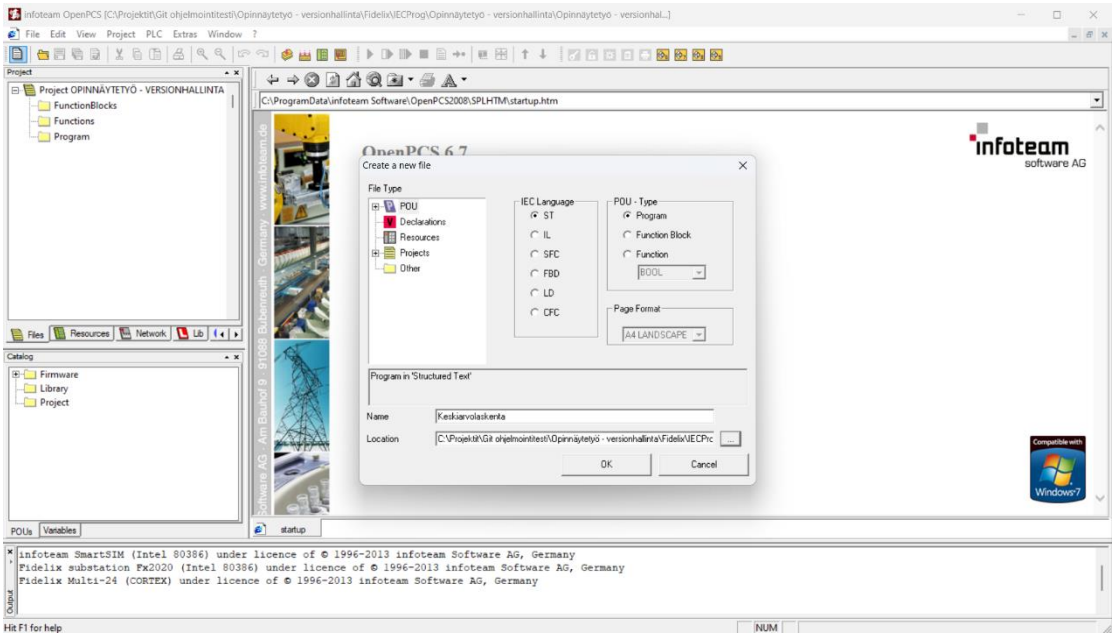
Kuva 31. Git GUI. Version 1.1 tallennukseen menevät muutokset



Kuva 32. Git GUI. Haaraikkuna version 1.1 jälkeen

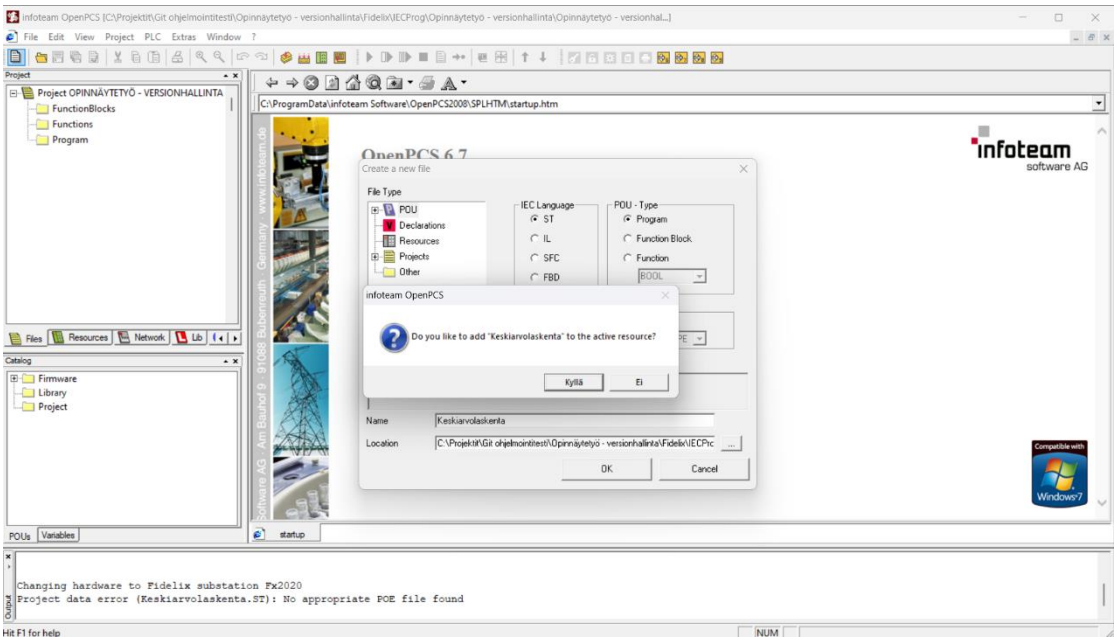
Nyt, kun grafiikkasivu sekä pistetunnukset ovat valmiita ja molemmat tallennettu omina versioinaan, voidaan aloittaa itse ohjelmointi käyttäen apuna infoTeam OpenPCS:ää.

Fidelix-järjestelmä käyttää IEC-kielenä ST-kieltä, joten valitaan se kieleksi. Koska kyseessä on yksi ohjelmatiedosto, jota ei tarvita projektissa useampaan otteeseen, valitaan POU-tyypiksi ”Program” (kuva 33).



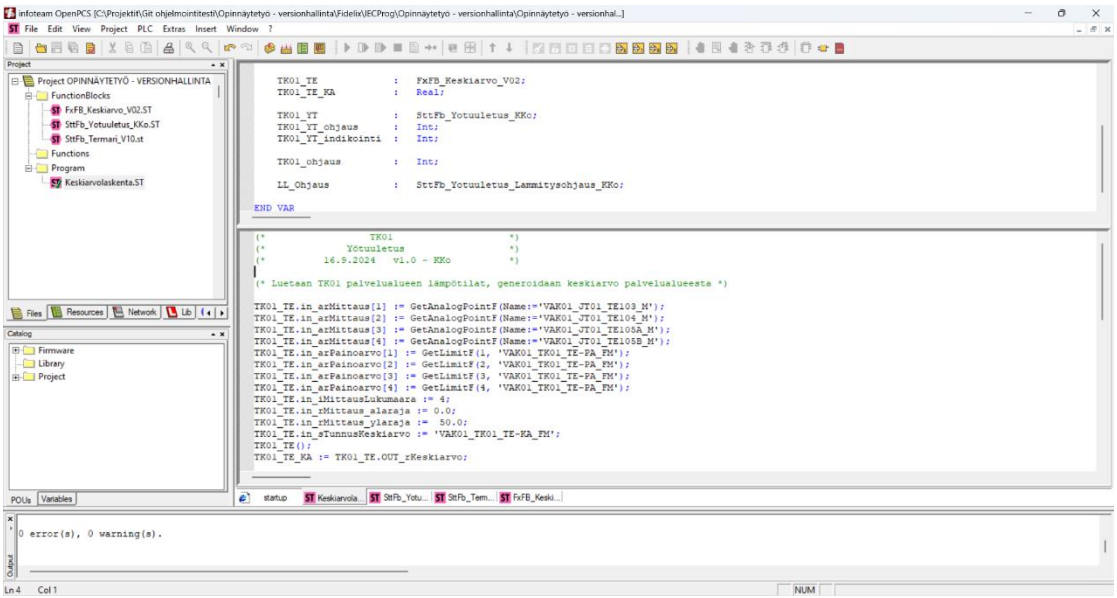
Kuva 33. OpenPCS. Ohjelmistokoodin tiedoston luonti

Lisätään juuri luotu ohjelmakoodi nykyiseen resurssiin, jotta muutokset voidaan ladata ala-asemaan, kun ohjelmakoodi on valmis (kuva 34).



Kuva 34. OpenPCS. Ohjelmakoodin lisäys aktiiviseen resurssiin

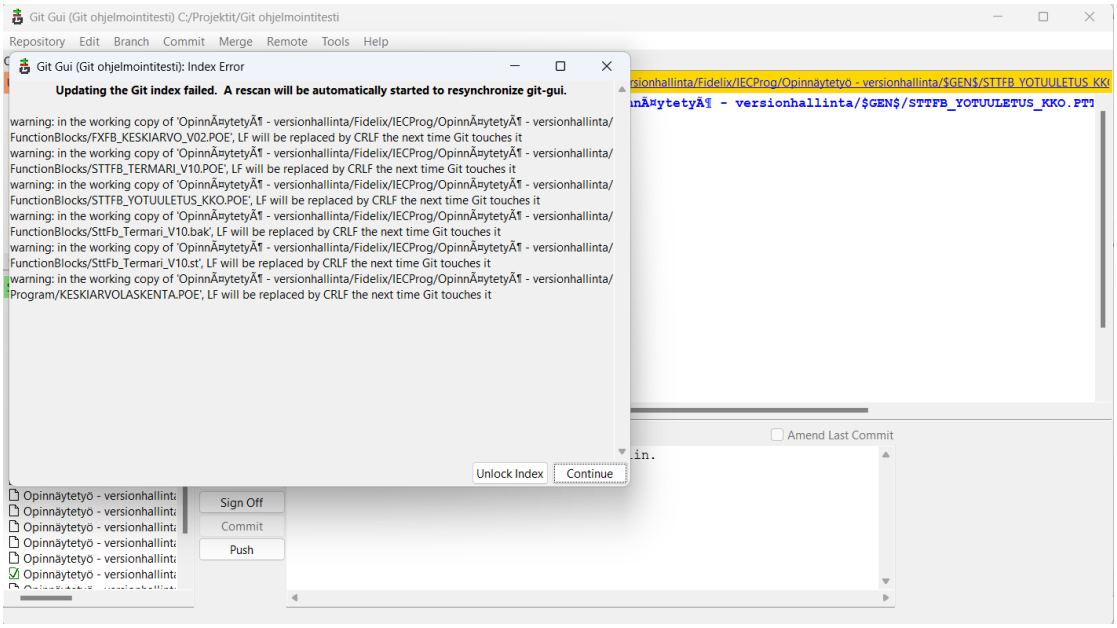
Kuvankaappauksen (kuva 35) yläosassa nähdään ST-koodiin luodut muuttujat ja niiden tyypit sekä koodissa käytössä olevat ”function blockit” eli valmiit ohjelmistokoodit, jota voidaan käyttää universaalisti. Tällaisia voivat esimerkiksi olla keskiarvolaskennat ja termostaattikäytöt. Alempana on itse suoritettava ohjelmistokoodi, jota ala-asema suorittaa sen käynnissä ollessa.



Kuva 35. OpenPCS. Valmis ohjelmakoodi

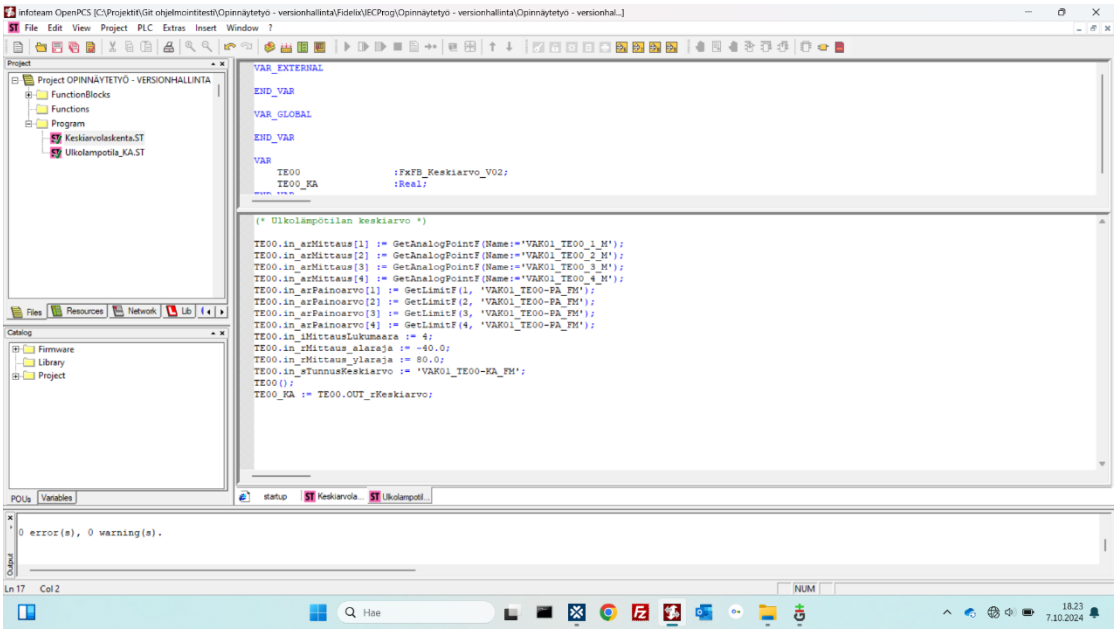
Ohjelmakoodin luonnin jälkeen skannataan Gitillä kansiorakenne uudestaan, ja lisätään ne nykyiseen haaraan.

Tässä vaiheessa huomataan, että mikäli ohjelmointiohjelmisto on auki Gitin haaraversiota tallentaessa, antaa Git index-virheen (kuva 36). Tämä saadaan kuitattua sulkemalla OpenPCS-ohjelmisto, jonka jälkeen painamalla Unlock Index -painiketta virhe kuittaantuu, jolloin Git tallentaa tallennettavan haaraversion ongelmitta.

