Git

Allt du behöver veta om Git!

- Kom igång
  - Vad och varför?
  - Installera Git på datorn
  - SSH-nycklar i Git
  - Vad är Repository, Branch och Commit?
- Commits och Repos
  - Starta ett remote repository
  - Starta projekt / lokalt repo
  - Lägg till ett remote repo till ditt projekt
  - Pusha en commit till ett remote repo
  - Hämta ändringar från ett remote repo
  - Skapa en commit
- Branching och träd
- Lathund
  - Bra sidor att lära sig Git via :)

Kom igång

Kom igång med dina äventyr med hjälp av Git!

Vad och varför?

Vad är versionhantering?

Versionhantering är en process som används för att hantera och spåra ändringar i källkod och andra filer över tiden. Det är ett verktyg som används främst inom mjukvaruutveckling men kan även vara till nytta för att hantera ändringar i andra typer av dokument, som textdokument eller konfigurationsfiler.

Versionhanteringssystem (VCS) gör det möjligt för utvecklare att spåra och hantera ändringar i sina projekt över tiden. Det ger också möjlighet att återställa tidigare versioner av filer och samarbeta med andra utvecklare på ett organiserat sätt.

Centraliserade versionhanteringssystem (CVCS):

I ett CVCS finns det en central server som lagrar alla filer och deras historik. Utvecklare checkar ut filer från servern för att göra ändringar och checkar sedan in dem igen för att spara ändringarna till servern.

Distribuerade versiohanteringssytem (DVCS):

I ett DVCS har varje utvecklare en komplett kopia av hela projektets historik på sin lokala maskin. Detta gör att utvecklare kan arbeta offline och göra ändringar utan att vara anslutna till en central server. Ändringarna kan sedan synkroniseras och delas med andra utvecklare vid behov.

Vad är Git?

Git är ett DVCS som utvecklades av Linus Torvalds för att hantera den enorma källkoden till Linux. Git lämpar sig väldigt väl till *open source* utveckling eftersom utvecklarna inte är beroende av en central server. Git är utformat för att vara snabbt, effektivt och skalbart för både små och stora projekt. Det tillåter flera utvecklare att samarbeta och spåra ändringar i en källkodsbas eller vilken annan typ av filstruktur som helst.

Varför Git?

Som ett exempel:

Ni ska bygga en hemsida, du och dina kurskamrater, som en del av webutvecklingskursen. Ni börjar med att bygga varsin sida i HTML och CSS. Men sen när det kommer till att få sidorna att fungera med varandra blir det helt plötsligt **väldigt, väldigt jobbigt**. Du och din kompis har gjort

helt olika med era klasser och hur era knappar fungerar.

