Tampereen ammattikorkeakoulu

Git-projekti

Olavi Kurola

Tammikuu 2020

Ohjelmistuotanto

SISÄLLYS

1 GITHUBIN KÄYTTÖ

Päätin jo heti projektin alussa ryhtyä kamppailemaan Visual Studion GitHub-integraation kanssa, ja tutustua GitHubiin sitä kautta. Jotta GitHubin Visual Studio -laajennusta pystyi käyttämään, se täytyi ensin ladata Visual Studion laajennuskirjastosta. Sen asentaminen oli hyvin suoraviivaista, muutaman klikkauksen jälkeen Visual Studio täytyi vain käynnistää uudestaan. Loin ensikertalaisena tunnukset GitHubiin, ja yhdistin GitHubin käyttäjäni Visual Studioon yksinkertaisesti vain kirjautumalla sisään Visual Studion GitHub -laajennuksen kautta.

Aloitin testaamalla GitHub-repositorion tekemistä suoraan tämän laajennuksen kautta. Laajennus loi sekä GitHubiin repositorion, että omalle koneelleni paikallisen repositorion. Seuraavaksi tein tätä Git-projektia varten kokonaan uuden C++ projektin Visual Studioon, ja lisäsin projektiin main.cpp-tiedoston. Kirjoitin laskinohjelmalle hyvin karun rungon, ja loin tälle projektille GitHub-repositorion lisäämällä projektin lähdeohjaukseen Visual Studiossa (Add to Source Control), jonka jälkeen käyttämällä projektin synkronointa GitHub-laajennuksessa saatiin tämä projekti GitHubiin. Poistin aikaisemmin testimielessä tehdyn GitHub-repon; tätä ei enää tarvittu, sillä laajennus teki itse uuden repositorion projektin synkronoinnin yhteydessä. GitHub repositorio tälle projektille löytyy osoitteesta: https://github.com/phenomi/GitTestProject

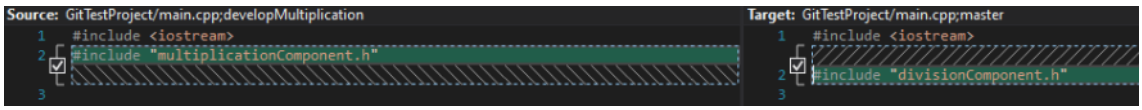
Tämän jälkeen tein ensimmäisen uuden haaran: developDivision. Lisäsin haarassa projektiin kaksi uutta tiedostoa jakolaskutoimintoa varten, ja committasin muutokset paikalliseen Git-repositorioon. Tämän jälkeen puskin muutokset vielä GitHubin repositorioon asti, ja synkronoin repositoriot keskenään. Tein vielä askel kerrallaan muutamia yksinkertaisia muutoksia joiden välissä aina commitoin muutokset paikalliseen repositorioon, ja puskin + synkronoin GitHubin repositorioon. Viimeisteltyäni jakolaskuominaisuuden loin taas uuden haaran kertolaskuominaisuutta varten, jolle tein samanlaisen käsittelyn. Pikkuisia melkeinpä yhden rivin muutoksia commitattuina, puskettuna ja synkronoituna GitHubin repositorioon muutos kerrallaan.

Kun olin saanut kummatkin näistä haaroista valmiiksi, siirryin takaisin master-haaraan. Myös tässä master-haarassa main.cpp:hen tehdyt muutokset tehtiin pieni osio kerrallaan, jonka jälkeen puskettiin ja synkattiin muutokset GitHubin repoon.

