**Git och Github**

**Vad är GIT?:**

GIT är ett versionshanteringssystem.

* När man skapar program gör man hela tiden förbättringar.
* Ibland kraschar programmet när man har gjort en ändring, eller så blir det inte som man tänkt sig, Då kan man gå tillbaka till en tidigare version.
* Man kan också spara som vanligt i olika versioner (utan git) men måste då ge dom olika namn, kanske ha olika mappar. Det blir lätt jobbigt och rörigt.
* Man kan också använda det för att testa olika saker utan att riskera det som fungerar.
* Man kan använda det för att jämföra kod.
  1. GIT kan användas i större team. Då kan man jämföra ändringar för att se vems som är bäst. Vid en krasch kan man se vad man ändrade på.
  2. Flera i ett team kan jobba på samma kod men med en egen version.
* Är även ett sätt att ha backup på ditt arbete.
* Kan användas för att visa kod som ett slags CV på gitHub.

Versionshanteringssystem finns i två varianter:

* Centraliserade
* Decentraliserade/distribuerade

I Centraliserade system används en server där källkoden sparas centralt.

* De flesta operationer kräver en nätverkskoppling.
* Det finns ett centralt ställe där alla kommer åt koden.

I distribuerade system är de flesta operationer lokala och kräver ingen server.

* Git är ett distribuerat (decentraliserat) system.
* Genom t.ex. gitHub kan man göra det centraliserat.

**Lite nyckelkoncept:**

* **Repository** – En behållare eller mapp som innehåller projektets filer. Även historiken för filerna. Vi kallar det repo i fortsättningen.
* **Versions Control** – Det huvudsakliga syftet med git. När du har t.ex. en wordfil så skriver du antingen över filen varje gång du sparar med en ny fil, eller så sparar du olika versioner under olika namn som fil1, fil2 o.s.v. Med git slipper du detta.
* **Commit** - Att "commit:a" sina ändringar talar om att man nu har uppdaterat sina filer och sparar dessa i en ny version, en commit. Detta görs i lokalt på din dator. Är ett slags ”foto” av hur filerna såg ut just då.
  + *Ex: git commit -m “Message here.” –m betyder att det som följer är ett meddelande. Försök göra tydliga meddelande så man vet vilka ändringar du gjort.*
* **Push** - Innebär att man tar de "commitade" filerna och skickar/pushar upp på en central server (eller något annat ställe). I Vårt fall kommer detta oftast vara GitHubs server.
  + git push origin master (origin är gitHub och master är huvudbranch)
* **Pull** – Hämta filer från central server.
  + git pull origin master
* **Forking** - Detta innebär att man hittar ett intressant projekt som någon annan lagt ut på GitHub 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.
* **Merge** – När du gjort ändringar och vill föra över dessa till huvudprojektet.
* **Branch/Branching -** Du kan 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 då man enklare kan jobba utan att skriva över varandras filer.
  + När man är färdig men en del går man ihop med huvudgrenen (masterbranch) igen. Har någon annan jobbat på samma del får man jämföra filerna för att bestämma vilken man ska använda.
* **Working directory** – Filer som git inte “ser”.
* **Staging area, Index** – Filer som git ”ser” men som inte är commitade
* **HEAD** – pekare som pekar mot var du är i dina commits. Default är ditt senaste commit.

