**GitHUB / Git**

* Vad är Git?
* Vad är ­­­GitHub?
* Varför använda git?
* Registrering vid GitHub
* Installera Git for windows
* Att börja med GitHub
* Att skapa ett repositorie
* Att jobba aktivt med git

**Vad är Git?**

Git är ett versionhanteringssytem som är utarbetat främst för mjukvaruutvecklare och hantering av kod, speciellt i större projekt med flera utvecklare som jobbar mot samma kod. Med ett versionshanteringssystem menas att systemet hjälper användarna att hantera filer och spara dem i olika versioner så att man kan spåra förändringar och t.ex. gå tillbaka i historiken. Det ger också möjligheter för flera utvecklare att dela och kunna samarbeta kring gemensamma filer utan att riskera att man skriver över någon annans arbete. Ni kommer att använda er mycket av Git men även stöta på andra versionhanteringssytem t.ex. TFS (Team Foundation Server ), Subversion (svn).

**Vad är GitHUB?**

GitHub är en tjänst som ger oss utvecklare möjlighet att, via webben, skapa, hantera och lagra projekt och dess data via Git. GitHub ger oss också möjlighet att dela kod med andra, studera andras kod och använda andras kod för att bygga vidare på. GitHub är alltså en tjänst för "social coding". Man kan också använda GitHub för att dela utvecklingsfiler i privata grupper eller som ensam utvecklare hantera sina filer på ett smidigt sätt. Vi skulle kunna kalla det ett slags Facebook, fast för kod. På senare tid har dock allt fler upptäckt möjligheterna med Git och GitHub och nu för tiden hittar man inte bara mjukvaruprojekt på GitHub utan används även av författare, lärare och en mängd andra branscher.

**Varför använda git?**

Först och främst kommer Git och GitHub vara något du med säkerhet kommer stöta på i ditt kommande arbetsliv.

Ditt GitHub-konto kommer fungera som ett CV där du samlar dina projekt och där eventuella arbetsgivare kan se vad du kan och har producerat tidigare. Detta är något vi märker fler och fler arbetsgivare tycker är viktigt så att redan från början bygga upp sitt GitHubkonto bör vara en viktig punkt i din utveckling.

Du kommer enklare kunna hantera dina filer. Inga datorkrasher kommer få dina filer att försvinna (om du är noga med att synka dina filer med GitHub). Du kan också komma åt dina filer från flera olika datorer och enheter.

Du kan enkelt dela dina filer med dina lärare. Vid examinering av t.ex. koduppgifter vill läraren titta på din kod och kan då via GitHub enkelt komma åt denna. Vid t.ex. frågor kring eller problem med din kod kan läraren enkelt komma åt den senaste versionen.

GitHub är ett kraftfullt verktyg för att studera och lära sig av andras kod.

**Registrering vid GitHub**

<https://github.com/>

**Installera Git for windows**

<https://windows.github.com/>

Forsätt genom att ange dina uppgifter i konfigurationen för Git. Använd såklart ditt namn och e-postadress (din studentaddress)

$git config --global user.name "John Doe"

$git config --global user.email johndoe@example.com

Kontrollera dina inställningar via:

$git config --list

**Att börja med GitHub**

När man börjar jobba med Git och GitHub kommer man stöta på vissa begrepp som är viktiga att känna till. Vanliga ord som vi kommer använda är:

**Repositorie/Repository** - Kallar ofta förkortat för "repo" och är oftast ett projekt. Vi kan likna ett repositorie med en mapp/katalog innehållandes filer och undermappar som kan versionshanteras. Du skapar oftast ett repositorie per kurs eller ett repositorie per applikation.

**Commit** - Att "commit:a" sina ändringar talar om att man nu har uppdaterat sina filer och sparar dessa i en ny version, en commit.

**Forking** - Detta innebär att man hittar ett intressant projekt som någon annan lagt ut och som man själv vill bygga vidare på. "Forkar" man då projektet får man en kopia av alla filer som man kan fortsätta jobba vidare med på egen hand och kanske i slutändan föreslå dessa förändringar för orginalskaparen.

**Pull request** - Detta innebär att du har forkat ett projekt/repositorie och gjort förändringar i detta som du vill dela med dig av till orginalprojketet. Du kan då föreslå detta för repositorie-ägaren genom att göra en "pull request" som ägaren får godkänna.

Merge - Om ägaren av repositoriet tycker att dina ändringar är bra kan hon/han välja att baka in dessa i orginalkoden d.v.s. göra en "merge", en sammanslagning.

**Branch/Branching** - Du kommer att använda "branching" eller förgreningar i ditt repro för att undvika att ändra i projektets huvudgren. Detta är extra viktigt då man är flera personer som samarbetar i samma repro.

Vi kommer återkomma till vissa av dessa begrepp längre ner i texten.

