KOMPENDIUM

Git på HiNT

En innføring i versjonskontroll med Git, SmartGit og Bitbucket på HiNT

Ståle A. Nygård

Høgskolen i Nord-Trøndelag
Kompendium

Steinkjer 2012

Git på HiNT

En innføring i versjonskontroll med Git, SmartGit og Bitbucket på HiNT

Ståle A. Nygård

Høgskolen i Nord-Trøndelag
Kompendium
ISBN 978-82-7456-666-8
Steinkjer 2012

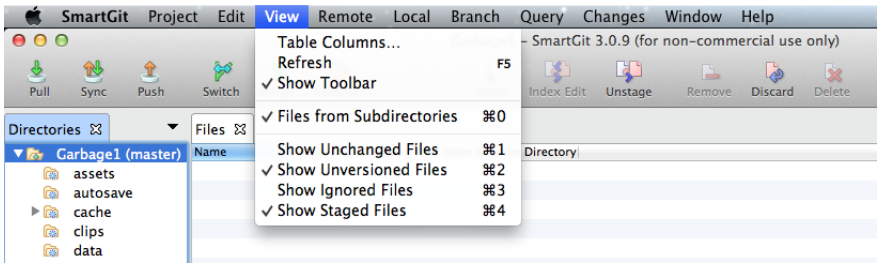
1 Innholdsfortegnelse

2 Innledning

Dette kompendiet er i laget som en innføring i bruk av versjonskontrollverktøyet Git på Høgskolen i Nord-Trøndelag (HiNT) sine datasaler på Steinkjer. Kompendiet er hovedsakelig myntet på studenter som går studiet Spill og opplevelsesteknologi og studiet Multimedieteknologi.

Det er en liten teoretisk intro til versjonskontroll og Git, men store deler av kompendiet viser praktisk bruk av verktøyene (hands-on). Det vil derfor være en fordel å prøve verktøyene samtidigs som man går igjennom eksemplene. Samtidig inneholder kompendiet mange ”skjermbilder” for å tydeliggjøre eksemplene, så det går også an å få en viss forståelse for bruk av Git ved å kun lese uten å prøve.

Det er valgt å bruke GUI-klienten ”SmartGit” og nettjenesten ”BitBucket” i eksemplene. Dette er verktøy og tjenester som kan endre utseende etter hvert som det kommer nye utgaver. Det er derfor ikke sikkert at alle skjermbilder stemmer helt overens med det som du opplever når du prøver eksemplene. Skjermbildene fra SmartGit er tatt med versjon 3.0.9 av dette programmet, og for å få mest mulig samme skjermbilde når du gjennomfører eksemplene kan du prøve å sette innstillingene som følger (View-menyen):



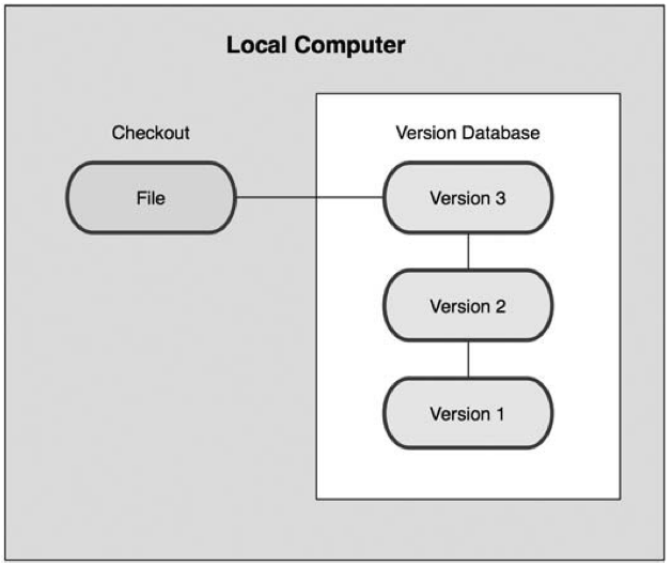
Figur 1: Visningsinnstillinger brukt i SmartGit

3 Intro til Git

3.1 Hva er versjonskontroll

Versjonskontroll er i vårt tilfelle et system (**VCS** – **V**ersion **C**ontrol **S**ystem) som tar vare på endringer som blir gjort på en fil eller et sett med filer over tid slik at det er mulig å gå tilbake til spesifikke versjoner senere. Grunnen til at man ønsker en slik mulighet kan for eksempel være at man ønsker å sammenligne tidligere versjoner av filer/prosjekt med siste versjon, eller at man av ulike årsaker har behov for å ”spole” et prosjekt tilbake til en tidligere (fungerende) tilstand.

