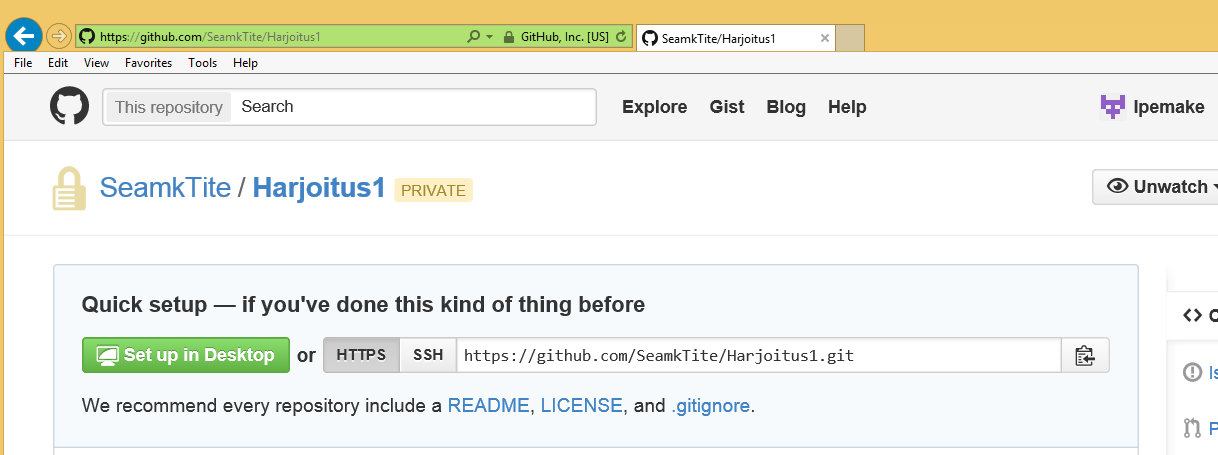
## Harjoituksia

### Valmistelut

* Luo itsellesi tunnus Githubiin.
* Asenna työkalut koneellesi
* Valmistele itsellesi repository Githubissa

### Harjoitus 1

1. Kirjaudu Githubiin ja mene Githubin verkkosivulla omaan repositorioon. Kopioi itsellesi repositorion osoite. Osoite on muotoa https://github.com/SeamkTite/Harjoitus1.git.



**Tee Fork (kopioi koko repositorio itsellesi githubissa). Sinun pitää siis tehdä kopio opettajan repositoriosta itsellesi.**

Tee omalle koneellesi työhakemisto. Avaa Git Bash -komentoriviohjelma ja mene sillä työhakemistoosi.

~

**$ cd c:**

/c

**$ cd tmp/GitHarjoitus**

1. Seuraavaksi kopioidaan Githubissa oleva repositorio omalle kovalevylle. Tämä tehdään komennolla git init.

/c/tmp/GitHarjoitus

**$ git clone https://github.com/omagittunnus/Harjoitus1.git**

Cloning into 'Harjoitus1'...

Username for 'https://github.com': **lpemake**

Password for 'https://lpemake@github.com':

1. Mene komentorivillä edellä kopioimaasi kansioon:

/c/tmp/GitHarjoitus

**$ ls**

Harjoitus1

/c/tmp/GitHarjoitus

**$ cd Harjoitus1/**

1. Seuraavaksi muokataan versionhallintaan talletettua tiedostoa. Avaa tiedosto README.md Notepadillä (tai luo se, jos sitä ei ole olemassa). Kirjoita tiedostoon jotain, esimerkiksi ”muutos 1”.

**$ notepad README.md**

1. Anna komento git status, joka näyttää työhakemiston tilan.

**$ git status**

Jos lisäsit juuri uuden README.md-tiedoston, on se untracked-tilassa. Jos README.md kopioitiin remote-repositoriosta, on se modified-tilassa. Molemmissa tapauksissa voit antaa komennon git add, jolloin tiedosto menee staged-tilaan.

1. Anna tiedostolle komento git add ja tarkista sen jälkeen tila git status -komennolla.

**$ git add README.md**

**$ git status**

Statuksesta nähdään, että tiedosto README.md on nyt valmis vietäväksi versionhallintaan.

1. Vie README.md-tiedosto nyt repositorioon git commit-komennolla. Jokaiseen commitiin pitää lisätä viesti, jossa kerrotaan muutoksesta. Katso commitin jälkeen vielä status.