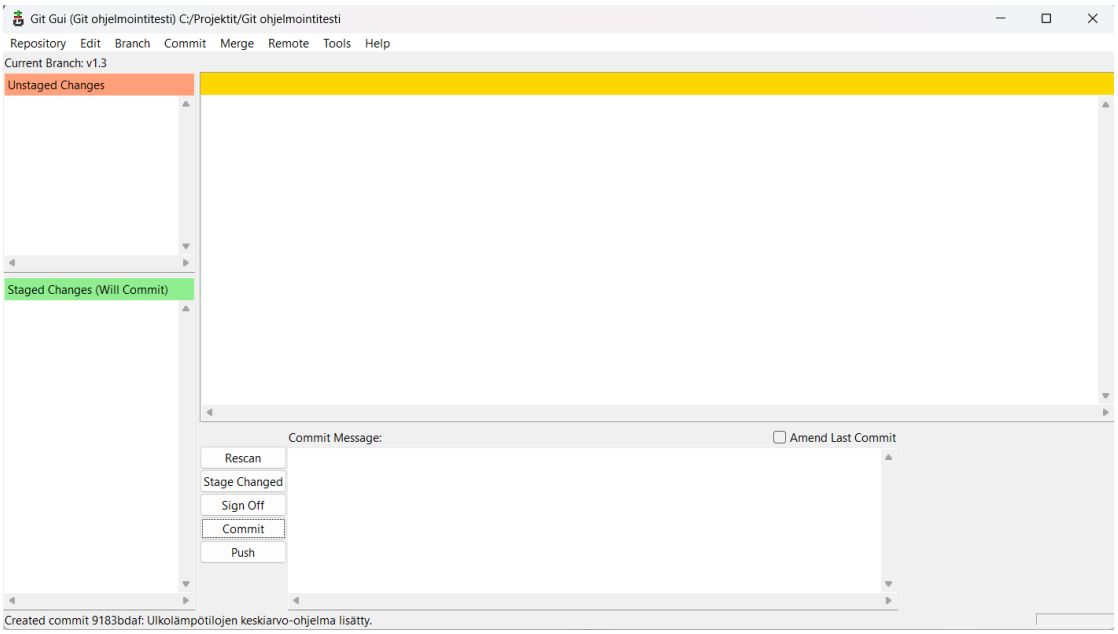


Kuva 36. Git GUI. Index-virhe

Lisätään vielä yksi ohjelmatiedosto, joka laskee kaikkien neljän ulkolämpötilamittauksen keskiarvon (kuva 37). Luodaan sille uusi haara v1.3 (kuva 38).



Kuva 37. OpenPCS. Lisätty uusi koodinpätkä

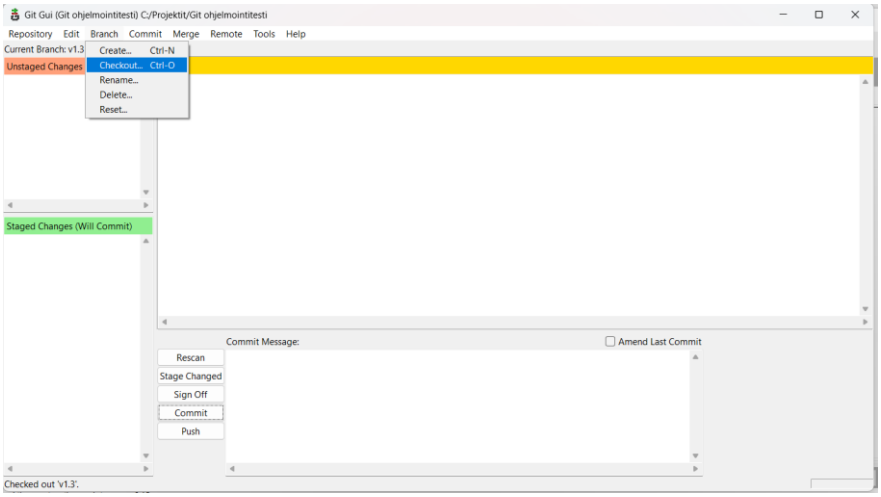


Kuva 38. Git GUI. Tehty uusi versiohaara v1.3, joka sisältää juuri tehdyn koodin

7.4 Versiohaaran vaihto

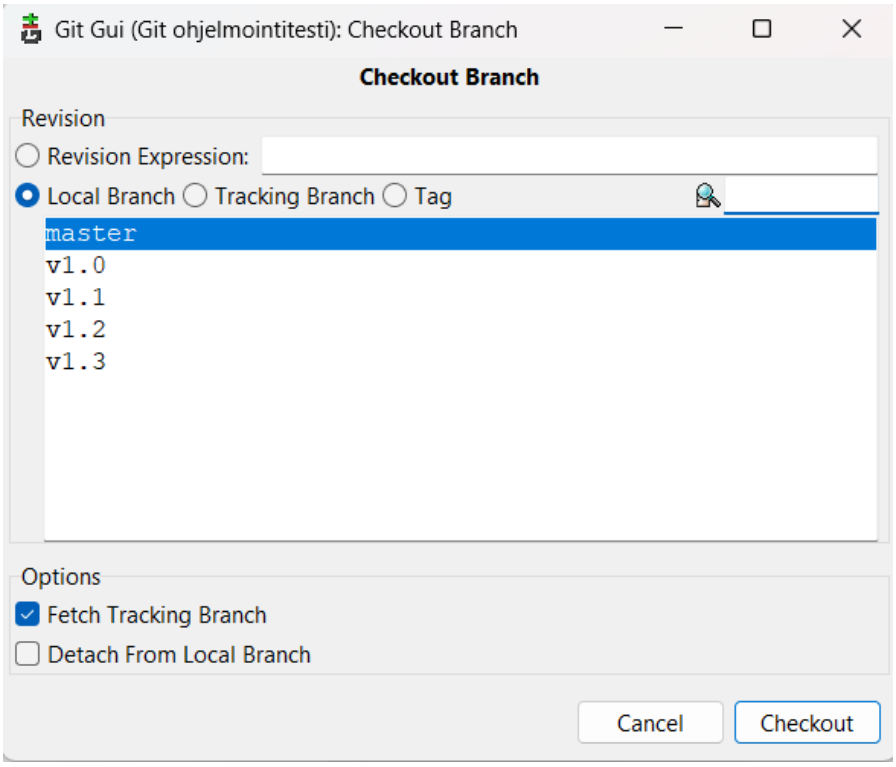
Nyt kun ohjelmakoodi on valmis, voidaan se ajaa ensimmäiseen master-haaraan, joka ohjelmaa käyttöönottaessa tehtiin. Koodista voitaisiin julkaista virallinen ensimmäinen versio ja poistaa edelliset työhaarat.

Vaihdetaan haara yläpalkista löytyvän Branch-valikon Checkout -painikkeen kautta (kuva 39).



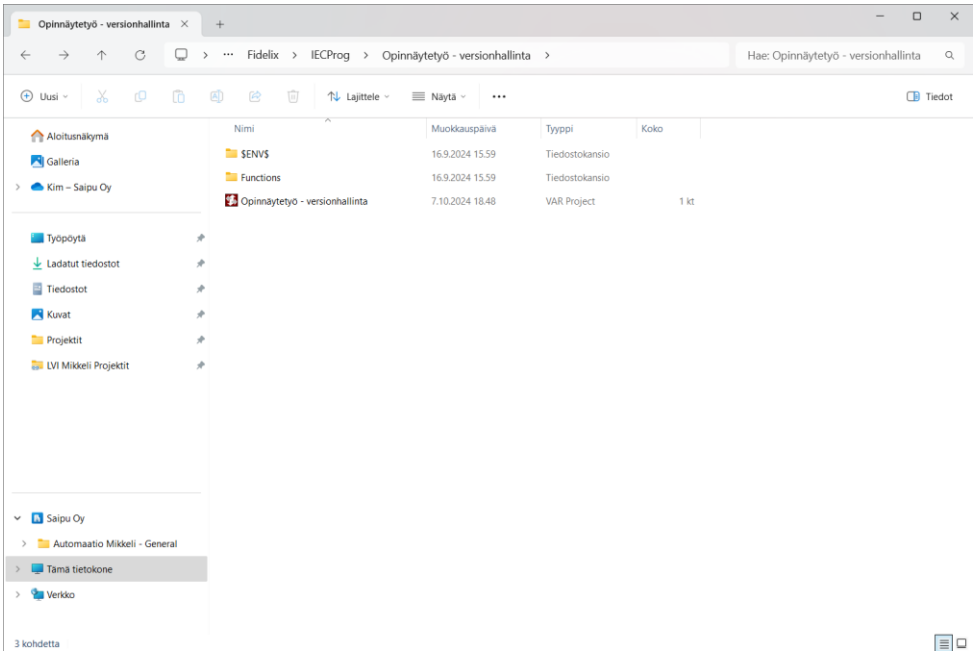
Kuva 39. Git GUI. Haaran vaihto

Valitaan master-haara ja painetaan Checkout -painiketta, jonka jälkeen Git aktivoi kyseisen haaran sisältämät tiedostot näkyviin ja hävittää aikaisemmat v1.3-haaran tiedostot näkyvistä kansiosta (kuva 40).



Kuva 40. Git GUI. Haaran vaihtovalikko

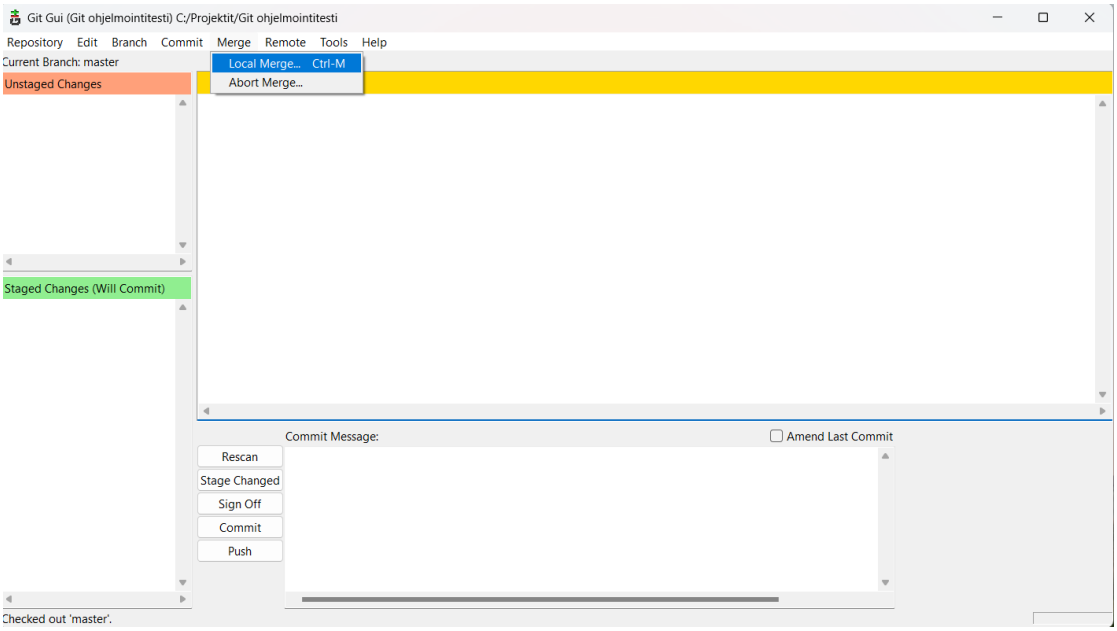
Seuraavaksi tarkastetaan, vaihtoiko Git haaran oikeaksi tarkastamalla tilanne resurssienhallinnan kautta (kuva 41). Ohjelmakansiosta nähdään, että se on käytännössä tyhjä, kuten sen pitääkin.



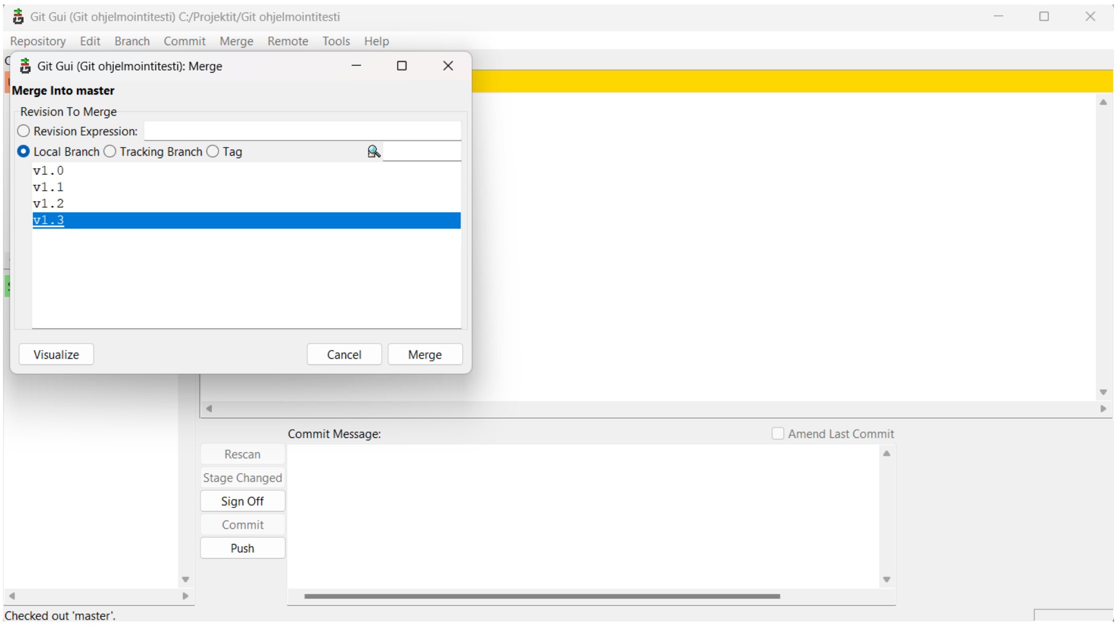
Kuva 41. Windowsin resurssienhallinta. Master-haara ennen versiohaarojen yhdistämistä

7.5 Versioiden yhdistäminen

Nyt, kun haaraksi on valittu master-haara, voidaan siihen liittää haaran v1.3 tiedostot. Tämä tehdään valitsemalla merge-valikon alta ”Local Merge” (kuva 42). Painetaan painiketta, joka avaa seuraavan valikon (kuva 43).