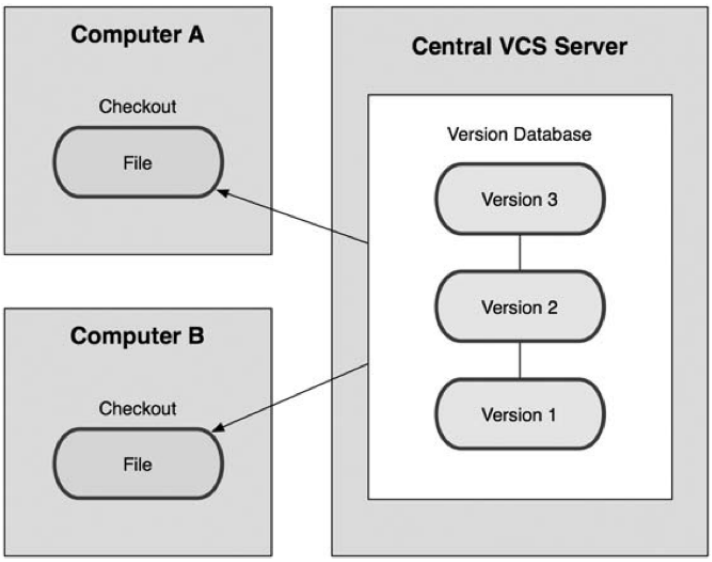
En typisk implementasjon av et VCS er at man har en ”versjonsdatabase” (**repository**) hvor man med jevne mellomrom lagrer endringene på filer som ”versjoner”. Man kan derfor ”sjekke ut” (checkout) hvilken som helst versjon av de versjonerte filene senere.



Figur 2: Local version control diagram (Chacon 2009:2)

3.2 Versjonskontroll og samarbeid

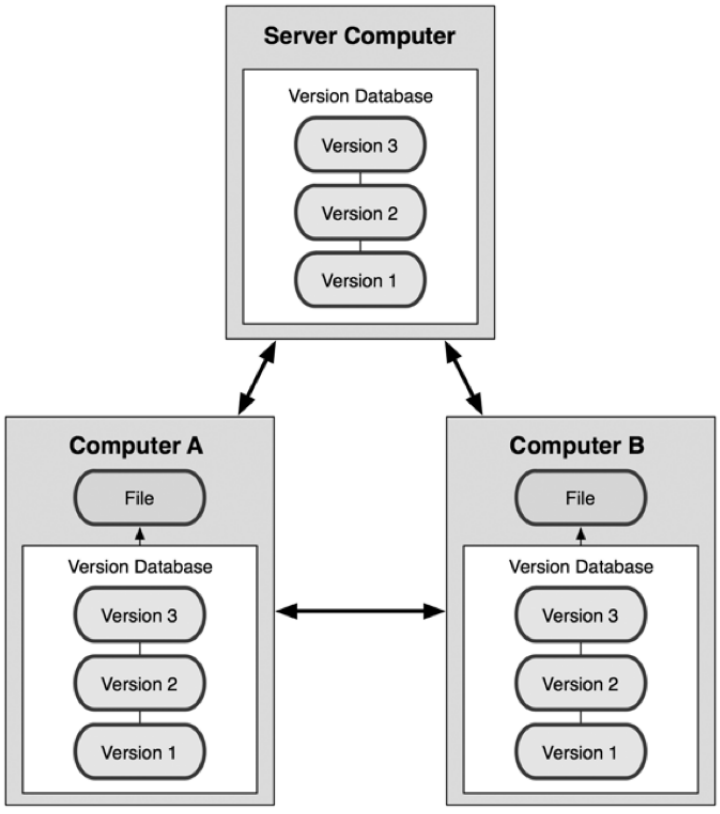
Noen VCS’er har også mekanismer som gjør det mulig for flere personer å samarbeide om å jobbe på et og samme prosjekt, på hver sin datamaskin, på en kontrollert måte. CVS og Subversion er begge eksempler på et såkalt **CVCS** – **C**entralized **V**ersion **C**ontrol **S**ystem, som støtter dette ved at versjonsdatabasen ligger på en sentral server. Hver bruker har sitt eget klient-program som sørger for å ”sjekke ut” og ”sjekke inn” filer fra/til serveren, og rydde opp i eventuelle konflikter.