Master-haarassa tehtyjen viimeistenkin hienosäätöjen ja niiden commitoinnin jälkeen oli aika siirtyä mielenkiintoisimpaan vaiheeseen. Oli siis aika mergettää eli yhdistää jako- ja kertolaskuhaarat takaisin master-haaraan. Tämä onnistui Visual Studion GitHub-laajennuksen kautta yllättävän kivuttomasti. Graafisen käyttöliittymän dropdown-listalta pystyi valitsemaan halutun haaran, joka yhdistetään, ja haluttu haara johon yhdistäminen kohdistuu. Valitsin tähän siis yhdistettäväksi haaraksi ensin developDivision-haaran, ja kohteeksi master-haaran. Muutaman hiiren klikin jälkeen haarat olivatkin yhdistetty ja samalla commitattu (lisäoptio yhdistämiselle). Koodin ja lisättyjen tiedostojen silmäilyn jälkeen totesin, että yhdistäminen onnistui tässä tapauksessa täydellisesti. Koska kaikki näytti olevan kunnossa, vein paikallisen commitin GitHubiin asti, ja synkronoin repositoriot. Tämän jälkeen sama tehtiin myös developMultiplication-haaralle. Tässä ilmeni kuitenkin odottamattomia ongelmia (kuva 1):

KUVA 1. Merge-konflikti sisällytetyistä header-tiedostoista

Kuvaa piti hieman rajata, jotta se näkyi edes jokseenkin hyvin (ainakin Word dokumentti -muodossa zoomaaminen auttoi vielä tähän ongelmaan), mutta tässä siis sekä jako- että kertolaskujen kehityshaaroissa main.cpp-tiedostoon lisätyt includet aiheuttivat konfliktin. Haaroja yhdistäessä Git ei automaattisesti ymmärtänyt, että kummassakin kehityshaarassa lisätyt header-tiedostojen includet kuuluivat mergeen mukaan. Tämä oli Visual Studion GitHub-laajennuksessa kyllä yllättävän helppo ratkaista; käyttöliittymästä sai valita checkbox-tyylisesti rivit, jotka halusi mergeen mukaan (kuva 2).



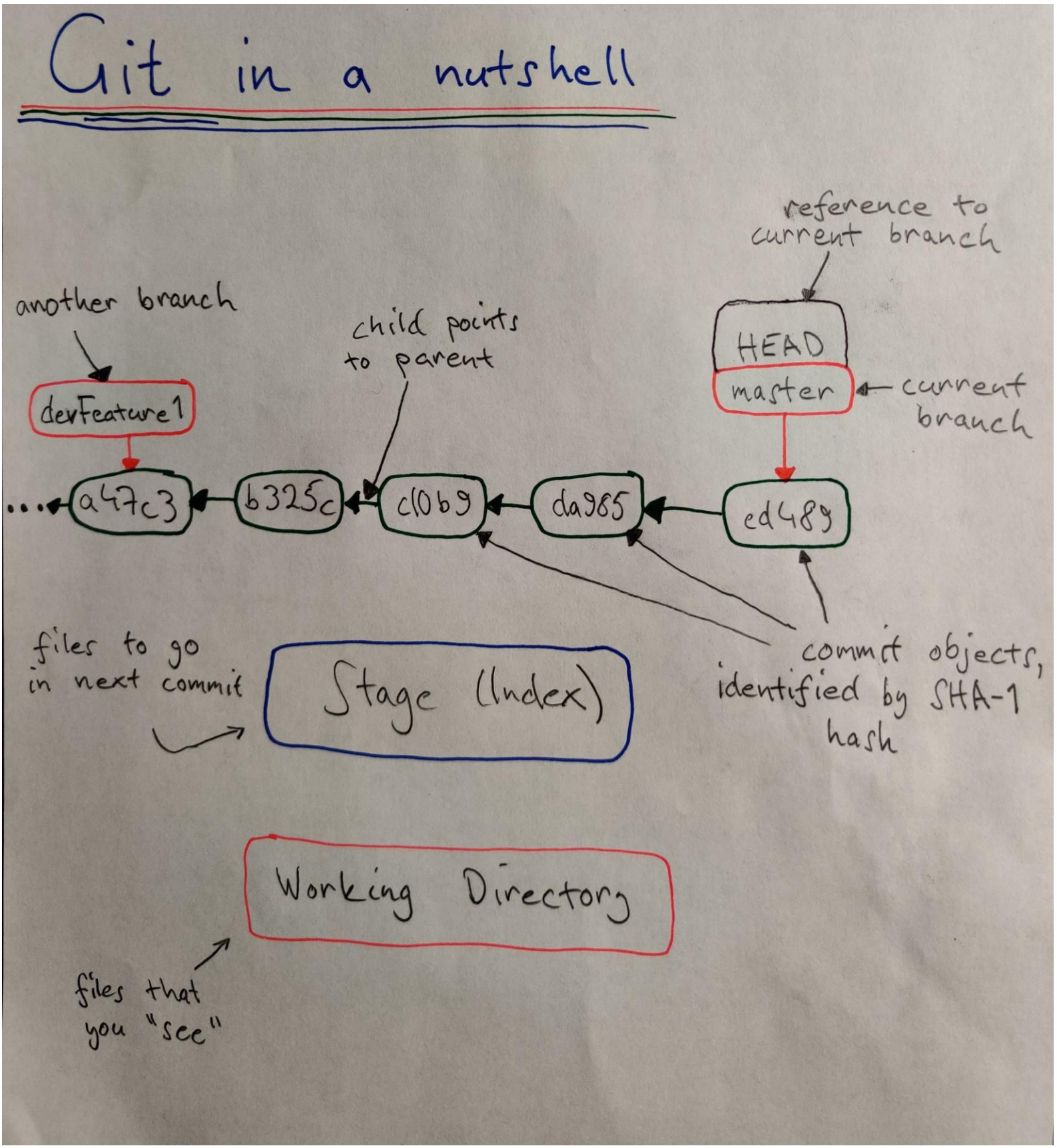
KUVA 2. Checkbox-tyylinen valikko tiedostojen sisällytykselle