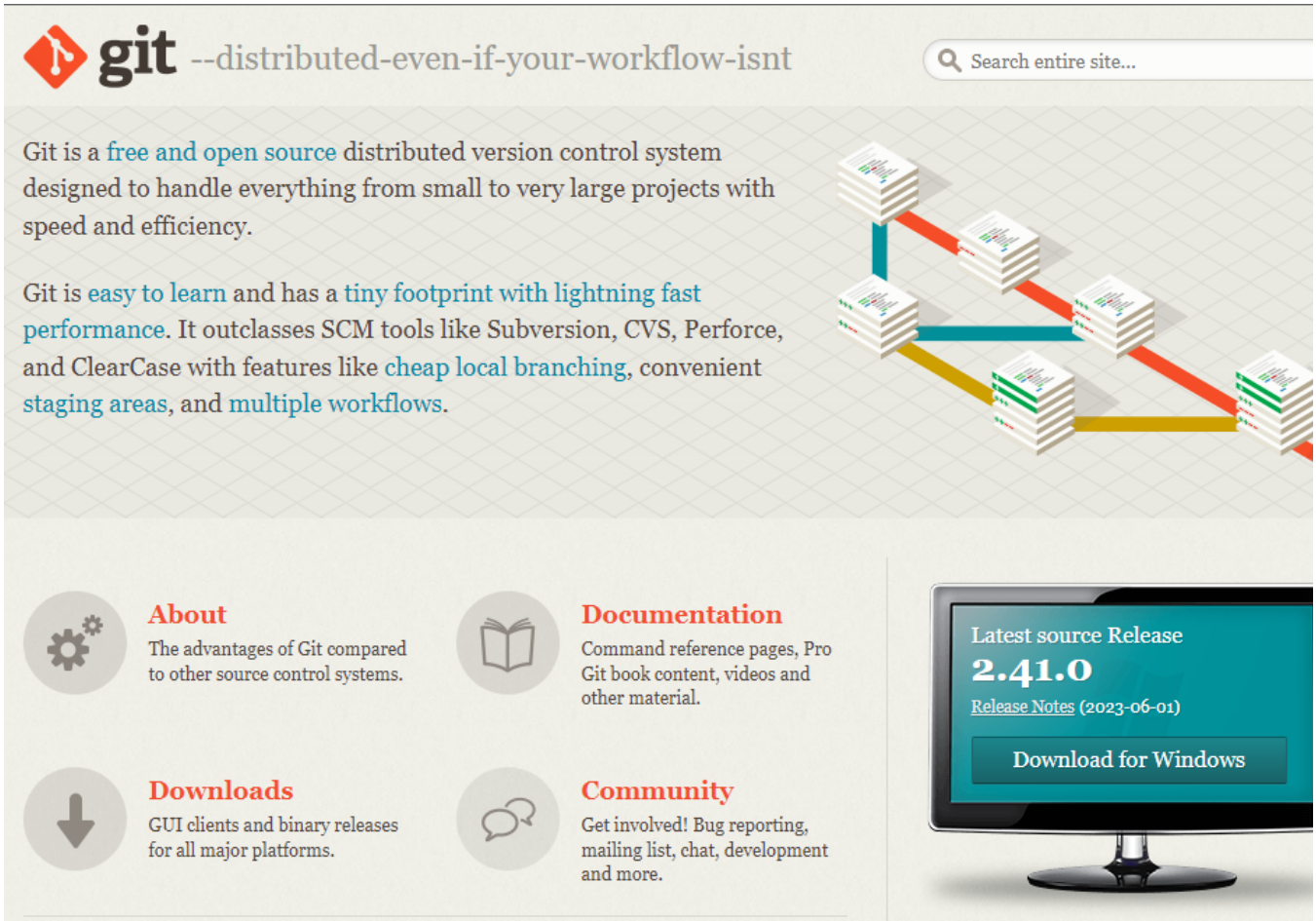
Git är menat att skapa en gemensam grund för alla inblandade. Förändringarna som dina klasskamrater gör behöver oftast du också. Om din kompis redan har gjort en knapp som är snygg kan du bara kopiera den, istället för att uppfinna hjulet igen.