**$ git commit -m "ensimmäinen muutos"**

**$ git status**

Tiedosto README.md on nyt paikallisessa repositoriossa. Seuraavaksi uusi tai muutettu tiedosto viedään Githubiin.

1. Vie paikalliseen repositorioon tehdyt muutokset Githubiin komennolla git push.

**$ git push**

1. Mene Githubin sivulle omaan repositorioosi ja päivitä näkymä. Tekemäsi muutokset ja commitiin antamasi viesti näkyvät nyt Githubissa.

### Harjoitus 2

Harjoituksessa kokeillaan tiedoston lisäämistä ja useamman tiedoston muokkaamista.

1. Hae aiempi harjoitus (1) Githubista (voit hakea oman harjoituksesi tai opettajan tekemän harjoituksen)

**$ cd /c/tmp/**

**$ git clone https://github.com/SeamkTite/Harjoitus1b.git**

**$ cd Harjoitus1b/**

1. Tee uusi tiedosto tiedosto.txt ja lisää siihen tekstiä. Muokkaa myös tiedostoa README.md. Katso sen jälkeen status.

**$ notepad tiedosto.txt**

**$ notepad README.md**

**$ git status**

Tiedosto README.md on nyt modified-tilassa ja tiedosto tiedosto.txt untracked-tilassa.

1. Lisää molemmat tiedostot staging-alueelle komennolla git add ja katso status.

**$ git add README.md tiedosto.txt**

**$ git status**

Sekä uusi että muokattu tiedosto ovat nyt valmiina vietäväksi versionhallintaan. Voit lisätä useita tiedostoja komennolla git add . (piste).

**$ git commit -m "uudet rivit lisätty"**

1. Vie paikalliseen repositorioon tehdyt muutokset Githubiin komennolla git push.

**$ git push**

Voit vielä katsoa, menivätkö muutokset Githubiin saakka.

### Harjoitus 3

Versionhallinnan avulla voit pitää ohjelmistoprojektisi ajan tasalla, kun työskentelet usealla koneella. Tekemäsi tiedostot ovat tallessa palvelimella olevassa repositoriossa (esimerkiksi Github). Voit kopioida palvelimella olevan repositorion mille koneelle tahansa. Tällöin tietoa ei tarvitse siirtää muistitikun tai jonkin muun pilvipalvelun kautta.

Seuraavassa harjoituksessa on tarkoitus kokeilla GitHubin käyttöä kahdesta eri työhakemistosta. Tällöin käytössä on kaksi paikallista repositoriota Githubin repositorion lisäksi. Todellisessa tilanteessa paikalliset repositoriot olisivat eri koneilla, mutta tässä harjoituksessa sinulla voi olla kaksi eri hakemistoa samalla koneella.

1. Tuhoa edellinen harjoitus omalta levyltäsi. Tee kaksi hakemistot matti ja teppo. Hae näihin hakemistoihin Githubista edellinen harjoitus. Käynnistä kaksi Git Bash -instanssia (matti ja teppo) ja siirry näihin hakemistoihin

**GitBash-ikkuna matti:**

**$ cd c:**

**$ cd tmp**

**$ mkdir matti**

**$ cd matti**

**GitBash-ikkuna teppo:**

**$ cd c:**

**$ cd tmp**

**$ mkdir teppo**

**$ cd teppo**

1. Kloonaa uusi paikallinen repositorio Githubista molempiin hakemistoihin. Anna perään uuden repositorion nimi. Siirry repositorion hakemistoon molemmissa ikkunoissa

**matti:**

**$ git clone https://github.com/SeamkTite/Harjoitus1.git matti**

**$ cd matti**

**teppo:**

**$ git clone https://github.com/SeamkTite/Harjoitus1.git teppo**

**$ cd teppo**

1. Katso, mitä tiedostoja hakemistoissa on. Voit katsoa myös statuksen.

**$ cd repo2:**

**$ ls**

**$ git status**

1. Siirry Matin Git Bash-ikkunaan. Muokkaa tiedostoa README.md. Anna tiedostolle komento git add ja tee sitten commit. Voit myös jättää add-komennon välistä antamalla commitille option -am (tai -a -m) Vie muutokset Githubiin komennolla git push.

**matti:**

**$ notepad README.md**

**$ git commit -am "taas uusi lisäys"**

**$ git push**

1. Vaihda toisen repositorion (teppo) Git Bash -ikkunaan. Voit katsoa README.md -tiedostoa ja huomaat, että se ei ole muuttunut. Alkuperäisestä paikallisesta repositorioista Githubiin viedyt muutokset täytyy ”vetää” vielä repo2:een. Tämä tehdään komennolla git pull.