**Att skapa ett repositorie**

Ett centralt begrepp inom Git är repositorie (repository) eller ofta förkortat som "repo". Detta kan förklaras som ett versionshanterat projekt. Du skapar oftast ett repositorie per kurs eller per applikation. Ett repositorie kan skapa på olika sätt. Antingen via att skapa ett helt nytt repo på din lokala dator eller genom att ladda ner ett redan påbörjat repo från t.ex. GitHub.

**Skapa ett nytt repositorie**

<https://github.com/> New repositoriy, ange lämpligt namn.

Starta git bash och ange komando

$git clone http:/github.com/user name/reponamnet.git

**git init**

Du har nu skapat ett helt eget lokalt repositorie och har förmodligen fått en mapp som heter samma som reponamnet du skapade i GitHub. Den mappen innehåller all information om detta repositorie och de versionsförändringar vi kommer göra. Denna mapp ska du alltså inte bry dig om så mycket utan bara låta Git sköta om. Du kan nu testa statusen på ditt repo genom att skriva:

**git status**

Vi har ännu inte lagt till några filer i vår versionshanterade katalog (vårt repo). Så skapa en fil index.html och spara i katalogen du nyss skapade. Kontrollera nu statusen igen med git status och observera nu skillnaden.

Git har alltså upptäckt att vi lagt till en ny fil i mappen och talar om att denna inte är versionshanterad. Varje fil i en mapp som är Git-hanterad kan antingen vara "tracked" eller "untracked". För att lägga till filen index.html vi nyss skapade skriver du:

**git add index.html**

Ovanstående kommando lägger till just filen index.html. Har man skapat flera filer samtidigt som vill lägga till använder man:

**git add .**

Vilket också är det vanligaste fallet. Kolla nu status på ditt repositorie igen med git status.

Filen är nu tillagd i vad man kallar "staging area" men den är fortfarande inte helt versionshanterad. Vi måste också göra en så kallad "commit" för att få till detta. En "commit" gör man när man suttit och jobbat ett tag med något, kanske skrivit en deluppgift, implementerat en funktion i sin applikation eller kanske bara tar en paus och vill spara undan det man gjort hittils. Man brukar säga att man ska "commit:a" sin kod ofta men inte så att det blir absurt. En "commit" ska också innehålla en bra kommentar som beskriver för andra vad du gjort sedan sist. Även om du kommer jobba ensam i vissa repositorier så var noga med att skriva bra och förklarande kommentarer till dina commit:s. För att göra en commit skriver du i terminalfönstret:

**git commit -m "I have created the file index.html, but it is still empty"**

Du bör såklart byta ut kommentaren innanför citattecknen mot en kommentar som passar i ditt fall. Det kan vara bra att skriva kommentarerna på engelska då man ibland stöter på problem med svenska tecken. Kontrollera nu ditt repositories status

**git status**

Testa nu att göra en ändring i filen index.html samt skapa en ny fil och lägg i mappen. Kolla status, lägg till den nya filen med

**git add .**

och gör en ny commit. Du bör nu ha ett lokalt repositorie med två versionshanterade filer och två stycken "commits".

Skapa ett repositorie på GitHub och klona ner lokalt

Men om man har ett repositorie på GitHub och vill utveckla lokalt på sin dator och sedan skicka upp förändringarna. Hur gör man då?

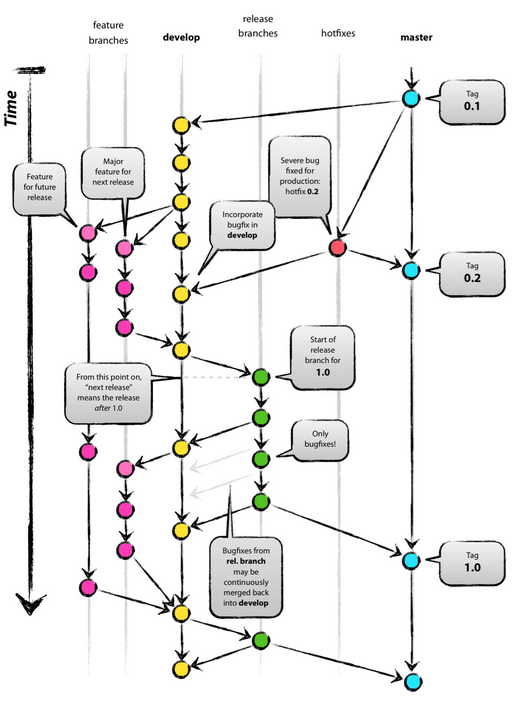
Börja med att skapa ett repositorie på ditt konto på GitHub. På ditt nya repositories första sida, nere till höger, kommer du se något som heter "HTTPS clone URL". Kopiera den sökväg som finns där. Den ska vi nu använda via terminalfönstret klona ner en kopia av repositoriet och få en koppling så vi kan skicka upp våra förändringar.

Gå till terminalfönstret och navigera dig till den mapp där du vill kopiera ner repositoriet och skriv:

**git clone https://github.com/xx222xx/myRepo.git**

där den sista delen är den address du kopierade. Du har nu sparat ner en exakt kopia av det som låg på GitHub. Alla filerna bör ha hamnat i en mapp som om man tittar på addressen ovan borde heta "myRepo". Du har nu ett lokalt skapat repositorie och kan jobba vidare med "add" och "commit" under arbetets gång. Observera dock att dessa förändringar bara sker lokalt än så länge.