Installera Git på datorn

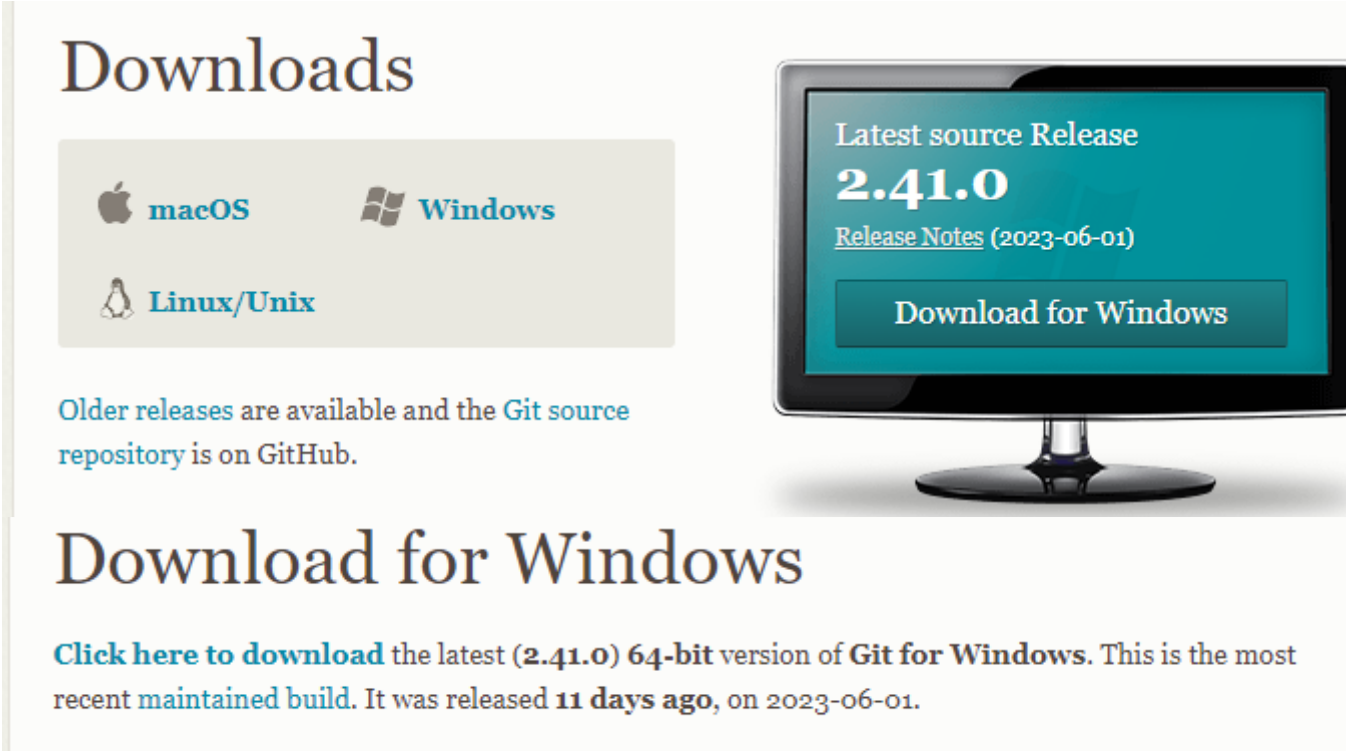
Windows

För att installera Git på en Windows-dator kan du följa dessa steg:

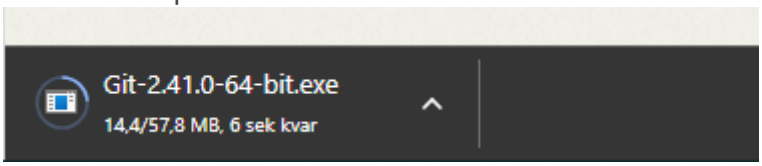
1. Besök Git:s officiella webbplats (https://git-scm.com) och klicka på "Download for Windows" (Ladda ner för Windows).



2. Detta länkar dig till nedladdningssidan. Klicka på den nedladdningsbara filen som matchar din Windows-version (32-bit eller 64-bit).



3. När du har laddat ner installationsfilen, dubbelklicka på den för att starta installationsprocessen.



4. Du kommer att bli ombedd att välja installationsalternativ. Här kan du välja att ändra installationsmappen eller använda standardinställningarna. För de flesta användare är standardinställningarna tillräckliga, så du kan fortsätta genom att klicka på "Next" (Nästa).

5. Välj "Components" (Komponenter). Här kan du välja vilka delar av Git som du vill installera. **För de flesta användare är standardinställningarna lämpliga.** Du kan fortsätta genom att klicka på "Next" (Nästa).

6. I nästa steg väljer du "Adjusting your PATH environment" (Justera din PATH-miljö). Här kan du välja om Git ska läggas till i systemets PATH-miljövariabel, vilket gör att du kan använda Git från kommandotolken. Det är vanligtvis en bra idé att välja alternativet **"Git from the command line and also from 3rd-party software" (Git från kommandotolken och även från tredjepartsprogram).** Klicka på "Next" (Nästa) för att fortsätta.

7. Välj "Use the OpenSSL library" (Använd OpenSSL-biblioteket). Här väljer du vilket SSL-bibliotek som Git ska använda för säkra anslutningar. Standardinställningen är att använda OpenSSL, vilket fungerar bra för de flesta användare. Klicka på "Next" (Nästa).