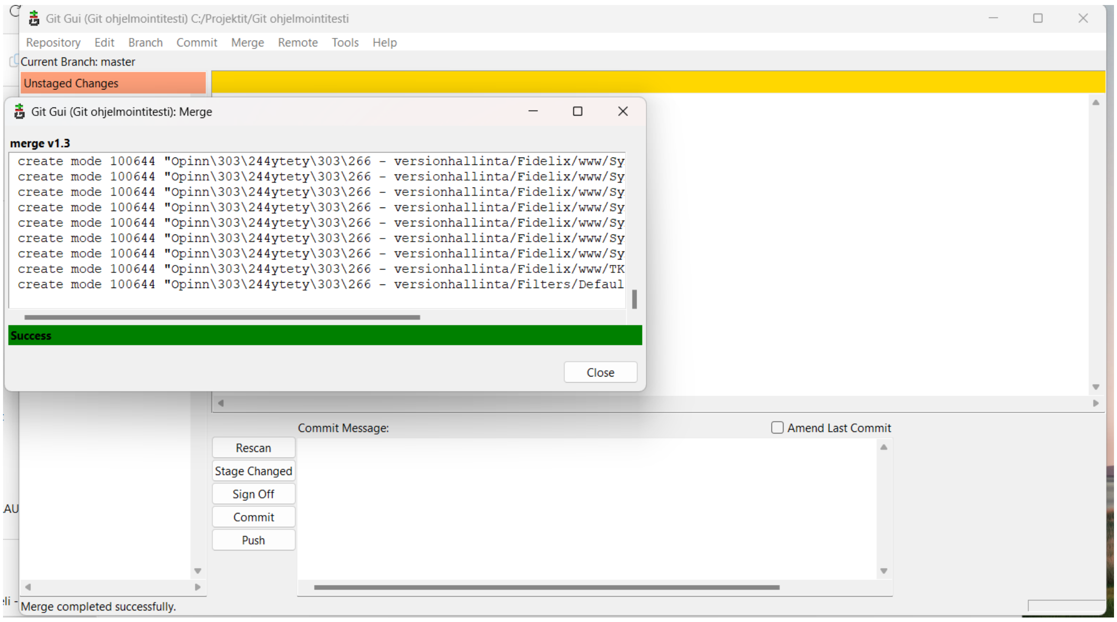


Kuva 42. Git GUI. Yhdistämisvalikko



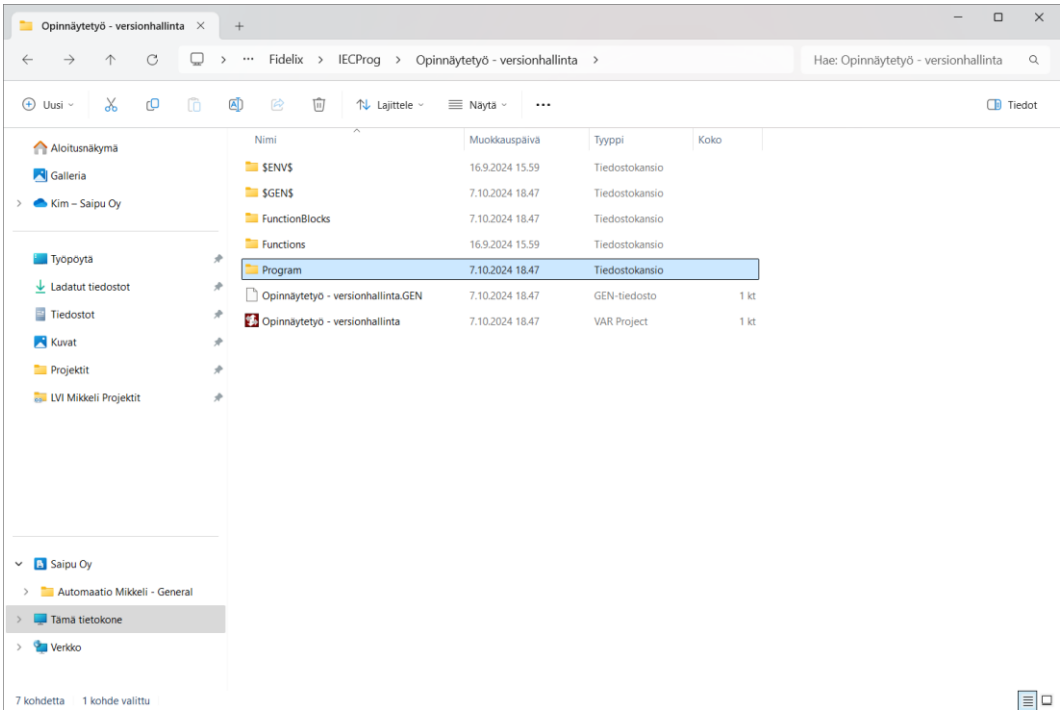
Kuva 43. Git GUI. Yhdistettävien haarojen valinta

Valitaan haaraversioksi v1.3 ja painetaan merge -painiketta, jolloin Git alkaa ajamaan haaran v1.3 tiedostoja master-haaraversioon (kuva 44).



Kuva 44. Git GUI. Haarojen yhdistäminen suoritettu

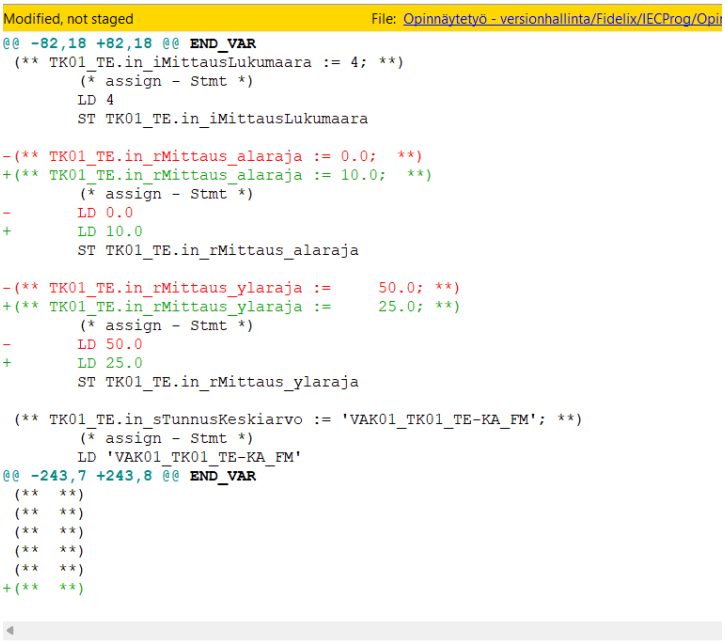
Kun merge-komennon suorittama tiedostosiirto on valmis, tarkastetaan kansiorakenteen sisältö (kuva 45). Nyt voidaan huomata, että aikaisemmin tehdyt muutokset v1.3-versiossa ovat näkyvissä nyt kansiossa, vaikka haarana on aikaisemmin luotu master-haara.



Kuva 45. Windowsin resurssienhallinta. Master-haara versiohaarojen yhdistämisen jälkeen

7.6 Koodin muutoksien näkyminen Gitissä

Kun projektin loppuvaihe on meneillään, monesti siihen tehdään pieniä viilauksia, joita voi olla hyvin vaikea huomata omilla silmillään. Tähänkin Git on suuresti apuna, koska se näyttää teksti-ikkunassa suoraan, mihin koodissa on koskettu. Kuvassa 46 nähdään punaisella olevan tekstin sisältämä alkuperäinen koodi ja vihreä on muutettu koodi.



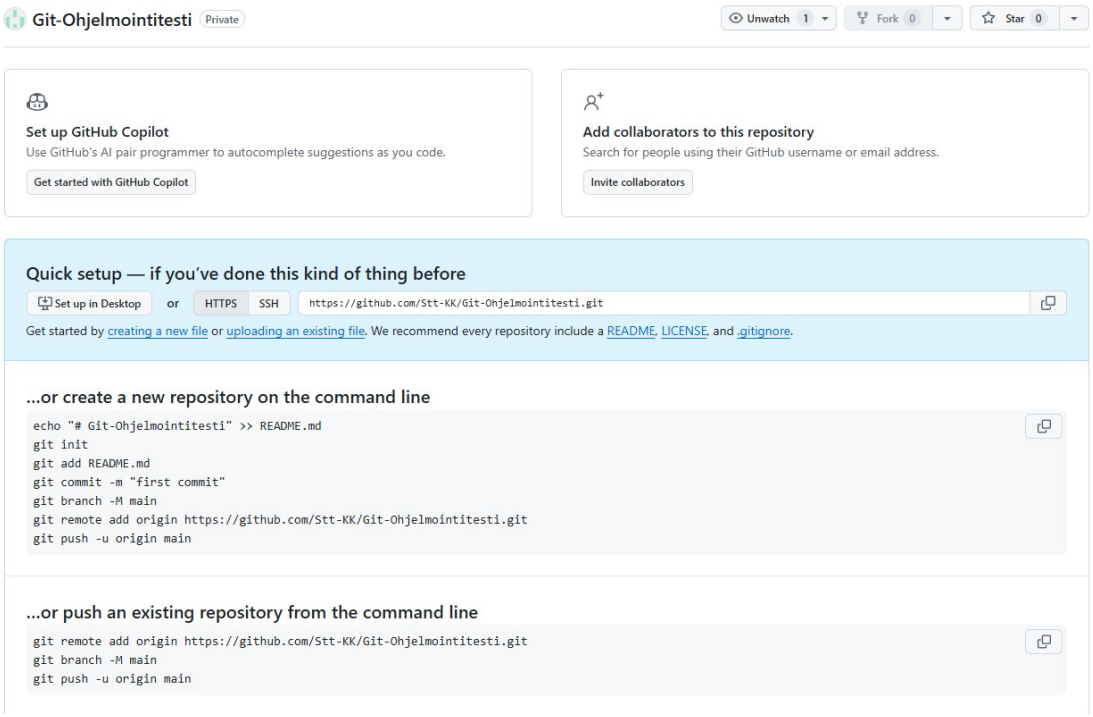
Kuva 46. Git GUI. Koodin muutoksien näkyminen Gitissä

7.7 Versionhallintaohjelmiston testaus pilvipalvelussa

Lopuksi jäljelle jää versionhallinnan testaaminen pilvipalvelussa. Koska tiedostot ajetaan GitHubiin, on sen käyttöönotto versionhallinnassa hyvin yksinkertaista. Riittää, että pääversiohaara kopioidaan GitHubiin käyttäen apuna Gitiä, ja määritetään Git seuraamaan tämän tietovaraston tiedostoja.

7.7.1 Käyttäjätilin ja pilvitietovaraston luominen

GitHub vaatii käyttäjätilin, jonka luominen on ilmaista. Kun käyttäjätili on luotu, luodaan tietovarasto GitHubiin. Varaston luomisen jälkeen se on määritettävä Gitille. Kuvasta 47 voidaan nähdä, että mikäli varasto on jo valmiiksi olemassa, riittää, että syötetään kolme komentoa Gitille, minkä jälkeen pilvivarasto on määritetty Gitille.



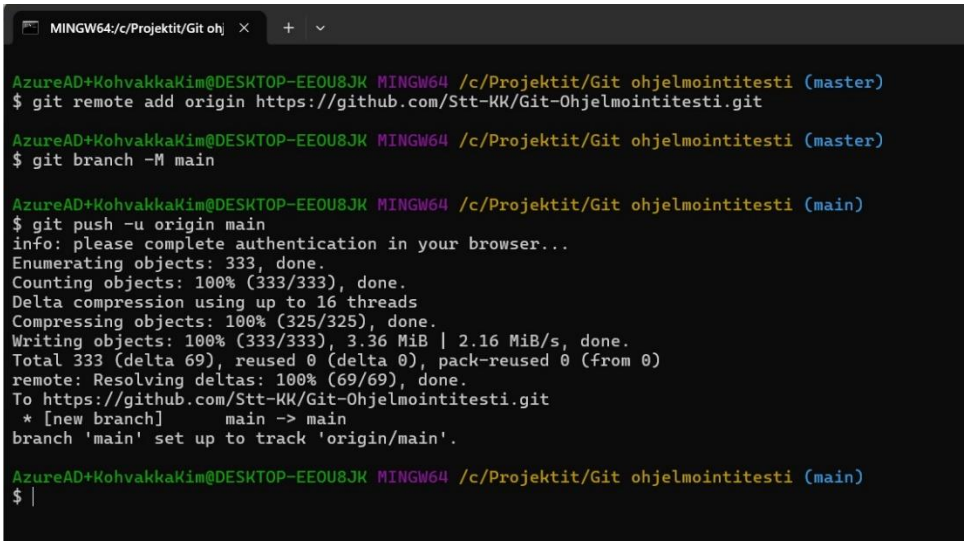
Kuva 47. GitHub. Luotu tietovarasto

7.7.2 Pilvitietovaraston määritys ja paikallisien tiedostojen lataus pilvitietovarastoon

Määritellään varasto Gitille käyttämällä Git Bashia (kuva 48) tai vaihtoehtoisesti käyttämällä Git GUIta (kuva 58). Tässä vaiheessa on syytä tarkistaa, että nykyinen haara on haluttu, ja mikäli näin ei ole, vaihdetaan haara ennen tiedostojen ajamista pilveen käyttäen apuna komentoa ”git branch --list”, jonka jälkeen valitaan listasta haluttu haara ja syötetään komento ”git switch *haluttu haaran nimi ”*.