**Git- kommandon:**

* **git** **status** – Kollar statusen på ditt repo. Vilka filer som finns där, vilka som har commitats och vilken gren (branch) som du arbetar på just nu.
* **git log** – Se alla commits
* **git** **add** - lägger *inte* nya filer i ditt repo. Istället gör det Git uppmärksam på att det har kommit nya filer i ditt repo.
  + När en fil inte har gits uppmärksamhet kommer den inte med i en commit. Filen räknas då som *untracked.*
  + Git add gör att filen kommer med när du gör commit.
* **git merge** – lägger till dina ändringar till huvudspåret.
  + Ex: *git merge utbetalningar* – Tar ändringarna du gjort i grenen *utbetalningar* och lägger till *master*.
* **git push** – ”pushar” lokala ändringar till gitHub.
* **git pull** – hämtar de senaste filerna från gitHub
* **git reset HEAD <filnamn>** - tar bort filer som lagts till med add från ”staging area” så de åter är ”untracked” (röda). Ifall man ångrar sig och inte vill ha med det i sin commit.
* **git checkout - - <filnamn>** - återgå till senaste commit**.**
  + Ex: git checkout - - readme.txt.
  + Alltså två bindestreck direkt efter varandra.
* **git fetch origin <commit-id>** - hämta en speciell commit från github.
* **git branch <nyGren>** - Skapa en ny branch
* **git checkout master** – gå till ny gren, i detta fallet master.
* **git branch –a** – lista dina ”branches”
* **git branch –d <namn på branch>** - ta bort gren. (du kan inte vara på den gren du vill ta bort)
* **git checkout –b <nyGren>** - Skapar ny branch och går till den i ett steg.

**Icke git-kommandon**

* **pwd** – se var du är i filhierarkin.
* **mkdir** – skapa ny mapp.
* **cd <mappnamn>**– byt mapp.
* **cd** .. – gå upp ett steg.
* **ls** – lista filer/mappar
* **clear** – rensa fönstret

**GitHub**

GitHub ett centralt ställe där du kan spara olika versioner av dina filer online, istället för att bara ha dem lokalt.

Det används också för att visa sin kod för andra. Du kan visa andra (t.ex. en ev. arbetsgivare) vad du gjort. Du kan också se vad andra har gjort.

När du använder git så sparar du olika ögonblicksbilder av din kod lokalt, och kan gå tillbaka till äldre versioner om det behövs. Med gitHub kan du spara dina filer online också. På så vis kan andra team-medlemmar jobba med filerna. Du kan också komma åt filerna t.ex. på en dator på jobbet och en dator hemma.

**Installera git och github för första gången.**

**Installera GIT:**

Gå till <http://www.git-scm.com/>

Välj 64-bitars version.

Klicka dig vidare i fönsterna.

När du kommer till ”select components” så klicka i allt utom den sista.

Vid nästa fönster med val ”Adjusting your path environment”, välj mittersta valet.

Välj sista valet om du bara har Windows (Vilket jag antar att ni har)

I nästa val ska ni konfigurera terminalen. Välj sista alternativet om ni bara kör Windows.

Vid nästa val (Performance tweeks) så låt rutan vara okryssad.

Avsluta installationen.

**Förbered Github**

Gå till Github.com och skapa ett konto om du inte redan har ett.

Det är gratis att skapa ett ”grundkonto”. Klicka i övre högra hörnet för att gå till din profilsida.

Nu ska vi skapa ett nytt s.k. repository. Det är ett ställe där du bevarar din projektfiler.

**Nytt repository**

Nu ska vi först namne vårt repository. Ge det ett kort, beskrivande namn.

Man kan skriva in en beskrivning men det är valfritt.

Välj public och att skapa en readme-fil.

Du kommer då till huvudsidan för ditt repository.

**Skapa lokalt konto**

Nu har vi ett repo online men det är inte där vi ska jobba med det.

Starta Gitbash och skriv in **pwd**. Det är kommandot för att se var du är.

Skriv **mdkr** projects, det betyder **make directory** och skapar en ny mapp. **projects** är bara ett namn vi väljer. Du kan välja ett annat.

Byt sedan till din nya mapp genom att skriva **cd projects**. Cd betyder Change directory (byt till mapp).

Skriv **pwd** igen för att se att du är i din nyskapade mapp.

**Namn och email**

Git behöver också namn och email-adress. Anger vi det inte försöker Git lista ut det ändå.

För säkerhets skull anger vi det manuellt.