8. Välj "Line ending conversions" (Omvandling av radändelser). Här kan du välja hur Git ska hantera radändelser för textfiler. Standardinställningen är att låta Git automatiskt hantera det, vilket brukar fungera bra. Klicka på "Next" (Nästa).
9. Välj "Terminal emulator" (Terminalprogram). Här kan du välja vilket terminalprogram Git ska använda för att visa kommandotolken. Standardinställningen är "Use Git Bash only" (Använd bara Git Bash), vilket är en bra allroundterminal för Git. Klicka på "Next" (Nästa).

10. I det sista steget kan du välja att använda den interna Windows SSL-biblioteket. Detta är till valfri och standardinställningen fungerar bra för de flesta användare. Klicka på "Next" (Nästa).

11. Nu är du redo att installera Git. Klicka på "Install" (Installera) för att påbörja installationen.

12. Installationen tar några ögonblick och när den är klar kommer du att se ett installationsmeddelande. Klicka på "Next"

Debian & Ubuntu

1. `sudo apt-get install git`

SSH-nycklar i Git

Vad är SSH & en SSH-nyckel?

SSH står för Secure Shell och är ett nätverksprotokoll som gör det enklare att ansluta säkert till andra datorer, speciellt över ett osäkert nätverk. En SSH-nyckel är hur man krypterar och avkrypterar informationen, det gör datorn automatiskt. Med SSH-nycklar använder man oftast inte lösenord.

SSH-nycklar

En SSH-nyckel innehåller två delar, en privat och en publik del.

Den publika nyckeln (`.pub`)

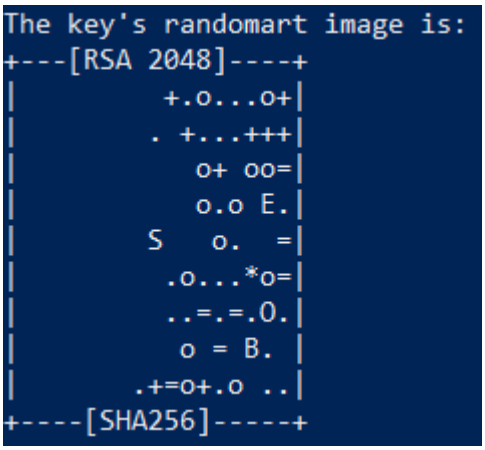
Den publika nyckeln är den delen av nycklen du delar ut till andra tjänster som ett "har du den här, kan du förstå mig"-kort. Den här nyckeln kan du fritt sprida till exempelvis Github eller vår egen GitLab för att autentisera dig.

Den privata nyckeln

> **Du ska aldrig dela med dig av den privata delen av din SSH-nyckel eftersom det är den delen av nyckeln som autentiserar att det verkligen är du som ansluter. Du kan identifiera den publika delen av din nycken som du kan dela med dig av för den har `.pub` i slutet.**

Skapa SSH-nyckelpar i Windows!

1. Öppna Powershell.
2. Skriv in `ssh-keygen` och tryck på enter.
3. Följ sedan instruktionerna tills du ser bilden nedan.



Nu finns nyckeln i `.ssh` i din användarmapp. (`C:\Users\[användarnamn]\.ssh`) i filen `id_rs.pub`

Skapa en SSH-nyckel i Linux!

> Det finns en mer detaljerad guide här som man kan följa, men den är på engelska.

Steg 1 - Skapa ett SSH-nyckelpar på den klienten du ska använda

I terminalen i Linux (eller ekvivalent WSL på Windows) skriver du in:

`$ ssh-keygen -t rsa`

Följ sedan instruktionerna som ges.

> Du behöver inte ange en passphrase såvida du inte vill ha ett extra lager av säkerhet ifall någon skulle komma över din privata SSH-nyckel. Men då skulle du behöva ange din passphrase varje gång du vill koppla dig via SSH med din nyckel.

När du har gått igenom processen kommer du i `~/.ssh` återfinna två filer (eller flera om du har flera nycklar) en `id_rsa.pub` och en `id_rsa` (eller vilket namn du nu valt). Det är innehållet i `id_rsa.pub` som du ska använda ifall du vill autentisera dig emot GitLab.

