

# DA336A Introduktion till Git och GitHub

## Introduktion

### Vad är versionshantering?

Ett versionshanteringssystem sparar/loggar alla ändringar du gör av en uppsättning filer så att du när som helst kan gå tillbaka till en tidigare version. Det här är naturligtvis oömbärligt för att rätta till katastrofiska misstag eller för att återskapa kod som raderats av misstag men det är också praktiskt i många andra sammanhang. Om du till exempel förvarar din (regelbundet uppdaterade) dokumentation i ett versionshanteringssystem kan du när som helst gå tillbaka och se hur designen av ditt system förändrats över tiden.

Versionshantering är speciellt användbart när du samarbetar med andra personer; det är lätt att kombinera allas bidrag och går alltid att se exakt vem som gjorde vilka ändringar.

Det finns många olika versionshanteringssystem men vi kommer att använda *Git*. Varför? Bortsett från att git är väldigt populärt och dess användning vitt utspridd så är det ett distribuerat versionshanteringssystem vilket betyder att varje användare som arbetar på projektet alltid har en komplett kopia av projektets historik lokalt på sin dator (till skillnad från historik som sparas centralt på en server). Detta gör att det går väldigt snabbt att arbeta med projektets hela historik även utan en uppkoppling till en central server.

### Nyckelord:

#### Repository

Ett git-repository är var alla data rörande ditt projekt och projektet självt är sparad. Detta kan vara ett lokalt repository på din dator där dina egna ändringar är sparade eller ett fjärr-repository som ligger på en annan dator (ofta delad med många andra användare) där dina ändringar kommer att hamna när du är redo att dela dem med andra.

#### Commit

När du har ändrat, lagt till eller raderat en fil i ditt projekt måste dessa ändringar *committas* till repositoryt för att registreras i projektets historik. En commit bör bestå av en grupp ändringar som är relaterade till varandra (men detta är inte ett krav och i praktiken inte alltid fallet).

#### Push

En *push* uppdaterar fjärr-repositoryt med de ändringar du gjort lokalt så att andra som jobbar på samma projekt kan se dina ändringar.

## Pull

En *pull* hämtar ändringar från fjärr-repositoryt och *merge* dem med dina lokala uppsättning filer.

## Clone

Ett projekt som finns publicerat på t.ex. GitHub kan *klonas* till en lokal dator. Då skapas en kopia av alla filer i projektet samt en full historik lokalt. Projektet "vet själv" var det hämtades från och vad det heter så att det är lätt att senare publicera uppdateringar.

## GitHub

GitHub (git är en webb-baserad tjänst som 'hostar' git repositories. GitHub leverera även en grafisk klient för att underlätta att arbeta med git utan att skriva in git-kommandon manuellt. Det är denna klient vi kommer att använda idag.

## Komma igång

En not om denna guide:

Dessa instruktioner fokuserar på användning av Git tillsammans med GitHubs desktop-klient. Det finns många andra grafiska klienter för Git men de kommer inte att diskuteras här (samma principer gäller för alla klienter). Kommandotolkinstruktioner kommer också att ges, då i *kursiv*, men det kan vara bra att läsa på för att verkligen förstå dem. I slutet finns en kort övning.

## Skapa ett konto på GitHub och installera GitHub Desktop

Skapa ett konto på GitHub: <https://github.com>

- Välj *Free personal plan*

Installera GitHub Desktop (finns redan på datorerna här i högskolan) och under Options...

- Logga in med ditt GitHub konto och välj ett användarnamn och en mailadress att koppla till dina commits.