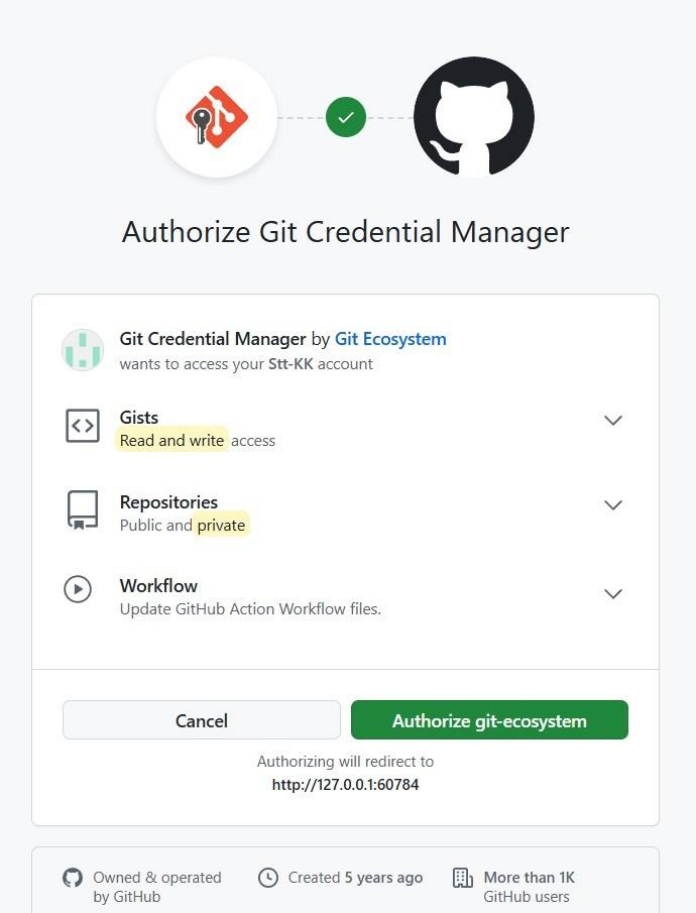
Syötetään luodun pilvivaraston osoite ensimmäisenä komennolla ”git remote add origin *pilvivaraston osoite*”, jonka jälkeen vaihdetaan nykyisen Master-haaran nimeksi ”Main” käyttäen komentoa ”git branch -M main”.

Viimeiseksi lähetetään juuri uudelleen nimetyn paikallisen haaran sisältämät tiedostot pilvivarastoon komennolla ”git push -u origin main”.



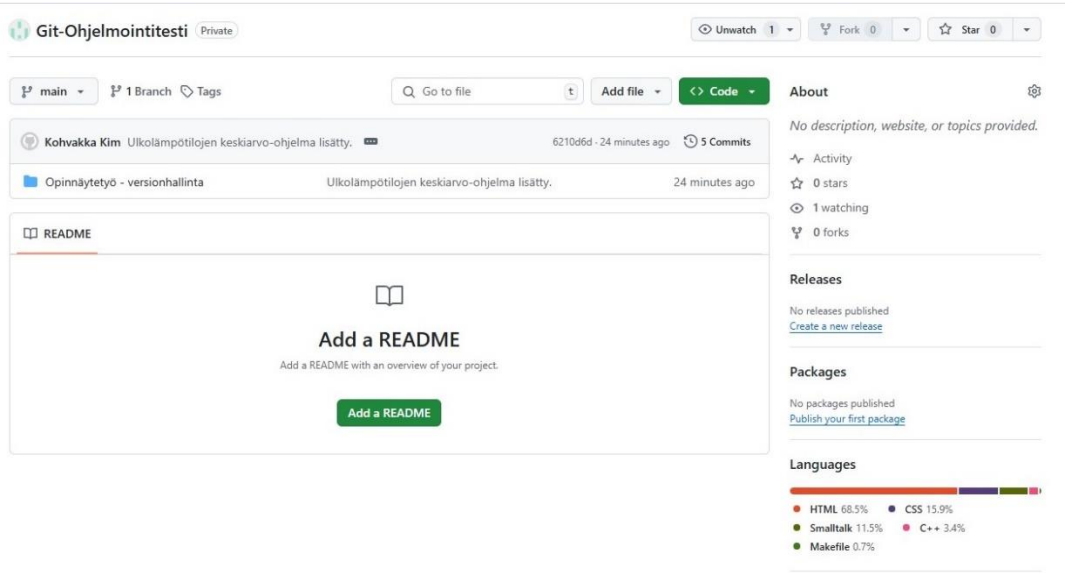
Kuva 48. Git Bash. Pilvivaraston määrittely paikalliseen kansioon

Tässä vaiheessa GitHub kysyy käyttäjätunnuksia (kuva 49), mikäli Gitille ei ole aikaisemmin annettu käyttäjätunnuksia ja käyttöoikeuksia, jolloin Git avaa selaimeen kirjautumisikkunan. Kun käyttöoikeudet on annettu, lataa Git paikalliset muutokset pilvivarastoon.



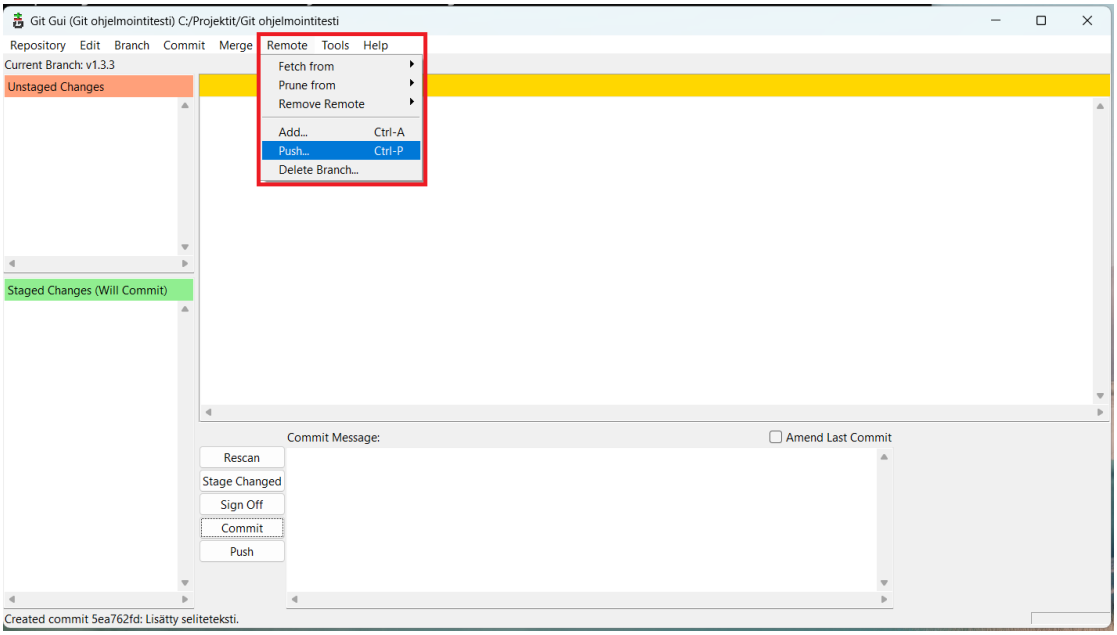
Kuva 49. GitHub. Käyttäjäoikeuksien antaminen

Kuvasta 50 nähdään, että aikaisemmin tyhjä pilvivarasto sisältää nyt aikaisemmin paikallisesti luodut projektitiedostot.

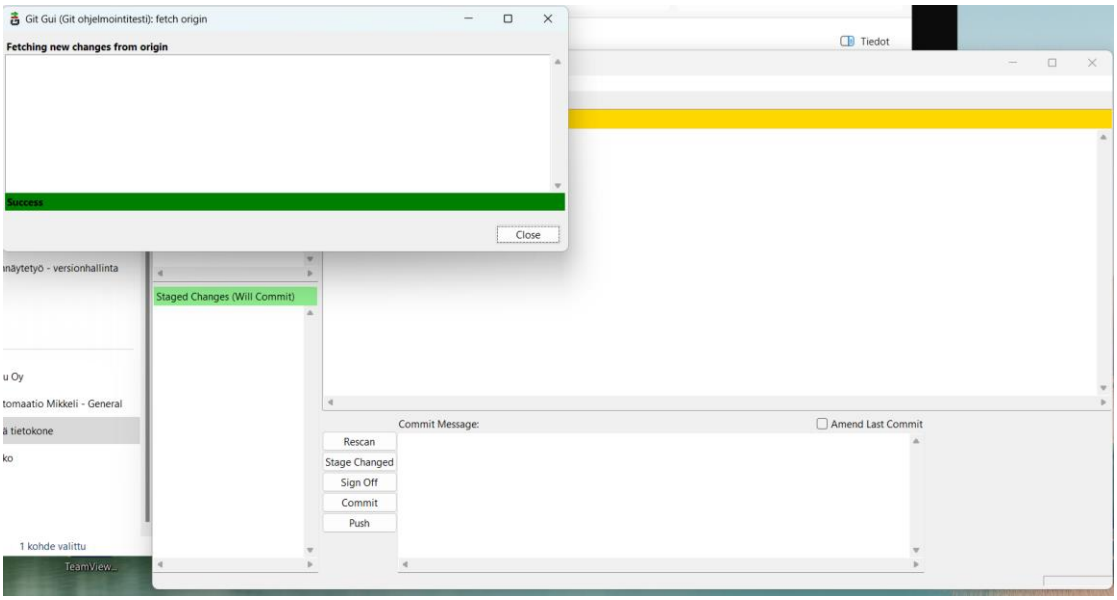


Kuva 50. GitHub. Kansiohakemistoon ladatut paikalliset tiedostot näkyvissä

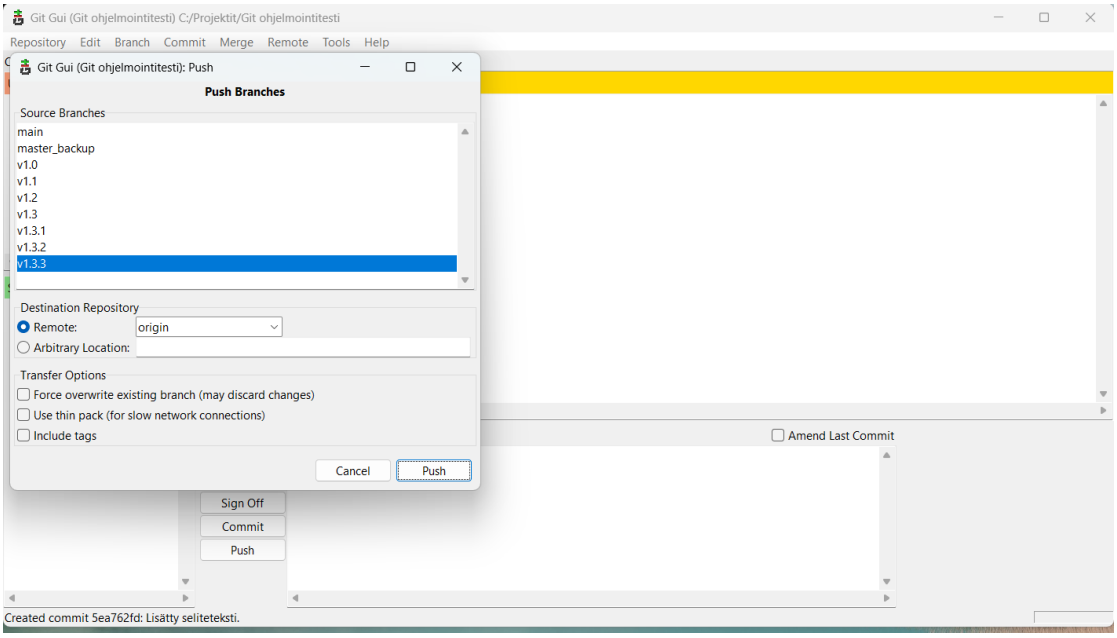
Vastaisuudessa projektin viimeisin versio voidaan hakea pilvivarastosta Remote-valikon Fetch-komennon kautta (kuva 51 ja kuva 52), joka vertaa nykyisen paikallisen tietovaraston tiedostoja pilvestä löytyviin varastoihin. Lisäksi saman valikon alta löytyvän Push-komennon avulla voidaan pilvivarastoon ladata uusi haara tai päivittää vanhaa haaraa (kuva 53).