Använd SSH-nyckeln i Git

1. Gå in på git.ssis.nu och logga in. (Använd ditt Google-konton, elev@stockholmscience.se)
2. Klicka på “Edit profile”
3. Klicka på “SSH Keys”
4. Kopiera allt i filen `id_rsa.pub` som vi hittar i `.ssh` i din användarmapp. `C:\Users\ELEV-TAG\.ssh` eller i `~/.ssh` i Linux
5. Klicka sedan på “Add key".

6. Klistra in nyckeln i rutan och klicka på spara.

Nu har din nya nyckel registrerats på skolans GitLab och nu kan du klona och skicka upp commits till dina repositories via SSH.

Vad är Repository, Branch och Commit?

Repository

Ett repository är den huvudsakliga komponenten som spårar förändringar i ditt projekt. Den komponent hittar man under `./git` i roten av ditt projekt. Genom att titta på den här katalogen kan Git manipulera koden som du har valt att spåra. Tar du bort `./git` från ditt projekt förlorar du all spårning av ditt projekt i den instansen (om du har skickat förändringar till en server, finns de förändringarna som du skickat sparade där).

> Ett repo kan vara både lokalt på din maskin eller online på en server (remote) som https://git.ssis.nu/ och https://github.com/ du synkar dem genom push- och pull-kommandon

Att spåra förändringar

När man skapar ett nytt Git repository måste du manuellt lägga till de filer som du vill spåra. Det kan vara rimligt att inte spåra alla filer i ditt projekt. Om du använder en IDE till exempel finns det vissa filer som automatiskt sparas, som inte är relevanta till din kod. Dessa filer kanske inte är nödvändiga i ditt repository.

> Det vanligaste felet (som även professionella programmerare gör ibland) är att spåra förändringar av API-nycklar eller lösenord till databaser. Dessa får aldrig spåras, punkt.

För att förhindra att av misstag lägga upp filer som innehåller känsliga uppgifter kan du använda något som kallas `.gitignore`. Det är en fil som berättar för Git att du inte vill spåra vissa filer eller kataloger. Du skriver helt enkelt namnet (och sökväg om det behövs) på filen på var sin rad.

Exempelvis:

`.gitignore`

```
/node_modules
/public/hot
/public/storage
/public/js
/public/css
/public/mix-manifest.json
/storage/*.key
/vendor
.env
.env.backup
.phpunit.result.cache
docker-compose.override.yml
Homestead.json
Homestead.yaml
npm-debug.log
yarn-error.log
```

För att faktiskt spåra en förändring behöver du använda ett kommando (om du använder ett grafiskt gränssnitt är det här inte lika relevant) som kallas `git **add**` (alla kommandon i git måste prefixas såhär: `git **kommando**`).

```
du@dator:~/myFirstRepo$ git add minfil
```

Det innebär att git kommer börja hålla koll på den filen (om det är första gången). `add` lägger även till en fil till en s.k. commit och då är den *staged for commit.*

Commit

En commit är det huvudsakliga verktyget som du som programmerare segmenterar dina förändringar i mindre delar med. Det kan verka tradigt och lite poänglöst men det kommer ge utdelning i längden för dig och ditt framtida team (som ibland, är dig själv fyra månader i framtiden).

En commit ska göras varje gång du gör en förändring. Inte varje sekund, eller varje minut (det är oftast inte rimligt). Men när du som programmerare väsentligt förändrar hur din kod fungerar ska du commita det. Det innebär att när du är klar med något som fungerar, bör du commita det. Den förändringen vill du spara, för då kan du återgå och titta på förändringarna över tid om något går fel eller om det inte fungerar lika bra som förut.

Om du förändrat många olika saker bör du separera förändringarna till sina domäner. Gör du en förändring i hur din användare fungerar och sen i din databas, bör dessa två förändringar vara i två olika commits. Det är inte rekommenderat att förändra många olika saker (som inte påverkar

varandra) och lägga dem i samma commit.