*Följande kommandotolksoperationer motsvarar samma process:*

*git config --global user.name "YOUR NAME"*

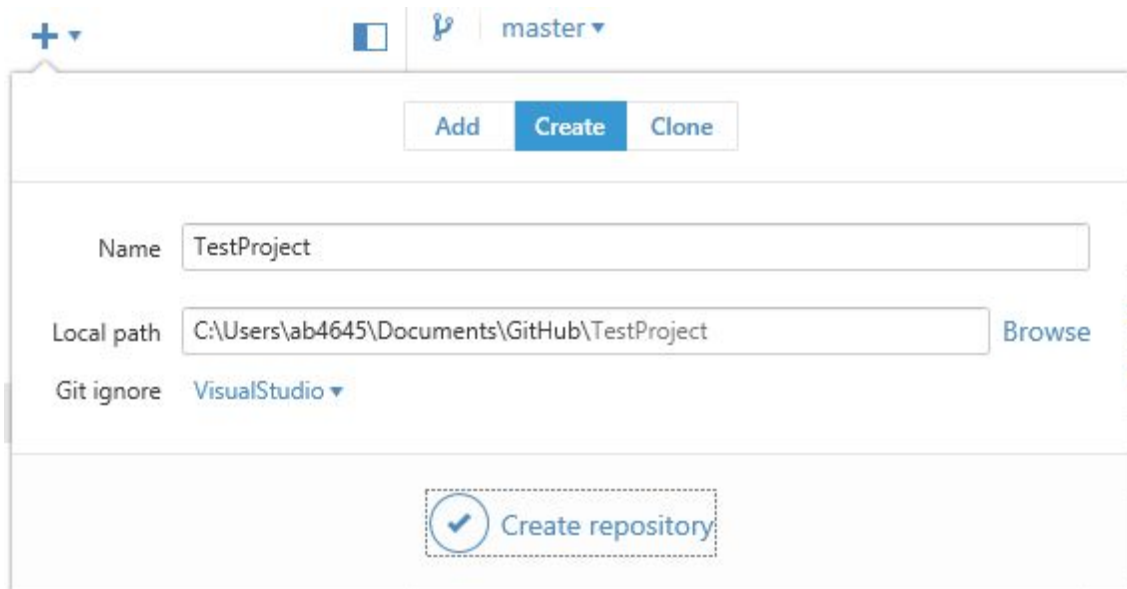
*git config --global user.email "YOUR EMAIL ADDRESS"*

- Välj ett *shell* (kommandotolk) att använda: PowerShell eller Git Bash, om du vill skriva git-kommandon direkt istället för (eller tillsammans med) att använda det grafiska användargränssnittet.

## Skapa ett repository

Ett repository kan skapas lokalt på din dator och senare publiceras på GitHub eller skapas direkt på GitHub (<https://github.com/new>) och sedan *klonas* till vilken dator som helst.

För att skapa ett repository i GitHub Desktop:



Klicka på “+” och välj *Create*. Välj ett namn, en sökväg och en lämplig .gitignore-fil (se nedan).

Ovanstående process är ekvivalent med följande kommandotolksoperationer:


`git init "TestProject"`

Efter detta bör en lämplig .gitignore-fil läggas till genom att den kopieras in i projektkatalogen och du skriver följande kommandon:

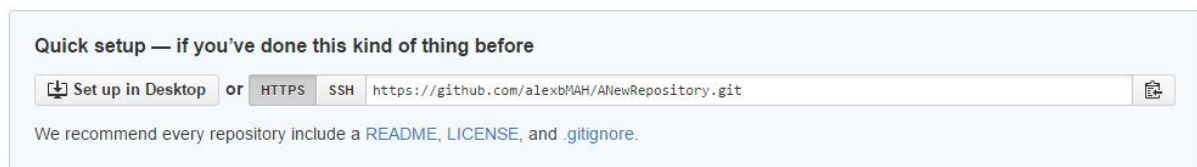
```
#Navigera till repositoryts katalog
cd TestProject
# Lägg till alla filer till "staging area" (i nuläget finns bara
.gitignore om du har lagt till en)
git add .
# Committa ändringarna och lägg till ett meddelande om vad som ändrats
git commit -m "Added .gitignore"
```