Kuva 51. Git GUI. Remote-valikko

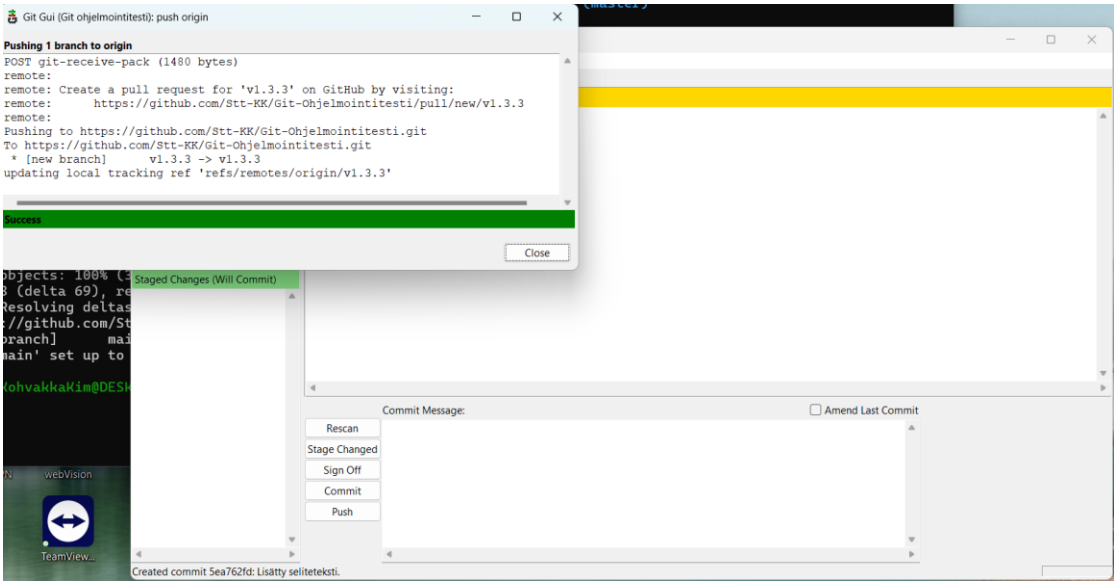


Kuva 52. Git GUI. Fetch-komennon suoritus



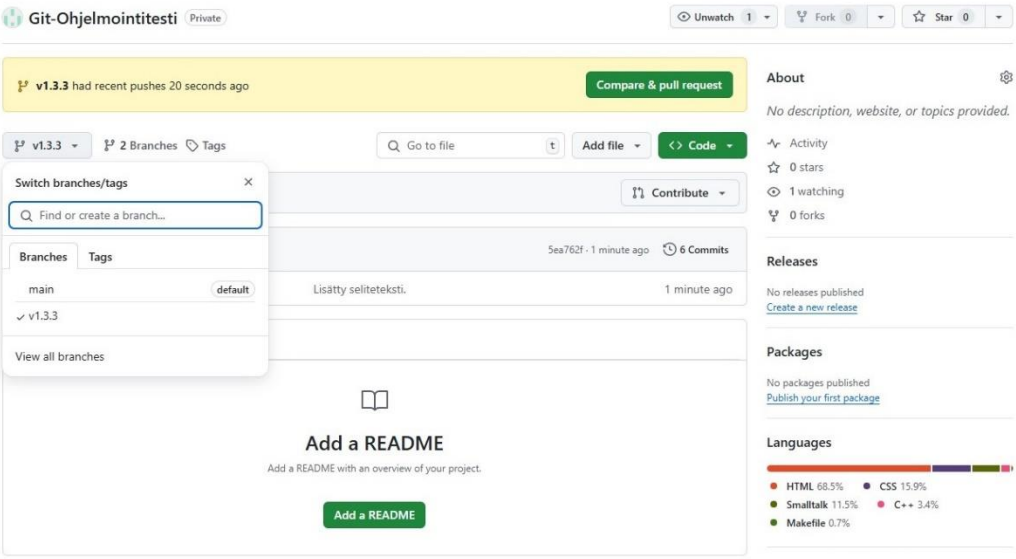
Kuva 53. Git GUI. Pilvihakemistoon lähetettävän haaran valinta

Testimielessä on luotu uusi haara v1.3.3, jonka koodiin on lisätty tekstiä. Lähetetään v1.3.3-haara pilveen (kuva 54), ja katsotaan, miten haaran muutokset näkyvät GitHubissa.



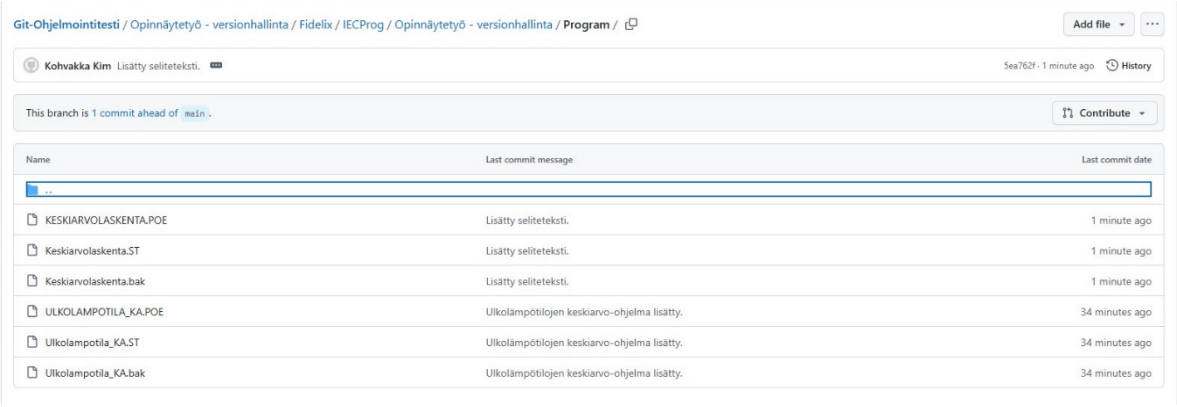
Kuva 54. Git GUI. Haaran lähetys pilveen

GitHubissa haaroja voidaan vaihtaa kuvassa 55 näkyvän valikon alta. Valitaan juuri ladattu v1.3.3-haara aktiiviseksi, jonka jälkeen tarkastellaan sen sisältämiä tiedostoja.



Kuva 55. GitHub. Haarojen vaihtovalikko

Kuvasta 56 voidaan nähdä, että vain muutetut tiedostot ovat ajettu haaraan ja tiedostot, johon ei ole kajottu ovat pysyneet ennallaan. Lisäksi oikeassa reunassa nähdään Gitissä aikaisemmin kirjoitetut selitetekstit.

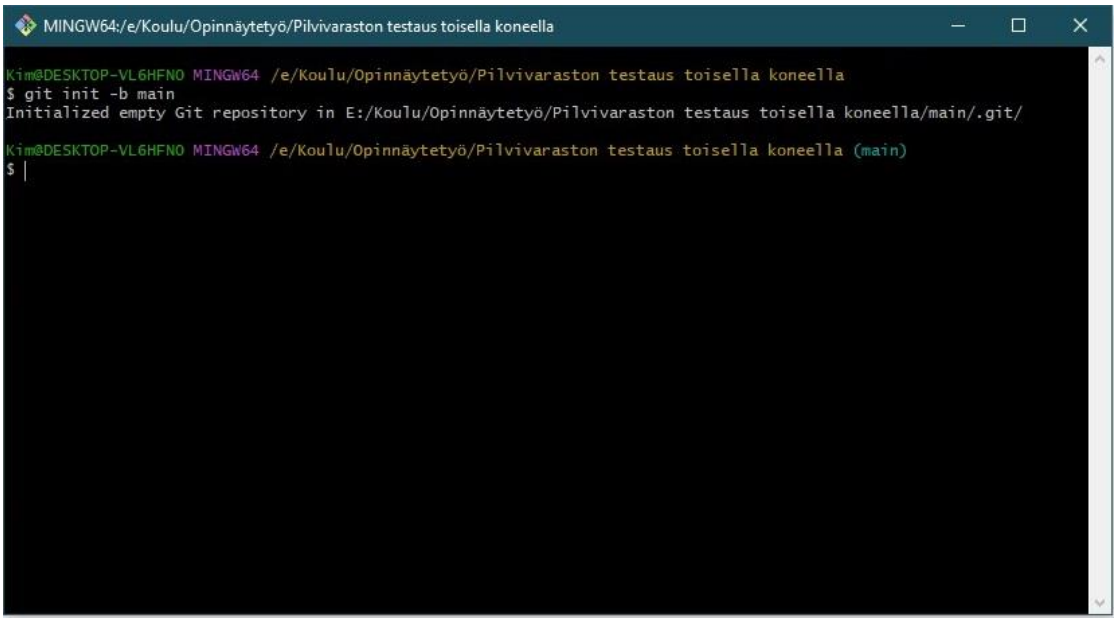


Kuva 56. GitHub. V1.3.3-haaran tietovarasto, jossa nähdään muutetut tiedostot

7.7.3 Toisen käyttäjän muutoksien lataaminen ja tiedostojen yhdistäminen

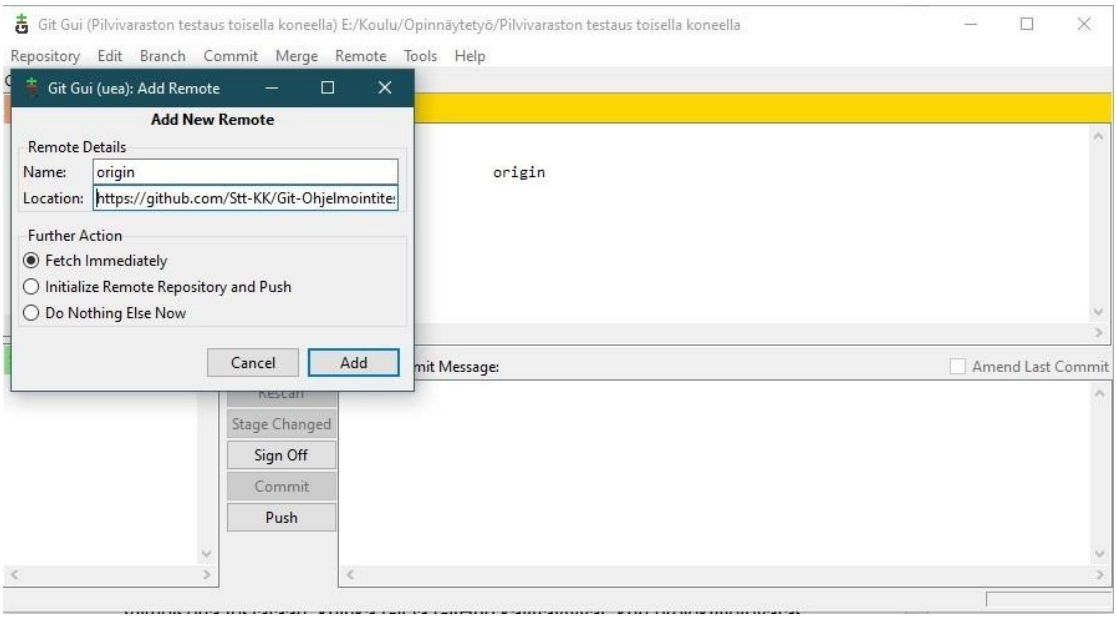
Viimeisenä testataan, kuinka Git ja GitHub käyttäytyvät, kun projektitietovarastoa käsittelee kaksi eri tietokonetta yhtäaikaisesti.

Otetaan avuksi toinen tietokone, johon on asennettu Git. Määritetään Git Bashin kautta uusi tietovarasto, johon pilvestä ladataan tiedostot. Annetaan komento ”git init -b main”, joka luo tietovaraston nimeltä ”Main” (kuva 57).