Exempel:

Branch

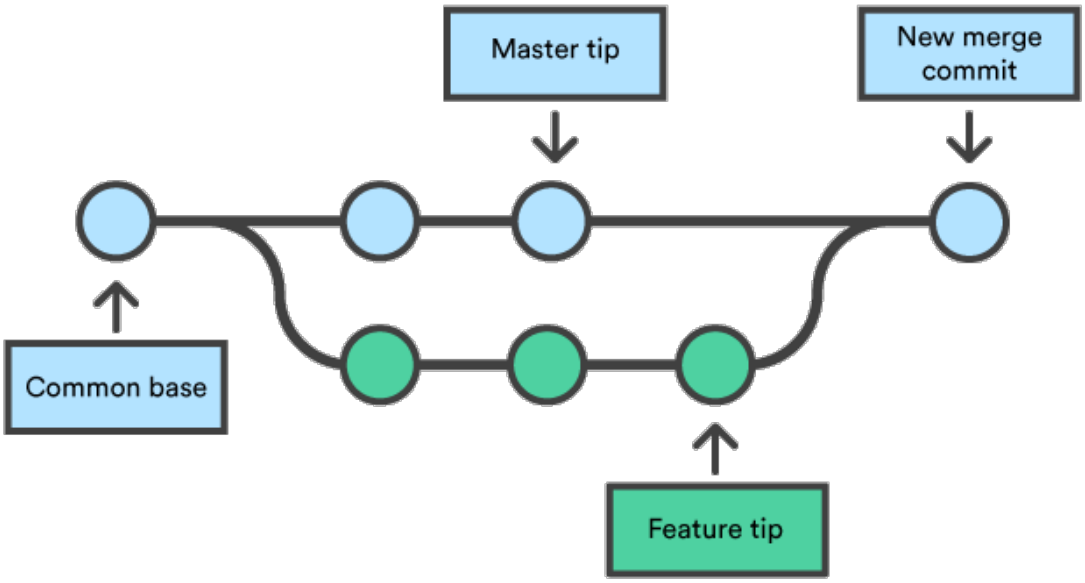
Git använder sig av något som kallas branches (eller grenar) för att spåra förändringar.

Huvudbranchen brukar kallas för `main` (eller `master` på aningen äldre system, läs mer här om varför). Det är det huvudsakliga projektet. Alla andra branches deriverar sin sanning från `main` vilket innebär att när du skapar en ny branch gör du det från main oftast.

Branching är främst till för att hålla main-branchen i ett fungerande stadie. Målet är att kunna klona ditt repo och veta att koden som laddas ner faktiskt fungerar.

Därför skapar du branches för din egen utveckling. Säg att du behöver lägga till en funktion (eller *feature*) till ditt program eller tjänst. Istället för att börja ändra i main-branchen skapar du en ny branch från main för att bevara mains integritet. På den här nya branchen kan du commita dina förändringar, utan att påverka main direkt. Därefter förenar du dem (*merge*) och låter main absorbera den nya branchen.

Det är lite som en tågräls där ett tåg fortsätter rakt fram och ett annat tåg tar av till ett annat spår för att få ny last och sen förenar sig med det tidigare tåget lite längre fram:



I mindre skolprojekt kan det vara så att branching inte är särskilt nödvändigt. Men i framtiden om du och dina skolkamrater ska arbeta tillsammans på samma projekt, är git ett otroligt bra sätt att kommunicera och åtgärda fel i kod som skrivs av många personer samtidigt.

Commits och Repos

Allt du behöver veta om Commits och Repos!

Starta ett remote repository

Skapa ditt första remote repo på GitLab (https://git.ssis.nu/)

1. Klicka på “Projects” och sedan på “Your projects”.
2. Där klickar du sedan på skapa nytt projekt. (Alltså ett blankt projekt)
3. Sätt sedan in ett namn på ditt projekt och välj “Internal” på “Visibility Level”.

Nu har du skapat ditt första “remote repository” och den kan nu kommas åt av personer som är inloggade på skolans GitLab.

> Processen för att skapa ett remote repository är ungefär samma på alla Git hemsidor

Starta projekt / lokalt repo

Skapa projektet genom att starta ett nytt med Git.