Figur 3: Centralized version control diagram (Chocon 2009:3)

En ulempe med denne CVCS-modellen er at serveren representerer en ”single point of failure”; dvs. hvis serveren er ”nede” vil ingen kunne jobbe mot versjonsdatabasen, og hvis harddisken på serveren krasjer kan man risikere at hele historien går tapt.

Et såkalt **DVCS** (**D**istributed **V**ersion **C**ontrol **S**ystem) har en modell som sikrer versjonsdatabasen på en bedre måte. Når man sjekker ut et prosjekt fra et DVCS kopierer (kloner) man hele versjonsdatabasen. Man sjekker inn nye versjoner i den lokale versjonsdatabasen, og systemet har videre mekanismer for å reflektere dette videre til alle kopier (kloner) av prosjektet (som oftest; men ikke nødvendigvis, via en ”originalkopi” som ligger på en server). Git og Mercurial er eksempler på systemer som benytter denne modellen.

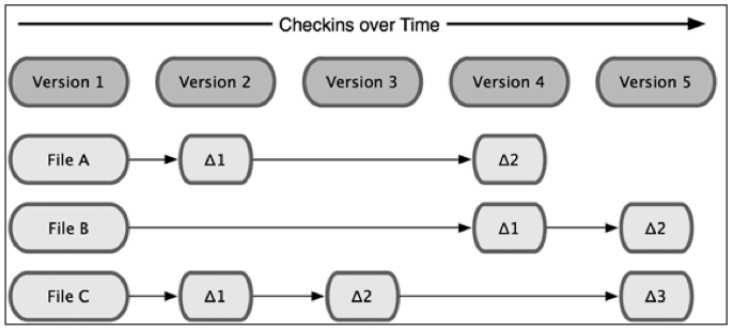


Figur 4: Distributed version control diagram (Chocon 2009:4)

3.3 Hva er Git

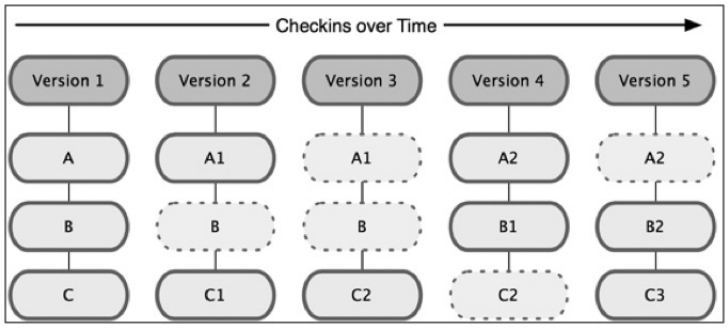
Git er et forholdsvis nytt distribuert versjonskontrollsystem (introdusert i 2005). Systemet er ”open source”, og opprinnelig utviklet til eget bruk av Linux-utviklingsmiljøet. Git ble laget med tanke på å være raskt og enkelt, men samtidig ha god støtte for å benyttes i ”ikke-lineære” utviklingsprosjekter. Siden 2005 har systemet modnet til å bli et av de mest brukte versjonskontrollsystemer. Populariteten skyldes ikke minst at den har en veldig kraftig ”branching”-funksjonalitet (utviklingsforløp i ulike ”forgreininger”).

Det som er litt spesielt med Git i forhold til andre versjonskontrollsystemer er måten de håndterer versjonsdata. Systemer som for eksempel Subversion, lagrer versjonsdata som en liste med forandringer som gjøres i filer.



Figur 5: Other systems tend to store data as changes to a base version of each file (Chocon 2009:6)

Git, derimot, lagrer ”snapshots” av hva slags tilstand alle filer er i hver gang man oppdaterer versjonsdatabasen.