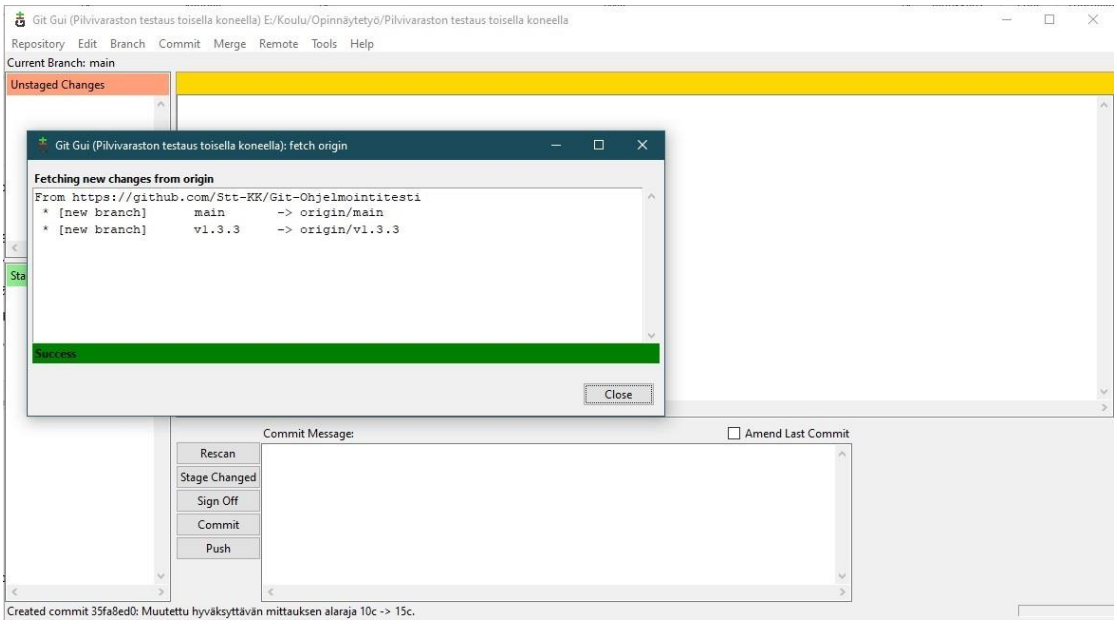
Kuva 57. Git Bash. Toisella tietokoneella tietovaraston luonti

Nyt, kun main-tietovarasto on luotu uuteen kansioon, voidaan avata Git GUI. Lisätään ulkoisen tietovaraston hakemisto Remote-valikosta löytyvän Add-toiminnon kautta. Annetaan pilvestä löytyvälle tietovarastolle tunnistettava nimi paikallisesta tietovarastosta erottamiseksi ja pilvitietovaraston www-osoite (kuva 58) ja painetaan Add -painiketta. Nyt Git tekee jokaisen pilvitietovaraston haaran nimeen tunnisteen ”origin/haaran nimi”. Vaihtoehtoisesti pilvitietovarasto voidaan lisätä Git Bashin kautta, kuten kuvassa 48 tehdään.



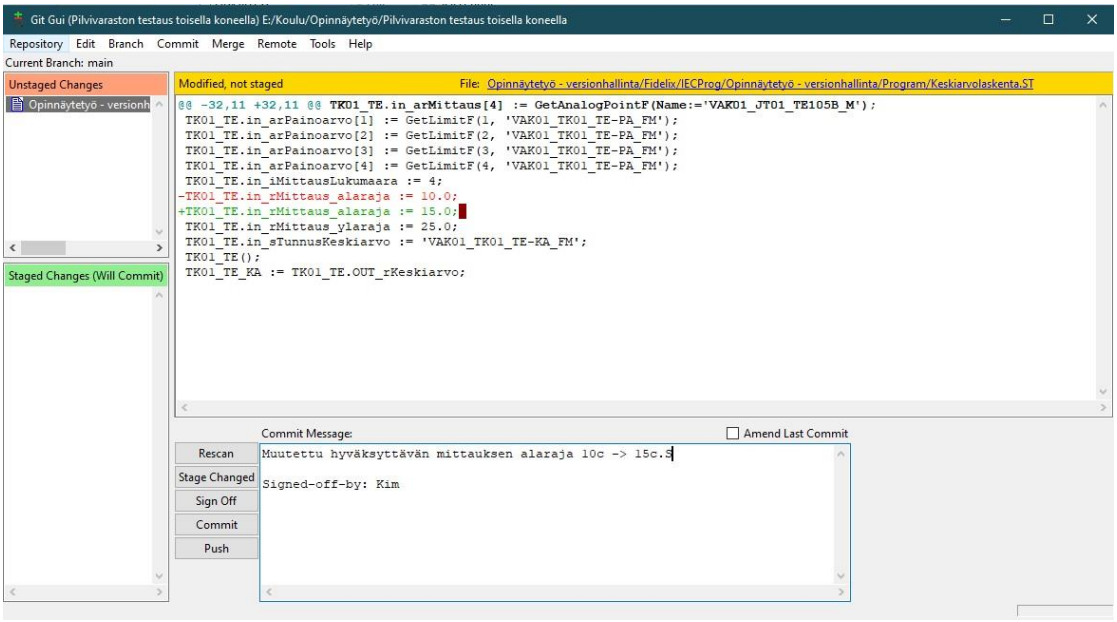
Kuva 58. Git GUI. Pilvitietovaraston lisäys ohjelmistoon

Seuraavaksi haetaan Remote-valikon kautta löytyvän fetch-komennon kautta pilvestä löytyvät tietovarastot (kuva 59). Nyt paikalliset tietovarastot ovat viimeisimmät ja ohjelmamuutoksia voidaan tehdä.

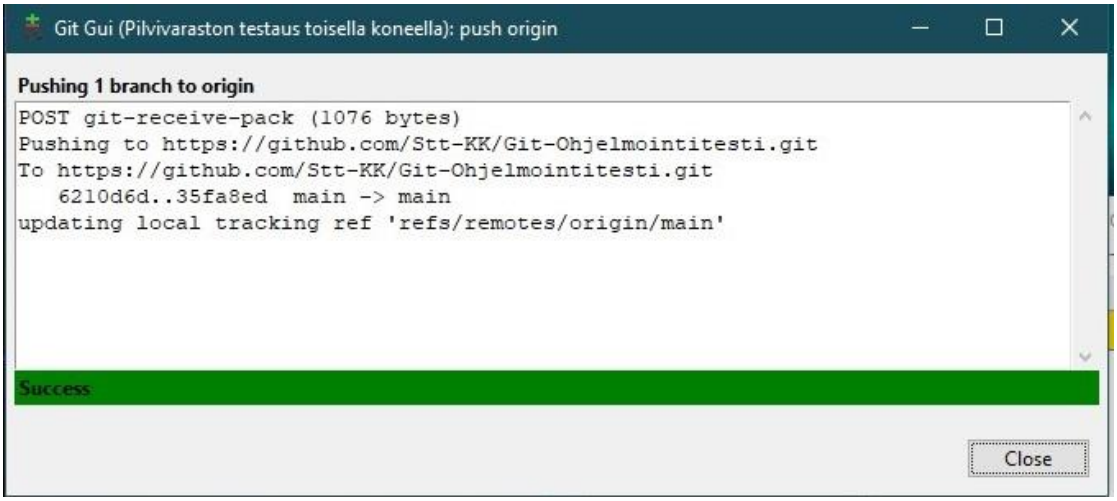


Kuva 59. Git GUI. Toisella tietokoneella pilvitietovarastojen haku

Muutetaan ohjelmakoodin laskennan alarajaa hieman isommaksi ja tallennetaan muutokset juuri luotuun paikalliseen main-haaraan, jonka jälkeen ajetaan haaran muutokset pilvessä sijaitsevaan main-haaraan (kuva 60 ja 61).

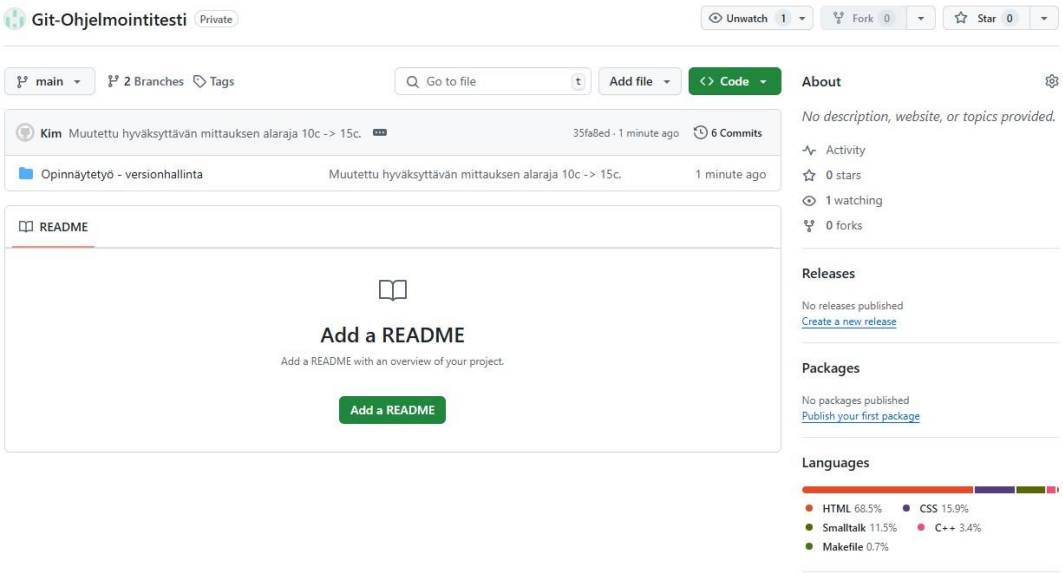


Kuva 60. Git GUI. Toisella tietokoneella muutoksien tallentaminen paikalliseen tietovarastoon



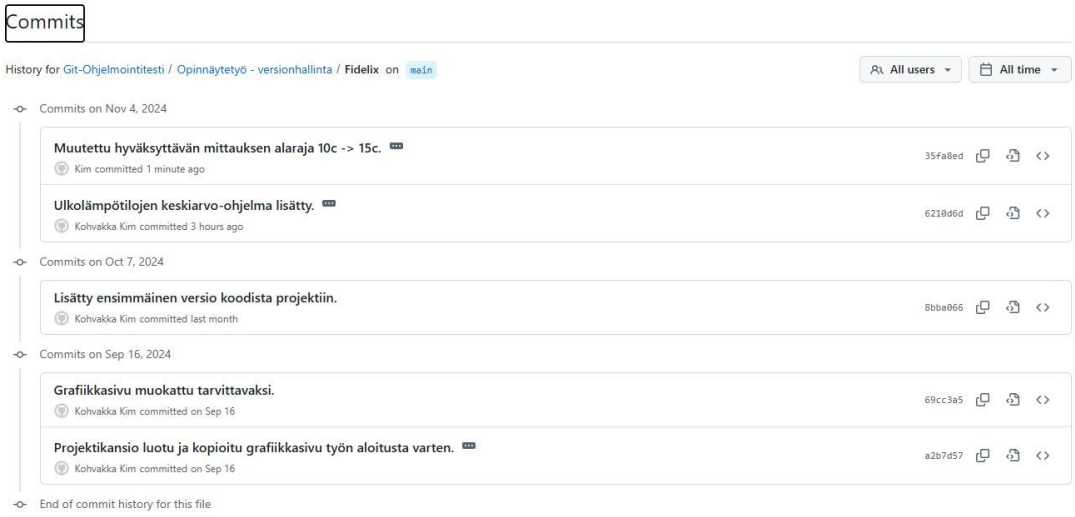
Kuva 61. Git GUI. Toisella tietokoneella muutoksien tallentaminen pilvitietovarastoon

Tarkastetaan, että juuri ladatut muutokset ladattiin onnistuneesti GitHubiin (kuva 62).



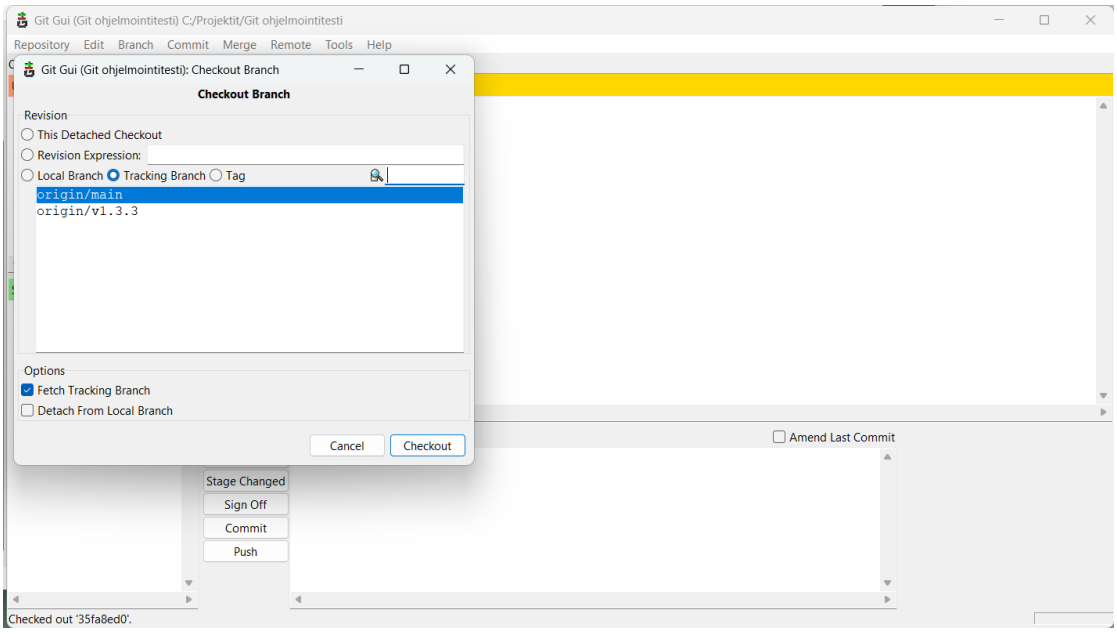
Kuva 62. GitHub. Toisella tietokoneella tehdyt muutokset näkyvät nyt main-haarassa

Tallennettujen haarojen loki voidaan myös tarkastaa GitHubista, josta voidaan nähdä juuri tehtyjen muutoksien selitetekstit ja kuka muutokset ovat tehnyt (kuva 63).