**teppo:**

**$ git pull**

README.md-tiedostoon tehdyt muutokset näkyvät nyt myös repo2:ssa.

### Harjoitus 4

Tässä harjoituksessa simuloidaan tilannetta, jossa kaksi henkilöä työskentelee saman projektin kanssa. Harjoituksen voi tehdä pareittain tai yksin. Pareittain tehtäessä molemmat henkilöt kopioivat Githubiin aiemmin luodun repositorion itselleen. Yksin harjoiteltaessa käytetään kahta erillistä repositoriota, kuten edellisessä harjoituksessa. Tässä harjoituksessa ei käytetä vielä haarautumia (branch).

1. Avaa Git Bash -työkalu molempia käyttäjiä (tai hakemistoja varten) ja kloonaa edellinen harjoitus hakemistoihin matti ja teppo (Harjoitus 3, kohdat 1 ja 2). Jos jatkat suoraan edellisestä harjoituksesta, tätä vaihetta ei tarvitse tehdä.
2. Matti muokkaa nyt tiedostoa README.md ja lisää siihin uuden rivin (esim. ”matin rivi”). Matti vie muutokset paikalliseen repositorioon.

**matti:**

**$ notepad README.md**

**$ git commit -am "masan muutos"**

Matti ei vienyt vielä muutosta Githubiin.

1. Teppo muokkaa myös tiedostoa README.md ja lisää uuden rivin (”tepon rivi”). Teppo vie muutokset omaan paikalliseen repositorioonsa.

**teppo:**

**$ notepad README.md**

**$ git commit -am "tepon muutos"**

Matti ja Teppo tekivät muutokset ja tallensivat ne omiin repositoriohinsa. Kumpikaan henkilöistä ei tiedä vielä toistensa muutoksista eikä muutokset ole vielä Githubissa.

1. Seuraavaksi Matti vie omat muutoksensa Githubiin.

**matti:**

**$ git push**

1. Tämän jälkeen Teppo yrittää myös viedä omat työnsä Githubiin.

**teppo:**

**$ git push**

Tepon git push ei kuitenkaan onnistu, sillä joku muu on vienyt muutoksen samasta tiedostosta Githubiin. Git pushin antama virheilmoitus kertoo, mistä on kyse.

! [rejected] master -> master (fetch first)

error: failed to push some refs to 'https://github.com/SeamkTite/Harjoitus1.git'

hint: Updates were rejected because the remote contains work that you do

hint: not have locally. This is usually caused by another repository pushing

hint: to the same ref. You may want to first integrate the remote changes

hint: (e.g., 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

Ennen komentoa git push Tepon täytyy synkronoida muiden tekemät muutokset omaan paikalliseen repositorioon komennolla git pull.

1. Anna komento git pull Tepon ikkunassa.

**teppo:**

**$ git pull**

Komento git pull haki Matin tekemät muutokset Githubista ja yritti yhdistää ne Tepon muutosten kanssa. Matti ja Teppo olivat kuitenkin muokanneet saman tiedoston samoja rivejä, joten automaattinen yhdistäminen (merge) ei onnistu. Git antaa seuraavan ilmoituksen, jossa on myös ohjeet tilanteen korjaamiseksi:

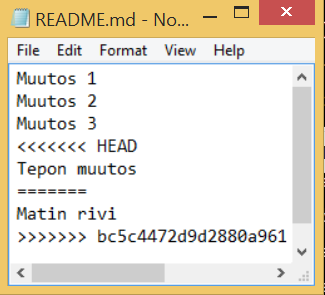
Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

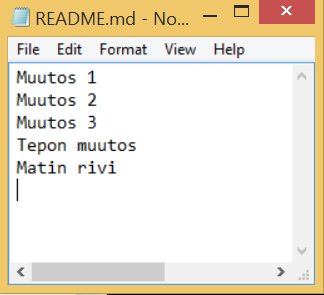
Automatic merge failed; fix conflicts and then commit the result.

Tepon täytyy nyt ratkaista konflikti. Tämä tapahtuu niin, että Teppo avaa tiedoston README.md ja muuttaa sen sellaiseksi, kuin sen pitää olla. Käytännössä Matti ja Teppo joutuvat tekemään yhteistyötä, että molempien tekemät muutokset tulevat huomioiduksi oikein.

1. Avaa tiedosto README.md Notepadilla ja poista siitä Gitin mergen tekemät huomautukset



Jätä tiedostoon Matin ja Tepon tekemät rivit.