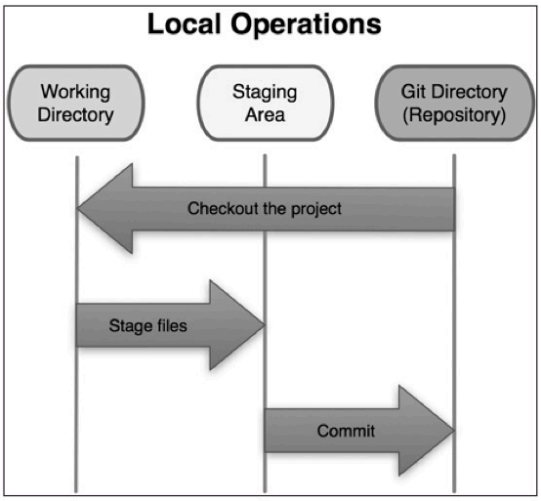
Figur 6: Git stores data as snapshots of the project over time (Chocon 2009:6)

3.4 Ord, begreper og mekanismer i Git

3.4.1 Grunnleggende arbeidsflyt

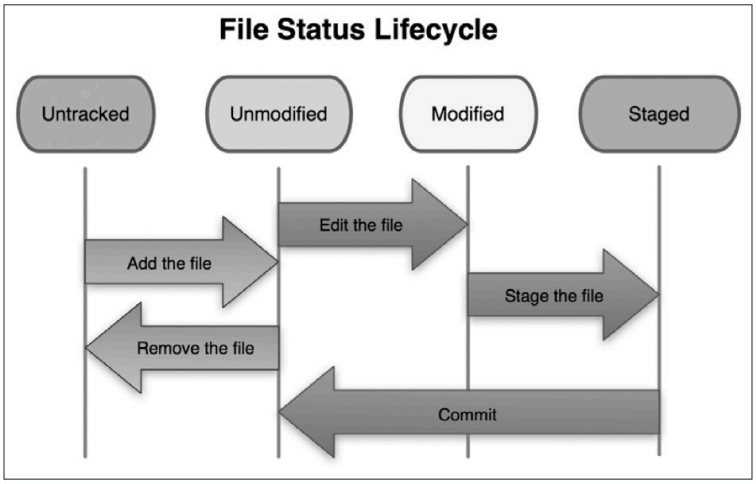
Den grunnleggende arbeidsflyten når du jobber i et prosjekt med versjonshåndtering i Git er typisk slik (punktene går som en runddans – se Figur 7):

1. Du modifiserer filer i din arbeidskopi av prosjektet (**working directory**/working tree)
2. Du legger til (**stage**) filer som er modifisert i en liste (**staging area**)
3. Du lagrer versjonsendringene (**commit**) i staged-listen til versjonsdatabasen (**Git directory**)



Figur 7: Working directory, staging area, and Git directory (Chocon 2009:8)

For at Git skal holde rede på hvilke filer som er modifisert må det først registreres hvilke filer som skal "spores" (tracked file). Nye filer som blir opprettet i arbeidskopien har derfor i utgangspunktet status **untracked**. Filer som har status **tracked** kan videre slik Git ser dem være i en av følgende tilstander: **unmodified**, **modified** eller **staged**.



Figur 8: The lifecycle of the status of your files (Chocon 2009:17)

De mest sentrale lokale operasjonene på arbeidskopien:

- **Stage** (add): forberede en fil (plasserer den i staging area) for å lagres i versjonsdatabasen med commit.
- **Unstage** (reset): fjerner en staged fil fra staging area (dvs. den vil ikke bli med på neste commit)
- **Ignore**: markerer for Git at en fil (eller filmønster) skal ignoreres i versjonshåndteringen; dvs. den vil ikke vises for oss som "untracked".
- **Commit**: lagrer en ny "snapshot" (versjon) av prosjektet på bakgrunn av hva som er "staged".