## .gitignore

.gitignore är en textfil som, när den läggs till i ett repository, definierar vilka filtyper och kataloger Git ska ignorera. Det är inte nödvändigt men starkt rekommenderat att använda en .gitignore-fil, i synnerhet när man jobbar med kompilerad kod eftersom detta kan generera många intermediära filer som inte är av intresse för någon annan (objektfiler och dylikt). Lyckligtvis finns det en samling av vanliga .gitignore-filer på GitHub (<https://github.com/github/gitignore>) som automatiskt kan läggas till i ett projekt när repositoryt skapas (om detta sker via GitHub Desktop eller GitHubs hemsida). Om du arbetar med Unity eller Visual Studio är en .gitignore-fil *väldigt rekommenderat*. Den valda .gitignore-filen kan sedan modifieras på precis samma sätt som alla andra filer i repositoryt.

När du är redo att publicera projektet på GitHub, klicka på . Detta skapar ett nytt repository för dig på ditt GitHub-konto, länkat till ditt lokala repository.

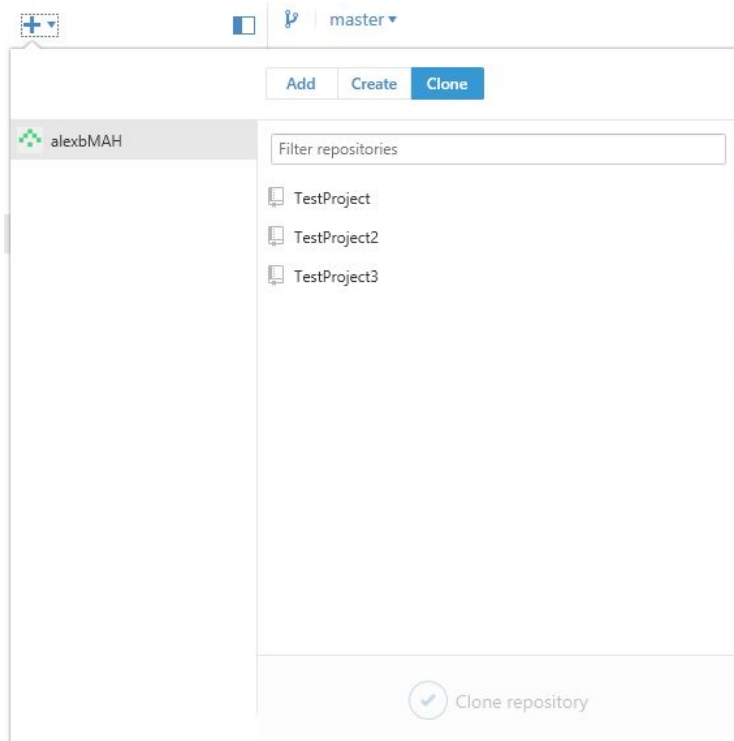
*Alternativt, om du redan skapat ett repository på GitHub, kan du pusha ditt lokala repository direkt till GitHub med hjälp av HTTPS-adressen du fick när du skapade repositoryt (se bild nedan) och kommandona:*



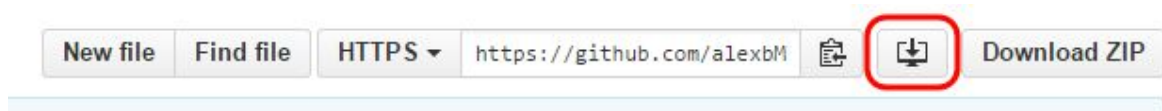
```
git remote add origin https://github.com/<username>/<repository>.git  
git push -u origin master
```

## Att kлона ett repository

Som nämnt innebär kloning att det skapas en lokal kopia av ett repository på din dator. GitHub-repositories som du äger eller jobbar på tillsammans med andra (see "Lägg till användare" nedan) kan klonas från GitHub Desktop:



Alla publika repositories på GitHub kan klonas till din dator från GitHubs hemsida genom ett klick på ikonerna som är inringad på bilden nedan:



Om du hellre vill kлона ett repo via kommandotolken kan du använda följande kommando med aktuell URL för repositoryt och en lämplig katalog att spara din kopia i:

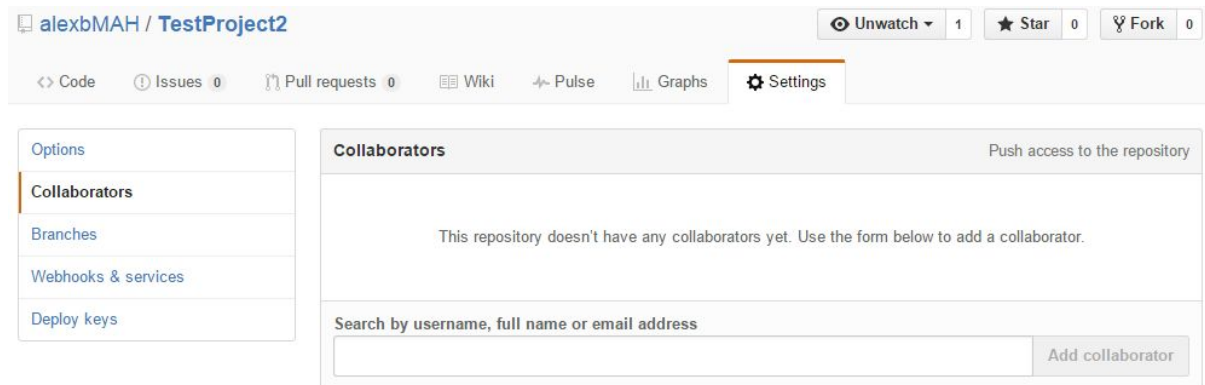
```
git clone https://github.com/<username>/<repository>.git MyDirectory
```

Notera att detta inte är unikt för GitHub, du kan kлона projekt från andra källor med motsvarande kommando.

## Lägg till användare

Andra kan se och kлона dina repositories men per default kan de inte pusha ändringar till ett repository som du äger. Du kan ge andra användare möjlighet att pusha till ett av dina repositories under *Settings*-fliken och sedan *Collaborators* för ett repository på github.com (se nedan). Lägg till alla medlemmar i din grupp!

(Not: det finns alternativa arbetssätt som inte kräver att en användare har *collaborator*-status, såsom <https://guides.github.com/introduction/flow/index.html>, men det förra tillvägagångssättet är bäst för den här kursen.)