1. Tallenna tiedosto ja lisää se versionhallintaan commitilla Tepon ikkunassa. Tämän jälkeen korjattu README.md voidaan viedä myös Githubiin

**teppo:**

**$ git commit -am "merge korjattu"**

**$ git push**

1. Myös Matti haluaa itselleen Tepon muokkaaman version README.md-tiedostosta (jossa on nyt siis molempien tekemät muutokset).

**matti:**

**$ git pull**

Tässä vaiheessa Matilla ja Tepolla on sama versio README.md -tiedostosta kuin Githubissa. Commitien historiaa voi tarkastella komennolla git log. Historian tarkastelu on kuitenkin helpompaa graafisilla työvälineillä.

**$ git log**

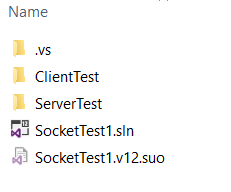
### Harjoitus 5

Tässä harjoituksessa on tarkoitus viedä Visual Studiolla tehty ohjelmistoprojekti Githubiin. Ohjelmistoprojekti koostuu usein useasta eri moduulista (Visual Studion projektista). Samaan kokonaisuuteen kuuluvat projektit kannattaa samaan Visual Studion Solutioniin ja viedä tämä kokonaisuus Githubissa yhteen repositorioon.

Java-projekteissa voi toimia vastaavalla tavalla. Esimerkiksi NetBeansissä on monia Gitin käyttöä helpottavia toimintoja.

1. Tee Visual Studio -projekti. Voit lisätä saman Solutionin alle useita aliprojekteja, jos haluat.

*Käytä tässä harjoituksessa projektia: Y:\Makela\_Petteri\TITE16\Verkko-ohjelmointi\****SocketTestThreadLoop***



1. Avaa Git Bash -työkalu ja mene siihen hakemistoon, missä on solution-tiedosto (.sln).

**$ cd c:**

**$ cd tmp/Test/TestApplication**

1. Alusta versionhallinta komennolla git init.

/c/tmp/Test/TestApplication

**$ git init**

Komento luo uuden alihakemiston nimeltä .git. Tässä hakemistossa on paikallinen repositorio. Kaikkia ohjelmistoprojektin tiedostoja ei ole kuitenkaan tarkoitus viedä versionhallintaan. Yleensä versionhallintaan viedään vain ne tiedostot, joita tarvitaan ohjelmistoprojektin kääntämiseksi. Näitä tiedostoja ovat esimerkiksi lähdekoodi- ja projekti- ja konfigurointitiedostot. Visual Studion tuottamia käännöstuloksia, kuten debug- ja release-kansioissa olevia tiedostoja ei viedä versionhallintaan.

Pois jätettävät tiedostot määritellään tiedostossa .gitignore. Netistä Visual Studio -projekteja varten valmiita gitignore-tiedostoja, joissa on tarpeelliset asetukset valmiina.

1. Hae Visual Studion gitignore-tiedosto osoitteesta https://github.com/github/gitignore/blob/master/VisualStudio.gitignore. Kopio tiedoston sisältö uuteen tiedostoon .gitignore.txt. Lisää .gitignore repositorioon. Kohdista komento git add vain .gitignore -tiedostoon. Vie tämän jälkeen .gitignore paikalliseen repositorioon.

/c/tmp/Test/TestApplication

**$ git add .gitignore**

**$ git commit –m “gitignore lisätty”**

1. Lisää tämän jälkeen projektin (Solutionin) kaikki tiedot staging-alueelle komennolla git add . (piste). Katso sen jälkeen status.

**$ git add .**

**$ git status**

1. Anna komento git commit, joka vie tiedostot paikalliseen repositorioon.

**$ git commit -m "eka versio"**

[master 40d70f8] eka versio

8 files changed, 260 insertions(+)

create mode 100644 MathLibrary/MathLibrary.csproj

create mode 100644 MathLibrary/MathUtilities.cs

create mode 100644 MathLibrary/Properties/AssemblyInfo.cs

create mode 100644 TestApplication.sln

create mode 100644 TestApplication/App.config

create mode 100644 TestApplication/Program.cs

create mode 100644 TestApplication/Properties/AssemblyInfo.cs

create mode 100644 TestApplication/TestApplication.csproj

Huomaa, että listassa on vain kääntämisen kannalta tarpeelliset tiedostot. Visual Studion käännöksen seurauksena syntyneet debug- ja release-hakemistot sekä niiden sisällöt jätettiin huomiotta.