Valitsin siis kummatkin header-tiedostojen includet sisällytettäväksi tähän mergeen. Kun konfliktit oli ratkaistu, pystyi mergen suorittamaan loppuun. Tässä ei tullut enää ongelmia, vaan nämä muutokset sai helposti commitattua, puskettua ja synkronoitua GitHubin repositorioon.

Ensimmäiset kaksi mergeä tehdessä unohdettiin kuitenkin käyttää pull requesteja, joten repositorioon lisättiin vielä yksi uusi haara potenssilaskun kehitystä varten. Ennen haaran lisäystä kokeiltiin vielä poistaa vanhat haarat (developDivision/Multiplication), sekä paikallisesta repositoriosta että remote-repositoriosta. Tämä onnistui mutkittomasti.

Uuden developExponentiation-haaran kehityksen valmistuttua tehtiin pull request master-haaraan. Tämä pull request käytiin hyväksymässä GitHubin repositorion syövereistä, ja minun tapauksessani asia onnistui ilman konflikteja. GitHub ilmoitti minulle, että ylimääräinen haara on nyt turvallista poistaa, joten poistin tämänkin haaran onnistuneen mergen jälkeen. Koska pull request käytiin hyväksymässä GitHubin etärepositoriossa, ja haarojen yhdistyminen tapahtui siellä, täytyi Visual Studion GitHub-laajennuksen sisällä tehdä pull-toiminto etärepositoriosta, jotta paikallinen repositoriokin saatiin synkronoitua etärepositorion kanssa samaan tilaan. Lopuksi testasin ohjelman toimivuutta ajamalla sitä master-haaran kautta. Kaikki toimi niin kuin pitikin, ja Git-projekti oli tältä osin menestys.