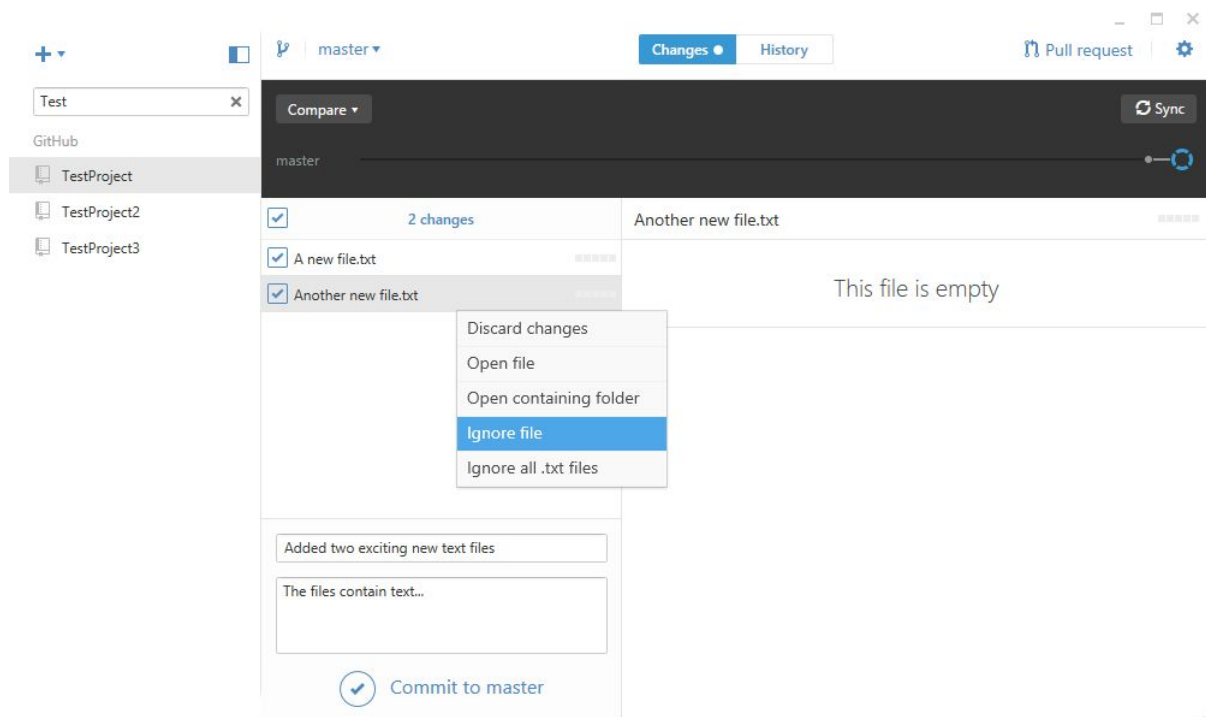
## Arbetsflöde

När du har satt upp ett repository och börjat jobba på ditt projekt är det dags att börja logga ändringar i filerna som ingår i projektet och att pusha till GitHub så att de andra i din grupp kan se ditt arbete.

## Comitta ändringar:

När man arbetar med Git kan filer vara i ett av tre tillstånd: **committed**, **staged** och **modified**. Committed filer har inte ändrats sedan du senast committade till ditt lokala repository. Staged filer är filer som ändrats och som köats för att läggas till i nästa commit. Modified filer är filer som ändrats men som inte köats för commit.

För att en ny fil ska börja loggas av git måste den läggas till i repositoryt. Git märker automatiskt när en ny fil lagts till i projektkatalogen och nya filer dyker upp under *Changes* i GitHub-klienten. För att lägga till filerna så att de blir en del av repositoryt, välj de filer som ska läggas till (detta ändrar status till *staged*), skriv ett commitmeddelande och klicka sedan på "Commit to master". Filernas status ändras nu till *committed* och kommer fortsättningsvis att loggas av git. Om du inte vill lägga till en ny fil i repositoryt kan du högerklicka på filen och välja "ignore file". Detta lägger till filen i *.gitignore* och den kommer sedan aldrig att dyka upp i *Changes* igen. Filerna måste committas igen varje gång de ändras så ändringarna hamnar i repositoryt.



För att göra det här från kommandotolken:

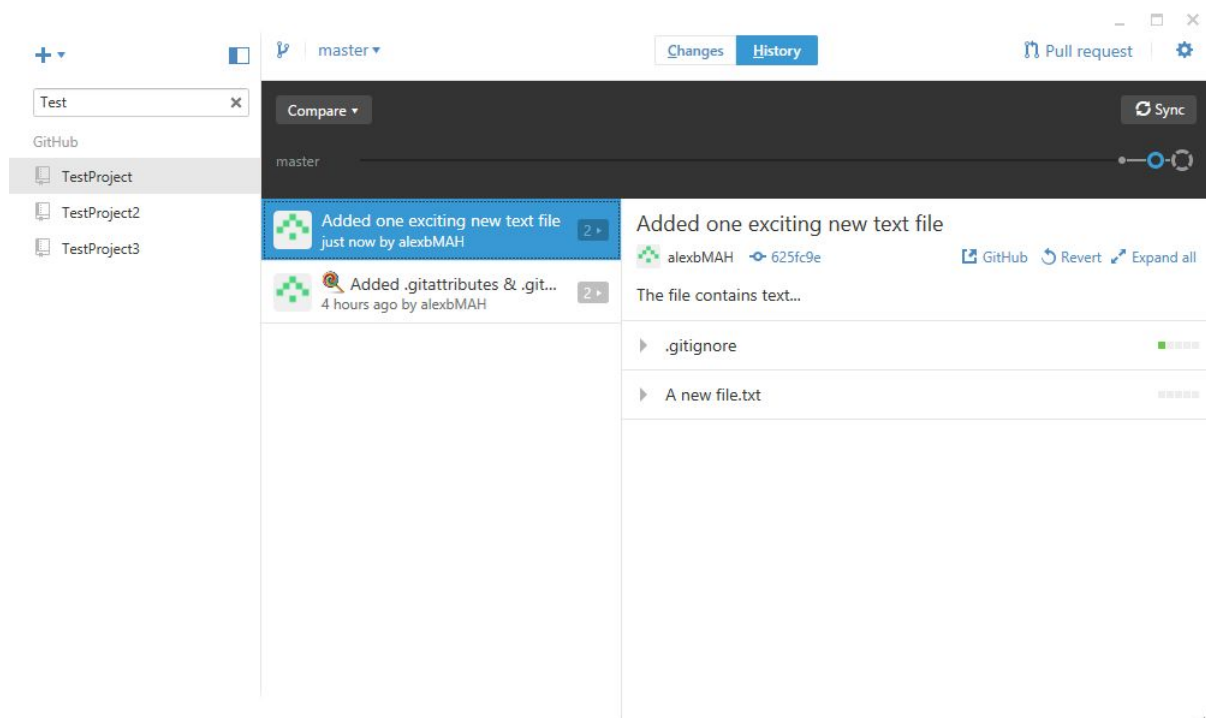
```
# lägg till en ny eller ändrad fil för commit  
git add filename.ext
```

```
# committa och skriv ett commit-meddelande  
git commit -m "A well-formed commit message."
```

Notera: om du inte lagt till några nya filer utan bara ändrat i filer som redan fanns i repositoryt kan du kombinera ovanstående kommandon:

```
git commit -am "A well-formed commit message."
```

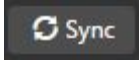
Efter att du committat dyker committen upp i *History*-tabben tillsammans med alla andra commits:



Samma historik kan visas med följande kommando:

`git log`

## Pusha till ett fjärr-repository

Efter en lokal commit är det dags att pusha ändringarna till GitHub så att alla som arbetar med projektet kan se vad som hänt. I GitHub Desktop görs detta genom ett klick på -knappen.

När du klickar på *Sync* sker i praktiken en kombination av två kommandon: **git pull** och **git push**.

- `git pull` hämtar hem ändringar från fjärr-repositoryt som andra har pushat under tiden sedan du senast synkade och mergear in dem i dina lokala filer (detta kan leda till konflikter vilket diskuteras nedan).
- `git push` uppdaterar fjärr-repositoryt med de ändringar du comittat lokalt-

Dessa två kommandon används tillsammans eftersom det är nödvändigt att hämta hem äldre uppdateringar innan du kan pusha dina egna commits (annars kommer din push inte att gå igenom).