1. Seuraavaksi projekti viedään Githubiin. Komento git push ei kuitenkaan toimi vielä, sillä tämän projektin repositoriota ei kloonattu alun perin Githubista. Repositorio pitää siis luoda ensin. Käy luomassa uusi tyhjä repositorio omalla Githubin verkkosivulla. Älä lisää .gitignore-tiedostoa Githubissa. Anna repositorion nimeksi projektin hakemiston nimi.
2. Seuraavaksi lisätään remote-repositorio komennolla git remote add [shortname] [url].

**$ git remote add origin https://github.com/SeamkTite/LiukuvaArvoDLL.git**

1. Lopuksi projekti viedään Githubiin.

**$ git push -u origin master**

1. Hae vielä projekti Githubista johonkin toiseen hakemistoon (tai pyydä, että toinen käyttäjä hakee). Käynnistä uusi Git Bash tähän hakemistoon.

/C/tmp/New folder

**$ git clone https://github.com/SeamkTite/LiukuvaArvoDLL**

Avaa projekti Visual Studiolla ja käännä se. Tee tiedostoihin muutoksia. Jos käytössäsi on TortoiseGit, on muutetut tiedostot merkitty punaisella symbolilla. Anna komento git commit -am viedäksesi muutokset paikalliseen repositorioon. Vie muutokset sitten Githubiin.

/C/tmp/New folder

**$ cd LiukuvaArvoDLL/**

**$ git commit -am "Divide method added"**

**$ git push**

1. Vaihda vielä alkuperäiseen hakemistoon. Päivitä muutokset myös tähän hakemistoon komennolla git pull.

### Harjoitus 6

Tässä harjoituksessa harjoitellaan haarautumien (branch) tekemistä ja niiden yhdistämistä.

1. Forkkaa projekti <https://github.com/SeamkTite/TestApplication> itsellesi Githubissa.
2. Avaa Git Bash –työkalu.
3. Kloonaa edellä tehty harjoitus TestApplication komennolla git clone.

/C/tmp/

**$ git clone https://github.com/Oma/TestApplication**

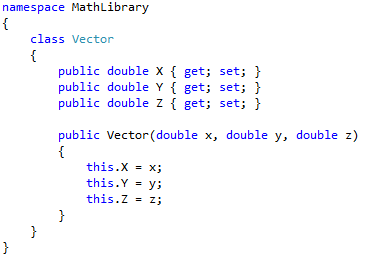
**$ cd TestApplication/**

1. Tee uusi haara nimeltä vectortopic ja siirry tähän haaraan.

**$ git branch vectortopic**

**$ git checkout vectortopic**

1. Avaa projekti Visual Studiolla. Lisää MathLibrary-kirjastoon uusi luokka Vector.



1. Lisää uusi tiedosto versionhallintaan.

**$ git add .**

**$ git status**

**$ git commit –m “new file added”**

**$ git status**

1. Seuraavaksi joudutkin tekemään korjauksen tuotantoversioon. Palaa ensin master-haaraan. Tee sitten hotfix-haara. Muuta sitten tiedostoa MathUtilities.cs. Kommitoi tämän jälkeen muutokset.

**$ git checkout master**

**$ git checkout –b hotfix**

**$ notepad MathLibrary/MathUtilities.cs**

**$ git commit -a -m "Comments added"**

1. Tässä vaiheessa voit testata hotfix-haaran, että se kääntyy ja toimii oikein. Tämän jälkeen hotfix voidaan yhdistää tuotantohaaraan. Vaihda ensin master-haaraan. Yhdistä sitten hotfix-haara master-haaraan.

**$ git checkout master**

**$ git merge hotfix**

Merge oli nyt fast forward.

1. Voit tuhota nyt hotfix-haaran.

**$ git branch –d hotfix**

Muutos tuotantoversioon on nyt tehty. Voit palata takaisin kehitysversioon.

1. Siirry haaraan vectortopic.

**$ git checkout vectortopic**

1. Muuta tiedostoa Vector.cs. Kommitoi sitten muutokset.

**$ git commit -a -m "Vector modified"**

Kehitysversio on nyt valmis. Testauksen jälkeen voit yhdistää sen päähaaraan.

1. Siirry master-haaraan.

**$ git checkout master**

**$ git merge vectortopic**

### Harjoitus 7

* Tee harjoitus kirjan esimerkkiä soveltaen
  + [http://git-scm.com /book/en/v2/Git-Branching-Remote-Branches](http://git-scm.com/book/en/v2/Git-Branching-Remote-Branches)
* Kappale 3.5