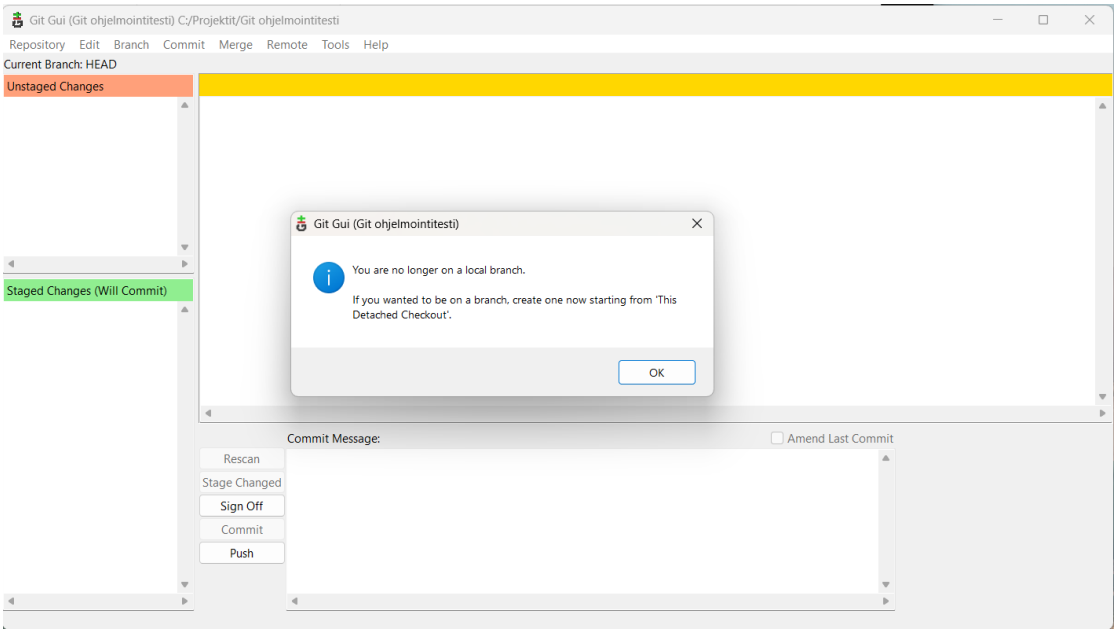
Kuva 63. GitHub. Haarojen loki

Seuraavaksi voidaan vaihtaa takaisin alkuperäiselle tietokoneelle, jolla projekti alun perin luotiin. Ladataan tietovaraston muutokset pilvestä Branch-valikosta löytyvän Checkoutin kautta. Valitaan Local Branchin sijasta Tracking Branch, joka hakee pilvitietovarastot (kuva 64). Valitaan haarapohjaksi origin/main, joka sisältää juuri toisella tietokoneella tehdyt ohjelmamuutokset.



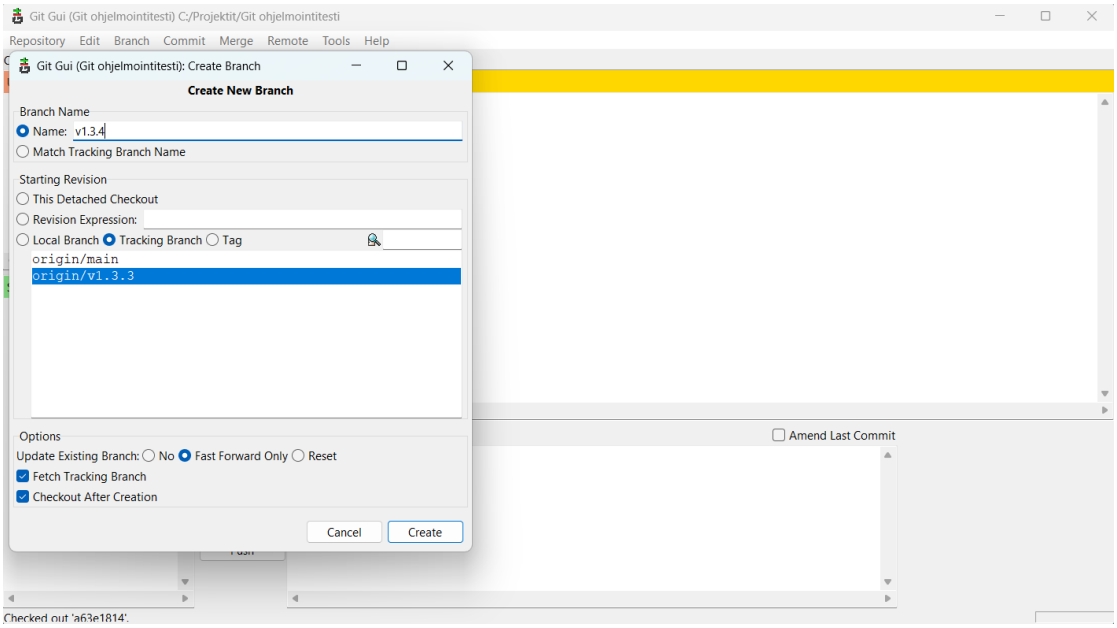
Kuva 64. Git GUI. Pilvitietovaraston lataus

Tämän jälkeen Git varoittaa, että käytössä oleva tietovarasto ei ole paikallinen ja kehottaa luomaan uuden (kuva 65).



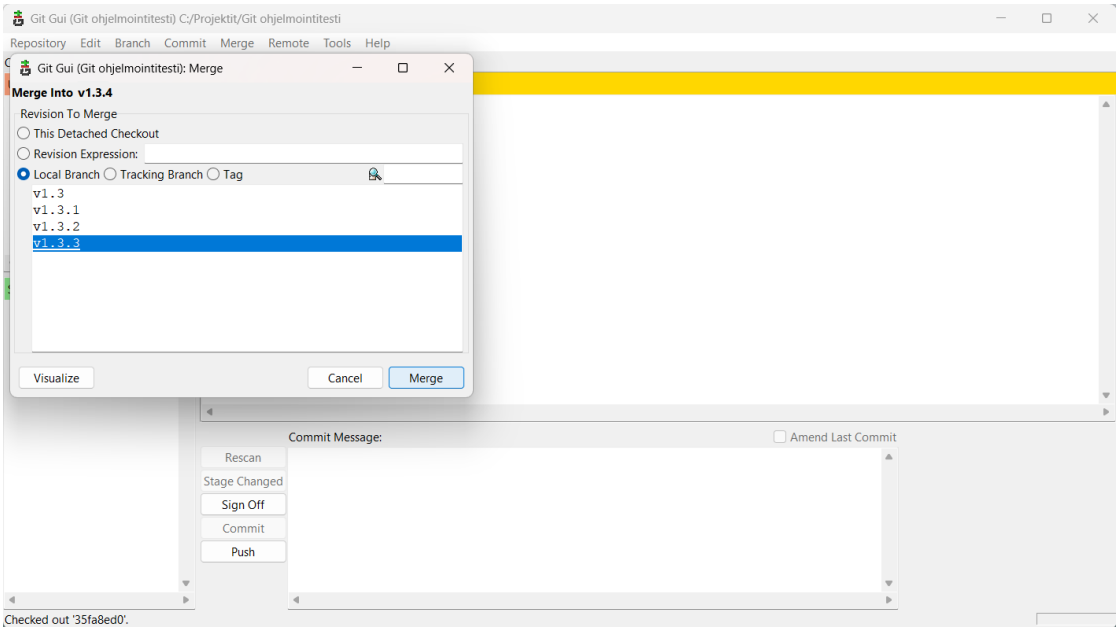
Kuva 65. Git GUI. Varoitus pilvivarastosta paikallisen tietovaraston sijasta

Luodaan uusi paikallinen tietovarasto v1.3.4, ja valitaan pohjaksi pilvestä löytyvä main-tietovarasto, jotta saadaan juuri tehdyt muutokset tähän tietovarastoon (kuva 66).



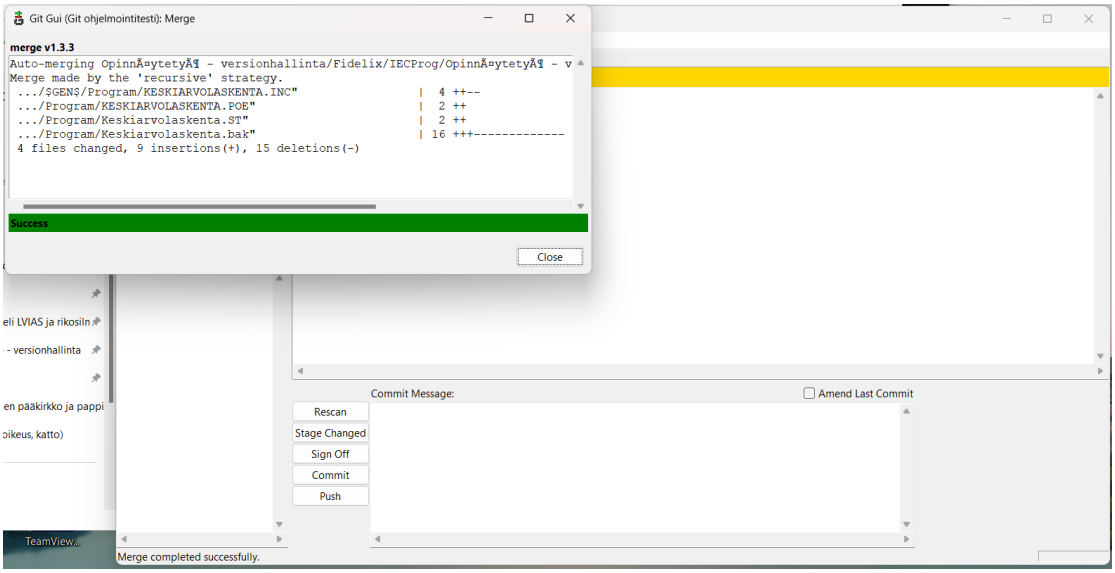
Kuva 66. Git GUI. Uuden tietovaraston luonti pohjana pilvitietovarasto

Seuraavaksi yhdistetään juuri luodun v1.3.4-tietovaraston tiedostot paikallisen v.1.3.3-tietovaraston tiedostoihin (kuva 67).



Kuva 67. Git GUI. Tietovarastojen yhdistäminen

Merge -painiketta painamalla Git ilmoittaa, mitä muutoksia tiedostoihin on tullut (kuva 68).



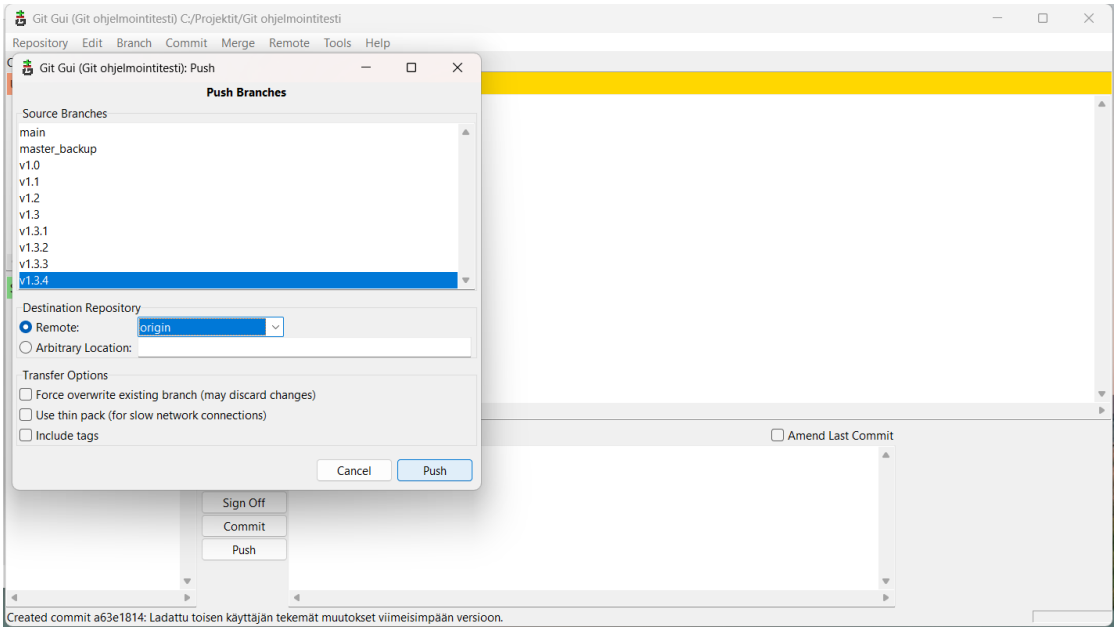
Kuva 68. Git GUI. Yhdistettyjen tietovarastojen muutokset tiedostoihin nähden

Tarkastetaan ohjelmakoodista (kuva 69), että tallentuivatko aikaisemmin pilveen tallennetun main-tietovaraston ja tietokoneella paikallisesti olleen v1.3.3-tietovaraston muutokset ohjelmakoodiin. Alarajana on 15 astetta, kuten aikaisemmin vaihdettiin ja ohjelmakoodin lopussa oleva testiselitetekstikin on näkyvissä. Tietovarastojen yhdistäminen on siis onnistunut.