2 KUVA GITIN KÄYTÖSTÄ



3 GITIN KESKEISET KÄSITTEET JA IDEOLOGIA

Gitin keskeisiin ideoihin kuuluu sen tarjoama sulava ja joustava työnkulku (Git workflow). Tässä on tarkoituksena luoda (mieluiten pieniä/lyhytkestoisia) kehityshaaroja päähaarasta, joka mahdollistaa joustavan ja rinnakkaisen kehityksen projektille. Jokaisen haaran sisällä olisi hyvä tehdä toistuvia committeja, aina kun jotain konkreettista saadaan aikaiseksi. Toistuvat commitit luovat haaroihin turvallisia palautuspisteitä, joihin voidaan tarvittaessa aina palata. Säännölliset commitit lisäävät myös repositorion historian luettavuutta ja ymmärrettävyyttä.

Kun haarassa työskentely saadaan valmiiksi, voidaan avata ns. pull request, joka lähettää projektin adminille liittymispyynnön kehityshaarasta takaisin päähaaraan. Terveellisessä työympäristössä näitä pyyntöjä tulisi kommentoida, testata ja evaluoida työryhmän kesken. Jos kaikki näyttää olevan reilassa, tai kun muutokset on saatu valmiiksi, voidaan pull request hyväksyä ja suorittaa päähaaraan liittyminen (merge). Mergen tapahtuessa GitHub säilyttää koko liittyvän kehityshaaran historian ja liittää sen päähaaran historiaan. Onnistuneen mergen jälkeen kehityshaaroja voi siis jopa poistaa, jos niille ei enää ole tarvetta, tai jos niissä ei tulla enää jatkokehittämään ominaisuutta.

Kun kaikki kehityshaarat ovat yhdistettyinä takaisin päähaaraan, voi työryhmä/organisaatio sopia ja valita sopivan aikataulutuksen ohjelman tai tuotteen kannalta. Tämän Gitin työkulun tarkoituksena on varmistaa, että julkaistu ohjelmakoodi on jämäkkää ja laadukasta, ja monihaaraisen työskentelyn ansiosta kehittäminen on myös nopeaa, sillä kehittämistä tapahtuu rinnakkain, eikä yleensä kenenkään tarvitse odotella jouten jonkin ominaisuuden valmistumista ennen kuin pääsee työn kimppuun. Oikeaoppisesti suoritetut avoimet lähdekoodikatselmoinnit pull request -pyyntöjen yhteydessä takaavat myös sen, että kaikki ovat tietoisia ohjelmassa tapahtuvista muutoksista ja projektin suunnasta. Jatkuva kehityksen näkeminen auttaa myös ryhmän työmoraalin kanssa, ja saa projektin tuntumaan siltä, että asiat oikeasti etenevät koko ajan.

4 GIT CHEATSHEET

fork kopio jostakin repositoriosta

clone repon kloonaaminen esim. etäreposta paikalliseen repoon

branch haara, jonka kehitys etenee omassa ”tilassaan” erikseen muista haaroista, voidaan lopuksi liittää (merge) takaisin päähaaraan

staging muutosten vieminen eräänlaiselle valmistelualueelle, josta muutokset voi sitten viedä paikalliseen repositorioon asti (commit)

commit muutosten vieminen valmistelualueelta paikalliseen repositorioon

push pusketaan paikalliseen repositorioon suoritetut commitit etärepositorioon asti

pull request pyyntö tuoda esim. jossakin haarassa kehitettyjä ominaisuuksia takaisin päähaaraan

rebase rebase-komento antaa mahdollisuuden muuttaa jo tehtyjä committeja, eli muuttaa repositorion historiaa (huono tapa jos commitit puskettu jo etärepositorioon!!)

Gitin suosima työnkulku:

1. luo master-haarasta periytyvä kehityshaara
2. lisää committeja usein, jotta saadaan snapshotteja kehityspisteistä
3. avaa pull request -pyyntö, jotta muutokset saadaan takaisin päähaaraan
4. pyydettyjen pull requestien koodin katselmointi ryhmittäin/tiimeittäin
5. jos kaikki OK, liitä haara merge-komennolla
6. julkaise ohjelmasta/tuotteesta julkaisuversio