**Viktigt:** du kan inte se andra personers ändringar och de kan inte se dina innan du klickat på *Sync*.

För att pusha till GitHub från kommandotolken:

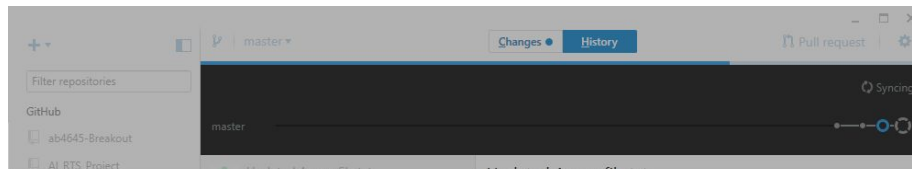
`git pull`

# här kan det vara nödvändigt att mergea konflikter manuellt

`git push`

## Mergea konflikter

Git kommer inte alltid att klara av att mergea filer automatiskt när du klickar på *Sync* (eller exekverar git pull) och kommer då att rapportera att en konflikt uppstått. När en konflikt föreligger måste du lösa den innan du kan synka igen.

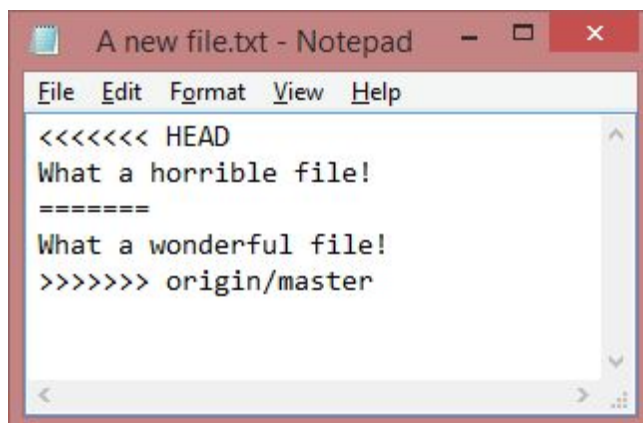


### Sync conflicts

Please resolve all conflicted files, commit, then try syncing again.



Vissa grafiska klienter för git inkluderar *diff tools* som hjälper dig att lösa en konflikt men detta är inte fallet för GitHub. Git lägger dock in markeringar i de filer som innehåller konflikter för att göra det lättare att se vad problemet är. I exemplet nedan har en användare ändrat innehållet i en fil till "What a horrible file!" medan den version som finns på GitHub ändrats till "What a wonderful file!". Git vet inte vilket variant av den här filen vi vill ha och signalerar därför att den här filen innehåller en konflikt. Mellan "<<<<<< HEAD" och "=====" finns den text som vi hade i vår lokala version av filen och mellan "=====" och ">>>>>> origin/master" finns den text som finns i GitHubs version av filen.



Välj den version av filen som du vill behålla, radera Gits konflikt-notation (HEAD etc) och committa filen igen så är konflikten löst.



## Övning:

Scenario för grupper om två personer (på två olika datorer): I den här övningen kommer du och en partner (A1 och A2) skriva en (vieldigt kort) roman tillsammans.

Börja med att skapa konton på GitHub. Om du arbetar på din egen dator måste du också ladda ner och installera GitHubs klient, om du jobbar på en MAH-dator finns GitHub-klienten redan tillgänglig.

**A1:** Öppna GitHub Desktop. Logga in på ditt GitHub-konto (klicka på kugghjulsikonen och välj *Options*) och under "Configure git", ge ett namn och en mailadress som kommer att länkas till dina commits. Här kan du också välja ett skal (en kommandotolk) att använda om du inte vill använda en grafisk klient i ditt arbete med git.

Skapa ett repository kallat "fantasy-novel" nångstans på datorn och sätt Git ignore till *None* eftersom vi inte kommer att ha några filer som inte bör loggas. GitHub kommer att skapa repot åt dig och committa en fil kallad .gitattributes. I nuläget finns repositoryt bara lokalt på din dator och inte på GitHub.