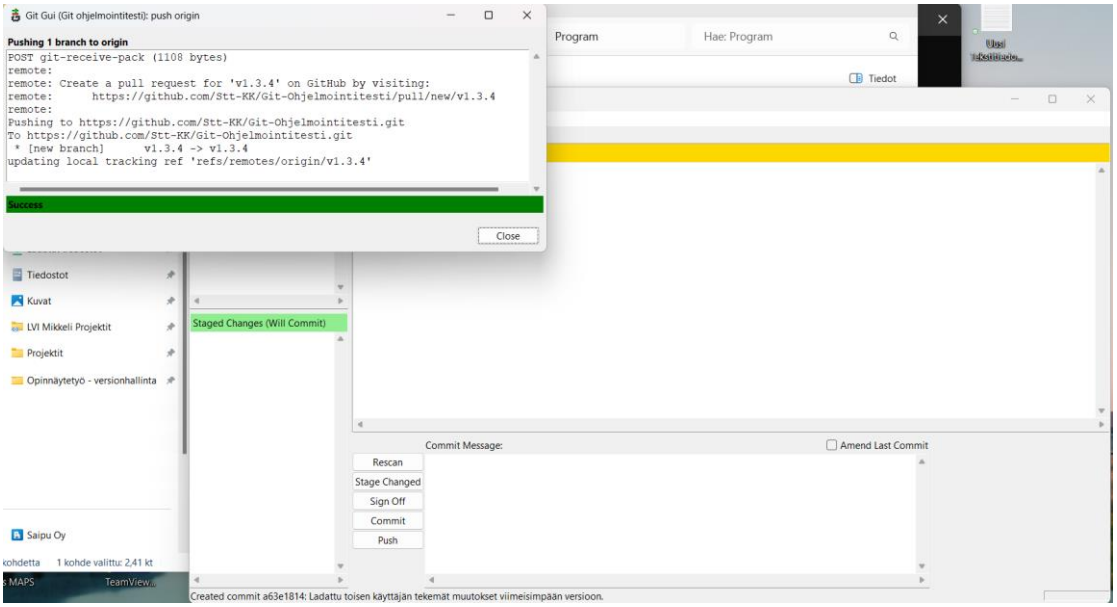
```
TK01_TE.in_rMittaus_alaraja := 15.0;
TK01_TE.in_rMittaus_ylaraja :=  25.0;
TK01_TE.in_sTunnusKeskiarvo := 'VAK01_TK01_TE-KA_FM';
TK01_TE();
TK01_TE_KA := TK01_TE.OUT_rKeskiarvo;

(* Luetaan aikaisemmin generoitu palvelualueen keskiarvo ja ohjataan tuuletusta keskiarvon ja ulkolämpötilan raja-arvojen mukaan, mikäli aikaohjel

TK01_YT.in_rUlkoLampotila               := GetAnalogPointF(Name:='VAK01_TE0_M');
TK01_YT.in_rUlkoLampotila_Paalle_Raja   := GetLimitF(1, 'VAK01_TK01_YT_FM');
TK01_YT.in_rUlkoLampotila_Pois_Raja     := GetLimitF(2, 'VAK01_TK01_YT_FM');
TK01_YT.in_rSisaLampotila               := GetAnalogPointF(Name:='VAK01_TK01_TE-KA_FM');
TK01_YT.in_rSisaLampotila_Paalle_Raja   := GetLimitF(3, 'VAK01_TK01_YT_FM');
TK01_YT.in_rSisaLampotila_Pois_Raja     := GetLimitF(4, 'VAK01_TK01_YT_FM');
TK01_YT.in_iKayntilupa                  := GetDigitalPointF('VAK01_TK01_YT_T');
TK01_YT.in_iKaynninesto                 := 0;
TK01_YT.in_iAlkuKuukausi                := GetDigitalPointF('VAK01_TK01_YT_KK1_FI');
TK01_YT.in_iLoppuKuukausi               := GetDigitalPointF('VAK01_TK01_YT_KK2_FI');
TK01_YT();
TK01_YT_ohjaus := TK01_YT.out_iYotuuletus;

TK01_YT_indikointi := SetDigitalPointF(Value:= TK01_YT.out_iYotuuletus, LockState:=1, Name:='VAK01_TK01_YT_FI');

if TK01_YT_ohjaus = 1 then;
    TK01_ohjaus := SetDigitalPointF( Value:=4,LockState:=1, Name:='VAK01_TK01_TILA_FI' );
end_if;

(* Lisätty testiksi lisää tekstiä *)
```

Kuva 69. OpenPCS. Tehtyjen muutoksien tarkistaminen

Tämän jälkeen ajetaan paikallinen v1.3.4-tietovarasto GitHubiin. Valitaan haluttu tietovarasto ja painetaan Push -painiketta, jolloin tietovarasto siirtyy GitHubiin (kuva 70 ja 71).

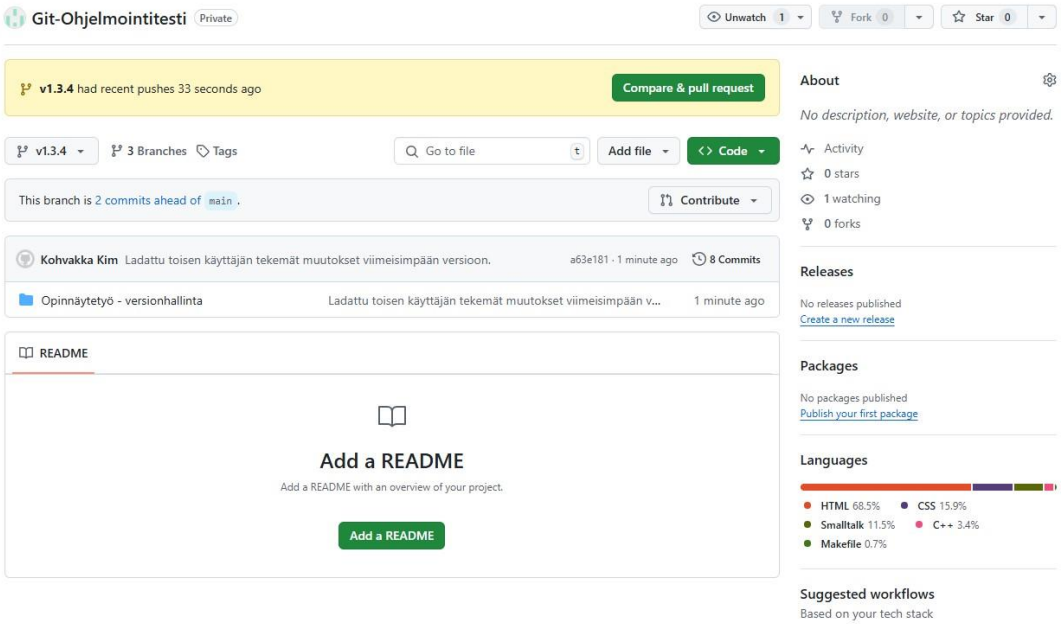


Kuva 70. Git GUI. Valintaikkuna paikallisen tietovaraston lataamiseksi GitHubiin



Kuva 71. Git GUI. V1.3.4-tietovaraston ajaminen GitHubiin

Viimeisenä tarkistetaan, että muutokset ovat näkyvissä GitHubissa. Kuvasta 72 nähdään, että muutokset ovat siirtyneet GitHubiin. Versionhallinta on siis onnistunut.



Kuva 72. GitHub. V1.3.4-tietovaraston tarkastaminen

9 POHDINTA

Kokonaisuudessaan voidaan todeta, että versionhallintaohjelmiston käyttöönotto ei ole kaikista yksinkertaisimmasta päästä varsinkin, kun ohjelmiston käyttöön ei ole yhtä suoraa ohjetta, kuinka sitä käytetään. Eli käyttöönotto vaati omaa selvittelyä erilaisista lähteistä ja kokeilua, kuinka ohjelmisto käyttäytyy. Ohjelmiston käyttöönotto ei kuitenkaan loppujen lopuksi ollut kovin monimutkaista, mutta välillä oli turvauduttava internetin ohjeistukseen, jotta ymmärsi, millä komennoilla asioita suoritettiin tai miten asioita piti parametroida.

Työtä tehdessä käytin versionhallintaohjelmistoa jo töiden ohella, ja sen edut tulivat nopeasti huomattua. Ei tarvitse enää edes harkita sitä edellistä toimintatapaa: kopioi lähdekansio ja liitä se kansiohakemistoon ”alkuperäinen”-nimellä ja aloita toisen kansion työstäminen ja myöhemmin ajan kuluessa alkaessa miettimään, mitähän muutoksia tuohon ohjelmistoon tulikaan tehtyä.

Muutamalla napin painalluksella Git vaihtaa vanhan version ja uuden version välillä, mikäli tarvetta tulee tällaiselle. Samoten versiot pysyvät hallitusti yhdessä osoitteessa, eikä useita hakemistoja tarvita. Etenkin koodista muutettujen rivien tai parametrien näkyminen versionhallintaohjelmiston ikkunassa helpottaa huomattavasti kokonaiskuvan muodostamista, mitä kyseisellä versiolla onkaan muutettu alkuperäiseen lähteeseen verrattuna.

Lisäksi versionhallintaohjelmiston käyttöönottamisella vältytään turhalta viimeisimmän lähdekoodin kyselemiseltä ohjelmoijien kesken, kun viimeisin versio lähdekoodista on aina saatavilla jokaiselle ohjelmoijalle. Suurinta etua työstä en ole vielä päässyt hyödyntämään, eli useamman ohjelmoijan yhtäaikaista työskentelyä. Tällaisessa käytössä ohjelmisto olisi erittäin hyvä.

LÄHTEET

Atlassian s.a. What is Git? WWW-dokumentti. Saatavissa: https://www.atlassian.com/git/tutorials/what-is-git [viitattu 21.10.2024].

BitBucket s.a. Pricing. WWW-dokumentti. Saatavissa: https://www.atlassian.com/software/bitbucket/pricing [viitattu 25.11.2024].

Dustin Briles. 2022. OneDrive and git - don't do it. Blogi. Päivitetty 14.1.2022. Saatavissa: https://dustinbriles.com/onedrive-and-git-dont-do-it/ [viitattu 4.11.2024].

Fidelix s.a. WWW-dokumentti. Saatavissa: https://www.fidelix.com/fi/fidelix/ [viitattu 23.9.2024].

Fidelix s.a. Alakeskus versiot Fx2020/Fx2025/Fx2030/Fx3000c + FxSpider-rev1/rev2/rev3/rev4 versio historia. WWW-dokumentti. Saatavissa: https://fidelixfi.sharepoint.com/sites/downloads/SitePages/CONTROLLER_CHANGES_FI.aspx#alakeskus-versiot-fx2020-fx2025-fx2030-fx3000c-%2B-fxspider-rev1-rev2-rev3-rev4-versio-historia [viitattu 23.9.2024].

Fidelix. 2014. Tuotemuutos, ala-asema FX-2030 -> FX-2030A. Tekninen tiedote 025 (2014/03). PDF-dokumentti. Saatavissa: https://fidelixfi.sharepoint.com/sites/downloads/files/Forms/AllItems.aspx?id=%2Fsites%2Fdownloads%2Ffiles%2FManuals%5FDocumentation%5FExamples%2FTechnical%5Fupdates%2FFx%5FTechnical%5Fbulletin%5F025%5Ffin%5Fv10%2Epdf&parent=%2Fsites%2Fdownloads%2Ffiles%2FManuals%5FDocumentation%5FExamples%2FTechnical%5Fupdates [viitattu 23.9.2024].

Fidelix. 2024. FX-3000-X. WWW-dokumentti. Saatavissa: https://www.fidelix.com/fi/tuotteet/fx-3000-x/ [viitattu 4.11.2024].

Free Software Foundation. 2022. GNU RCS. WWW-dokumentti. Saatavissa: https://www.gnu.org/software/rcs [viitattu 21.10.2024].

Git s.a. Git-kotisivu. WWW-dokumentti. Saatavissa: https://git-scm.com/ [viitattu 23.9.2024].

Git. 2024. Download mirrors index. WWW-dokumentti. Saatavissa: https://mirrors.edge.kernel.org/pub/software/scm/git/ [viitattu 23.9.2024].

GitHub s.a. Pricing. WWW-dokumentti. Saatavissa: https://github.com/pricing [viitattu 4.11.2024].

GitLab s.a. Pricing. WWW-dokumentti. Saatavissa: https://about.gitlab.com/pricing/ [viitattu 25.11.2024].

infoteam Software s.a. PLC Programming Systems. WWW-dokumentti. Saatavissa: https://infoteam.de/en/our-know-how/plc-programming-systems [viitattu 23.9.2024].

McMillan T. 2021. A History of Version Control. WWW-dokumentti. Päivitetty 24.9.2021. Saatavissa: https://tarynwritescode.hashnode.dev/a-history-of-version-control [viitattu 21.10.2024].

Rahul Bakale. 2023. LinkedIn-päivitys 2023. LinkedIn-verkkoyhteisöpalvelu. Saatavissa: https://www.linkedin.com/posts/rahul-bakale-443a196a_today-i-learned-onedrive-can-be-used-as-activity-7105924724634886144-D-oF [viitattu 4.11.2024].

The Apache Software Foundation. 2024. News. Apache Subversion. WWW-dokumentti. Saatavissa: https://subversion.apache.org/news.html [viitattu 23.9.2024].

TheirStack s.a. Companies that use GitHub. Saatavissa: https://theirstack.com/en/technology/github [viitattu 21.10.2024].

@Helloall_16. 2024. Reddit-päivitys 2024. Reddit-sosiaalinen verkkosivusto. Saatavissa: https://www.reddit.com/r/git/comments/1cotkpb/onedrive_folder/?rdt=58530 [viitattu 4.11.2024].

Liite 1

KUVALUETTELO