Skriv:

**Git config–global user.name ”Ditt namn”** : Detta registrerar ditt användarnamn.

**Git config–global user.email ”dittNamn@live.com”** : Detta registrerar din email.

**Git config–global --list** : Detta kollar att det blivit registrerat rätt.

**Klona ditt online repository till ditt lokala**

**Nu ska vi se till att vårt online repository också finns lokalt.**

Se till att du är i din nya mapp som du skapade.

Du ska nu kopiera IP-adressen som finns på din huvudsida i Github till ditt lokala GitHub. Antingen tryck på knappen ”Copy to clipboard” eller markera och kopiera adressen.

I ditt konsolfönster skriver du:

git clone + adressen (klistra in adressen)

Ex: git clone <https://gitbub.com/net777/git-hub-demo.git>

För att kolla att mappen kopierats så skriv:

**ls**

Detta visar innehållet i vår nuvarande mapp (projects)

Nu ska vi gå till vårt lokala repository, d.v.s. den mapp vi klonade från Github.

Här ska ni se samma mapp som ni hade på gitHub.

Byt till den mappen genom att skriva cd <mappnamn>

Skriv sedan **ls** igen för att se att filen/filerna från gitHub finns där.

Nu ska vi fråga git om statusen på vårt repo med:

***git status***

Vi bör få veta vilken ***branch*** vi är på. Det bör vara på master-bransch som är s.k. default. (eftersom vi inte har skapat någon annan).

Vi har nu också en koppling till gitHub.

**Uppmärksamma git på förändringar**

Nu har vi skapat ett online repository och ett lokalt. Nu ska vi testa att skapa en fil i vårt lokala gitkonto.

Skriv:

*touch Readme.txt.*

Trots det förvirrande namnet så skapar det här en tom fil i vår nya mapp. Gå till utforskaren och kolla.

Du ska nu kunna se din nyskapade fil. Men kan git se den?

Skriv:

*git status*

git bör då ange i röd färg att Readme.txt är en *untracked file*.

Det innebär att den inte inkluderas när du gör en *commit* (ögonblicksbild).

För att git ska ta med filen skriver man:

*git add Readme.txt* för att ta med just den filen. Eller så kan man skriva ”*git add .”*  (git add punkt) för alla filer i mappen.

Skriv git status igen för att se att filen har blivit registrerad.

git talar om vilka förändringar som kommer med i nästa *commit*, som är Readme.txt.

**Commita förändringar**

Vi kan lägga till så många filer vi med **git add.** Filerna hamnar då i ett slags mellanläge där git ”har koll” på filerna men de inte har ”checkats in” så att säga. Detta kallas för gits ”Staging area” eller ”index” (finns också andra namn).

Nu ska vi ta vår första ögonblicksbild. Vi ska *commita* vår förändring så vi kan gå tillbaka om något går fel t.ex.

Skriv:

*git commit -m “Add Readme.txt”* (-m talar om att det bara är ett textmeddelande som följer)

Om du skriver *git status* igen och det gick bra säger git:

***nothing to commit, working directory is clean***

Det innebär alltså att vi nu har skickat alla filer från **Staging area** till **commit.** Vi har alltså skapat en ögonblicksbild.

Det står också:

***Your branch is ahead of ‘origin/master’ by 1 commit***

Origin är där vi klonade ifrån, alltså vårt repo på gitHub. Git talar om att vi nu har en commit som inte finns på gitHub

**Pusha filer till gitHub**

För att de commitade filerna ska komma till gitHub måste vi använda kommandot ***push***.

Det är dock bra att kolla att vi har alla filer lokalt som vi har på gitHub.

Därför bör man först ”hämta ner” alla filer som vi har på gitHub i aktuellt repo.