1. Gå in i din projekt mapp och öppna ditt val av konsol i den mappen. (Git Bash / Powershell / Terminal)
2. Skriv `git init` för att starta ditt projekt.

När du nu startat ditt projekt så kan du börja arbeta.

> Du kan även göra detta på ett projekt som du redan har börjat med!

Lägg till ett remote repo till ditt projekt

För att enkelt kunna dela koden med dina kamrater eller kollegor så behöver du använda ett remote repo.

1. Gå in i din projekt mapp och öppna ditt val av konsol i den mappen. (Git bash / Powershell / Terminal)
2. Skriv “git remote add origin” och stoppa in SSH/HTTPS länken till ditt remote repo och klicka på enter.
3. Skriv "git remote" och så borde du se "origin" komma upp.

> "origin" är det standardnamnet som används på huvud remoten för projektet. men kan självklart bytas ut till vad du själv vill!

> Det är rekommenderat att alltid använda SSH för Git då det oftast är smidigare än användarnamn och lösenord!
>
> Kolla hur du lägger in en SSH nyckel i följande KDBI: SSH Nyckel och dess användning i Git

> HTTPS länken för projekt på https://git.ssis.nu/ kräver ett Admin lösenord och det rekomenderas att använda SSH länken.
>
> Detta kräver då att du har registrerat en SSH nyckel på hemsidan. Se följande KDBI: SSH Nyckel och dess användning i Git

Pusha en commit till ett remote repo

Pusha dina commits genom att pusha dem till ett remote repo.

För att kunna pusha till ett remote repo så måste du ha lagt till den i projektet. Se följande KDBI:

Lägg till ett remote repo till ditt projekt

1. Gå in i din projekt mapp och öppna ditt val av konsol i den mappen. (Git bash / Powershell / Terminal)
2. Skriv "git push RName RBName" där RName är namnet på din remote och RBName är namnet på din remote branch.

> Om det är första gången du ska ladda upp något till ditt remote repo så måste du ha med argumentet "-u"
> Alltså ser det då ut som följande "git push -u RName RBName".

> Om du redan definierat vilken remote och vilken remote branch du vill pusha till så räcker det med att skriva "git push"

Hämta ändringar från ett remote repo

Hämta commits från ett remote repo.

För att kunna hämta commits från ett remote repo så måste du ha lagt till den i projektet. Se följande KDBI: Lägg till ett remote repo till ditt projekt

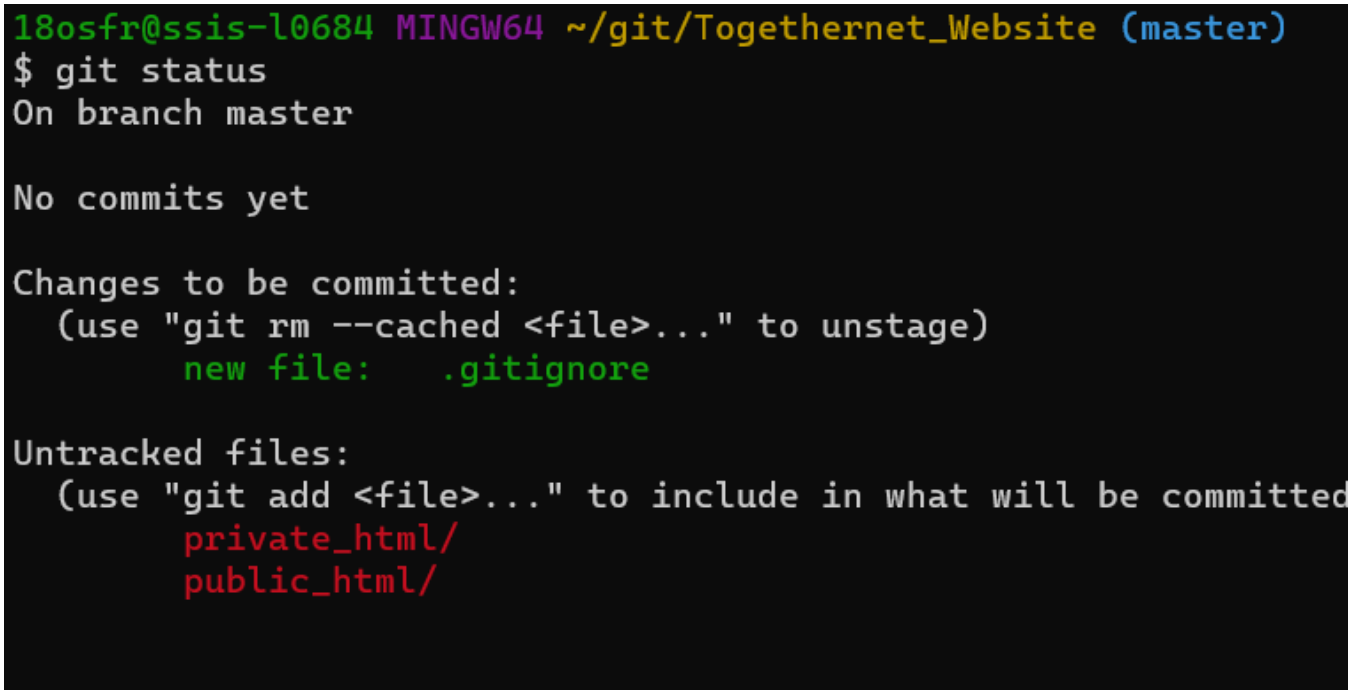
1. Gå in i din projekt mapp och öppna ditt val av konsol i den mappen. (Git bash / Powershell / Terminal)
2. Skriv "git pull" och då kommer den att tanka ner alla commits som är framför din nuvarande.