Innan det är dags att publicera repot ska vi lägga till det första kapitlet. Ladda ner chapter1.txt från itslearning och kopiera in det i fantasy-novel-katalogen. Filen bör nu automatiskt dyka upp i *Changes* i GitHub-klienten. Skriv ett meddelande i "Summary"-rutan och committa. Kom ihåg att commitmeddelandet ska beskriva vad du gjort men det bör vara kort och koncist. En längre beskrivning kan lämnas i "Description"-rutan om det behövs.

|   | COMMENT                            | DATE         |
|---|------------------------------------|--------------|
| ○ | CREATED MAIN LOOP & TIMING CONTROL | 14 HOURS AGO |
| ○ | ENABLED CONFIG FILE PARSING        | 9 HOURS AGO  |
| ○ | MISC BUGFIXES                      | 5 HOURS AGO  |
| ○ | CODE ADDITIONS/EDITS               | 4 HOURS AGO  |
| ○ | MORE CODE                          | 4 HOURS AGO  |
| ○ | HERE HAVE CODE                     | 4 HOURS AGO  |
| ○ | AAAAAAA                            | 3 HOURS AGO  |
| ○ | ADKFJSLKDFJSDKLFJ                  | 3 HOURS AGO  |
| ○ | MY HANDS ARE TYPING WORDS          | 2 HOURS AGO  |
| ○ | HAAAAAAAAAANDS                     | 2 HOURS AGO  |

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Publicera repositoryt på GitHub.

Lägg till A2 som *Collaborator* i repot så att ni kan jobba tillsammans. A2 bör nu få ett mail från GitHub med en länk till ditt repository.

**A2:** Klicka på länken i mailet från GitHub och klon repositoryt till din dator ("Save fantasy-novel to your computer and use it in GitHub Desktop.")

Lägg till en ny fil i repositoryt kallad Chapter2.txt med ett kort stycke text. Committa dina ändringar men synka inte.

Ändra "The End" i Chapter1.txt till "To be continued." eftersom kapitel ett nu inte längre är slutet på romanen. Committa dina ändringar men synka inte.

**A1:** Lägg till några rader i Chapter1.txt (ta inte bort något) och committa dina ändringar. Synka.

**A2:** Synka. Synk bör fungera utan problem (men om ni får en konflikt, försök att lösa den enligt texten ovan), Git kan mergea A1s ändringar utan vår inblandning.

## Nu kommer vi att introducera och lösa en konflikt:

**A1:** Ändra raden "The hero is victorious!" i Chapter1.txt till "The hero is fabulously handsome!". Committa och synka.

**A2:** Ändra raden "The hero is victorious!" i Chapter1.txt till "The anti-hero is a scurrilous swine!". Committa och synka. Du bör nu få ett meddelande om "Sync conflicts". Om du öppnar Chapter1.txt ser du att filen innehåller konfliktnotation som Git lagt till, det som fanns i din version av filen och det som fanns i Gits version av filen. Välj den version som du föredrar, eller kombination av båda alternativen, radera Gits konfliktnotation och spara filen. Committa och synka och allt ska nu fungera som vanligt igen.

## Nu ska vi se hur man ångrar en commit.

**A1:** Kapitel två är för dåligt så ta bort filen Chapter2.txt, committa och synka.

**A2:** Det kanske hade varit bättre att behålla Chapter2.txt och bara ändra texten? Synka ditt repo, klicka på den commit som tog bort Chapter2.txt och klicka på "Revert"-knappen. Detta skapar en ny commit som ångrar allting som gjordes i den commit som var vald (i det här fallet kommer Chapter2.txt att komma tillbaka). Synka.

## Tips

- Git kan göra mycket mer än det som beskrivs i detta dokument. Mer information kan hittas på (bland annat):

<http://git-scm.com/documentation>

<https://www.youtube.com/playlist?list=PLg7s6cbtAD15G8INyoeYDuKZSKyJrgwB->

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

- Committa ofta.
- Pusha (synka) inte kod som inte kompilerar (åtminstone inte till master-branch).
- Om du bara lär dig ett git-kommando bör det vara "git status". Det ger en beskrivning av repositoryts läge vilken sedan kan användas för att googla fram en lösning till ett problem!