Skriv:

***git pull origin master***

Sedan skriver du:

***git push origin master***

**Olika stadier för filer**

|  |  |  |  |
| --- | --- | --- | --- |
| Local | | | Remote |
| Working Dircectory  Här är filerna före de har lagts till med ”add” | **Staging area**  **Eller**  **Git index**  Här är filerna **efter** de har lagts till med ”add” | **Repository (.git-mapp)**  Här är filerna när de har commitats | Remote repository  Här är filerna efter push. |

**Övning**

* Skapa ett nytt repository på gitHub
* Klona ner det till din projektmapp lokalt.
* Spara en fil i din lokala mapp.
* Gör git uppmärksam på att filen finns så den kommer med i nästa commit.
* Gör en commit.
* Skicka commiten upp till gitHub och kolla att den kom dit. (Glöm inte köra en pull först)
* Gör en ändring i din lokala fil.
* Adda filen så git känner till den.
* Commita filen igen.
* Skicka den till gitHub igen.
* Kolla så båda commiten finns där.
* Kolla alla commits genom att skriva ***git log***.

**Branches**

Först ska vi skapa ett nytt repo som vi klonar ner.

Branches eller ”grenar” är ett sätt att skapa en egen utrymme där man kan jobba utan att förstöra huvudspåret vilket är master.

Gå till ditt lokala repository (t.ex. git-hub-demo) och skriv:

***git branch –a***

Då ska du kunna se alla ”grenar” du har, både den lokala och de på gitHub.

Den som har en asterix (\*) är den aktuella.

**Skapa ny branch**

Nu ska vi prova att skapa en ny branch.

Skriv:

***git branch myNewBranch***

Nu har vi skapat en ny branch. Skriv ***git branch –a*** igen. Nu bör du se den nya grenen.

Vi är dock inte i den branchen utan fortfarande på **master**, vilket syns på asterixen.

För att byta till den nya grenen skriver vi:

***git checkout myNewBranch***

Om allt stämmer säger gitbash att vi har bytt till den nya grenen.

Skriv ***git branch –a*** igen för att se om asterixen har hamna på vår nya gren.

**Ta bort branch**

Vi ska nu för övningens skull ta bort vår nya gren. Man ska dock inte såga på den gren vi sitter på. Det ska/kan man inte heller göra i git, så därför ska vi byta tillbaka till vår ”huvudgren”.

Byt nu tillbaka igen till ***master.*** Skriv:

***git checkout master***

Nu när vi är tillbaka i vår huvudgren kan vi ta bort vår nya ***branch***.

Skriv:

***git branch –d myNewBranch***

Kolla att grenen är borta med: ***git branch –a***

**Branching och merging**

Skapa en ny fil genom att skriva **touch test.txt**

En anledning till att man skapar en egen branch är att man kan jobba med en fil utan att riskera att förstöra det som redan finns i master.

Vi ska nu testa att skapa en ny branch och jobba med en fil i den.

Skapa en ny **branch** som du har lärt dig och gå till den.

Gå in i filen test.txt ändra något, spara sedan.

Kolla statusen på ditt repo.

Add:a din fil som vanligt och commita sedan.

Nu har vi commitat en ändring i en fil, men bara i vår nya branch.

Vi måste lägga till förändringen till vår huvudgren, d.v.s. master.

För att göra det måste vi först gå till master igen:

***git checkout master***

**Kolla vad som ändrats**

Innan vi lägger till den förändrade filen till master, ska vi först jämföra vad som har förändrats.

Skriv:

***git diff master newbranch*** – här är ju master vår huvudgren och newbranch vår nya gren (eller vad ni kallade den) Minus visar vad som tagit bort och plus vad som lagts till.

Nu är det dags att slå ihop våra **branches (merge)**. Vi skriver över de fil som finns i master med filen från vår nya branch.

Skriv:

***git merge newbranch*** – Vi slår ihop vår nya gren med master och skriver över förändringar.

**OBS! Detta kallas fast forward och fungerar bara om vi inte har commitat i master. Då kan det bli konflikter om det är skillnad på vår commit i master och commit i nya grenen. Det ska vi inte lära oss just nu. Har du gjort det börja om.**