**Att jobba Aktivt med Git**



**Hämta ett projekt från GitHub**

$ git clone <http://github.com/hamidsalehian/test.git>

cd test

**Skapa en feature bransch**

$ git checkout -b myfeature

Skapar ny branch "myfeature"

$ git commit -m “Skriv en kommentar”

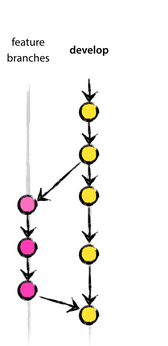
Eller

$ git commit -am “Skriv en kommentar”

# För att lägga till en skapad fil till repo och skriva kommentar

$ git status

För att se senaste ändringar



**git checkout develop**

**git merge myfeature**

**git status**

**git push origin myfeature**

**Major GIT commands:**

* **git config**

**Sets configuration values for your user name, email, gpg key, preferred diff algorithm, file formats and more. Example: git config --global user.name "My Name" git config --global user.email "user@domain.com" cat ~/.gitconfig [user] name = My Name email = user@domain.com**

* **git init**

**Initializes a git repository – creates the initial ‘.git’ directory in a new or in an existing project. Example: cd /home/user/my\_new\_git\_folder/ git init**

* **git clone**

**Makes a Git repository copy from a remote source. Also adds the original location as a remote so you can fetch from it again and push to it if you have permissions. Example: git clone git@github.com:user/test.git**

* **git add**

**Adds files changes in your working directory to your index. Example: git add .**

* **git rm**

**Removes files from your index and your working directory so they will not be tracked. Example: git rm filename**

* **git commit**

**Takes all of the changes written in the index, creates a new commit object pointing to it and sets the branch to point to that new commit. Examples: git commit -m ‘committing added changes’ git commit -a -m ‘committing all changes, equals to git add and git commit’**

* **git status**

**Shows you the status of files in the index versus the working directory. It will list out files that are untracked (only in your working directory), modified (tracked but not yet updated in your index), and staged (added to your index and ready for committing). Example: git status # On branch master # # Initial commit # # Untracked files: # (use "git add <file>..." to include in what will be committed) # # README nothing added to commit but untracked files present (use "git add" to track)**

* **git branch**

**Lists existing branches, including remote branches if ‘-a’ is provided. Creates a new branch if a branch name is provided. Example: git branch -a \* master remotes/origin/master**

* **git checkout**

**Checks out a different branch – switches branches by updating the index, working tree, and HEAD to reflect the chosen branch. Example: git checkout newbranch**

* **git merge**

**Merges one or more branches into your current branch and automatically creates a new commit if there are no conflicts. Example: git merge newbranchversion**

* **git reset**

**Resets your index and working directory to the state of your last commit. Example: git reset --hard HEAD**

* **git stash**

**Temporarily saves changes that you don’t want to commit immediately. You can apply the changes later. Example: git stash Saved working directory and index state "WIP on master: 84f241e first commit" HEAD is now at 84f241e first commit (To restore them type "git stash apply")**

* **git tag**

**Tags a specific commit with a simple, human readable handle that never moves. Example: git tag -a v1.0 -m 'this is version 1.0 tag'**

* **git fetch**

**Fetches all the objects from the remote repository that are not present in the local one. Example: git fetch origin**

* **git pull**

**Fetches the files from the remote repository and merges it with your local one. This command is equal to the git fetch and the git merge sequence. Example: git pull origin**

* **git push**

**Pushes all the modified local objects to the remote repository and advances its branches. Example: git push origin master**

* **git remote**

**Shows all the remote versions of your repository. Example: git remote origin**

* **git log**

**Shows a listing of commits on a branch including the corresponding details. Example: git log commit 84f241e8a0d768fb37ff7ad40e294b61a99a0abe Author: User <user@domain.com> Date: Mon May 3 09:24:05 2010 +0300 first commit**

* **git show**

**Shows information about a git object. Example: git show commit 84f241e8a0d768fb37ff7ad40e294b61a99a0abe Author: User <user@domain.com> Date: Mon May 3 09:24:05 2010 +0300 first commit diff --git a/README b/README new file mode 100644 index 0000000..e69de29**

* **git ls-tree**

**Shows a tree object, including the mode and the name of each item and the SHA-1 value of the blob or the tree that it points to. Example: git ls-tree master^{tree} 100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 README**

* **git cat-file**

**Used to view the type of an object through the SHA-1 value. Example: git cat-file -t e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 blob**

* **git grep**

**Lets you search through your trees of content for words and phrases. Example: git grep "www.siteground.com" -- \*.php**

* **git diff**

**Generates patch files or statistics of differences between paths or files in your git repository, or your index or your working directory. Example: git diff**

**Merge-tool**

http://twobitlabs.com/2011/08/install-diffmerge-git-mac-os-x/3