> Om du vill hämta men inte hoppa framåt så kan du göra "git fetch" och ladda ner commits utan att hoppa frammåt.

Skapa en commit

Skapa en commit med dina ändringar.

1. Gå in i din projekt mapp och öppna ditt val av konsol i den mappen. (Git bash / Powershell / Terminal)
2. Skriv `git status`. Då borde du få fram något som liknar bilden nedan:



3. Skriv `git add FILEName` där FILEName är den relativa vägen till filen.
   Detta kan du göra för varje fil eller använda `git add .` för att lägga till alla filer som är antingen untracked eller modified.
4. Skriv `git commit -m "TITLE"` där TITEL är titeln på din commit.
5. Skriv `git log` för att se dina commits. Följande bild är ett exempel på hur det kan se ut:

```
18osfr@ssis-l0684 MINGW64 ~/git/Sweeper_UnityEdition (master)
$ git log
commit 50da193e088fe801168978e38a68e0627f38eae7 (HEAD -> master, origin/master, origin/HEAD)
Author: Oscar Freij <18osfr@ad.ssis.nu>
Date:   Mon Mar 29 16:45:41 2021 +0200

    Added custom models and camera movement script.

    Also fixed problem where hover color overwrote the flag color on tiles

commit 3fab98d3801acef2c067380180f2e818b16a05bc
Author: Oscar Freij <otg020313@gmail.com>
Date:   Mon Mar 29 00:57:55 2021 +0200

    Added hover color, remade color change system and fixed error with bomb generation creating inf loo

commit 8e79ae07218bbbd6ff4774c7926723f7baad316a
Author: Oscar Freij <otg020313@gmail.com>
Date:   Sat Mar 27 19:11:20 2021 +0100

    First commit with code

commit 5f853b64f8b79f670db2e7ce9dda454d059b7c7e
Author: Oscar Freij <otg020313@gmail.com>
Date:   Sat Mar 27 19:09:05 2021 +0100

    Initial commit
```

Du kan stoppa in flera `-m` i din `git commit` för att lägga till flera rader.
Exempelvis: `git commit -m "TITLE" -m "MESSAGE"`

`git commit` tillåter ASCII så som `\n` `\r` och flera andra.

Branching och träd

Här finns inget ännu men det kommer snart :)

Lathund

För dig som vill ha en ett enkelt och snabbt sätt att hitta olika kommandon eller guider!

Lathund

Bra sidor att lära sig Git via :)

https://learngitbranching.js.org/

Denna sida har en interaktiv miljö där du kan testa dig fram och lära dig hur Git fungerar genom utmaningarna som du får av hemsidan.