3.4.2 Arbeidsflyt samarbeid via server

Når man skal benytte Git som et verktøy for samarbeid, er det vanlig å ha en kopi av versjonsdatabasen på en server (se Figur 4). Den sentrale versjonsdatabasen kan gjøres tilgjengelig via vanlig fildeling i et lokalnettverk (smb://, afp://, file://), via webtjenerteknologi (https://), via secure shell (ssh://) eller via en proprietær Git-protokoll (git://). Her på HiNT er det pr. dags dato mest aktuelt å benytte fildeling på

lokalnettverk, men i dette kompendiet vil vi også vise hvordan man kan benytte den delvis gratis åpne webtjenesten ”Bitbucket” som støtter både https:// og ssh://.

Den grunnleggende arbeidsflyten når dere samarbeider via en server er som følger:

1. Du overfører en kopi (**clone**) av Git-prosjektet fra serveren til en egen datamaskin og lagrer den som egen arbeidskopi (inkludert alle filer og all versjonshistorikk).
2. Du gjør egne endringer og legger nye shapshots (commit) i lokal versjonsdatabase.
3. Du overfører dine endringer (**push**) til serveren.
4. Du overfører andre sine endringer (**pull**) fra serveren.

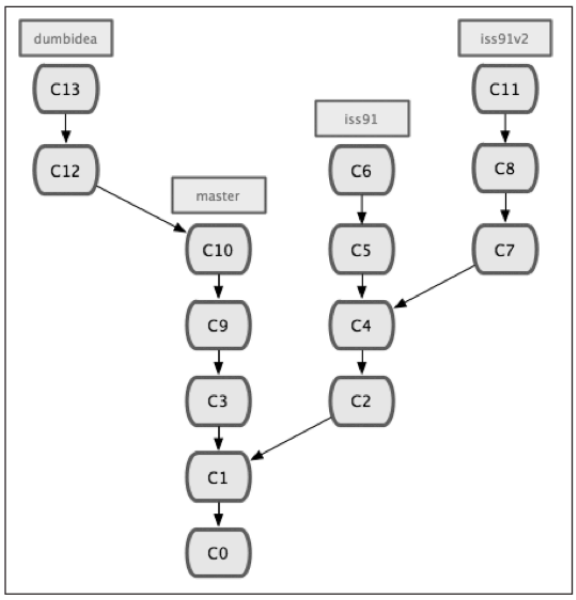
Hvis du allerede har en lokal kopi av Git-prosjektet er det ikke nødvendig å kopiere (clone) prosjektet fra serveren. For å sikre at du har siste versjon av prosjektet kan du da i stedet overføre eventuelt endringer som er gjort på server-versjonen i forhold til din lokale versjon (pull).

De mest sentrale operasjonene mot en server:

- **Clone**: kopiere/klone en annen versjonsdatabase (for eksempel, men ikke nødvendigvis, en versjonsdatabase som ligger på en server).
- **Push**: sende lokale ”commits” til en ”remote” versjonsdatabase (remote repository).
- **Pull**: hente ”commits” fra en ”remote” versjonsdatabase og ”integrerer” dem inn i den lokale versjonsdatabasen (kan innebære at enkelte versjonskonflikter må løses).
- **Synchronize**: dette innebærer i bunn og grunn at man utfører push og pull i en og samme operasjon.

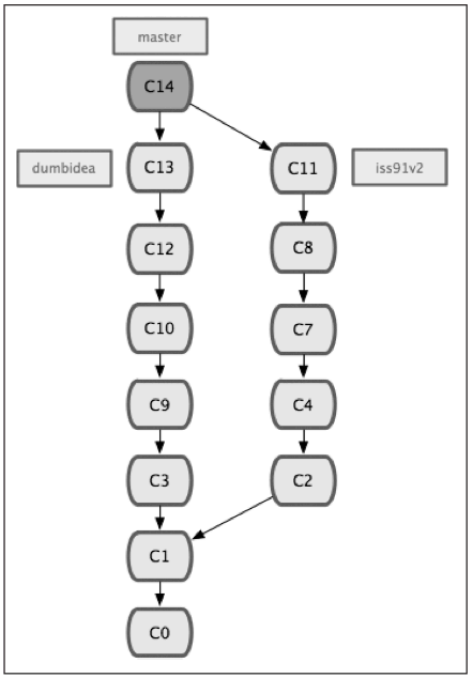
3.4.3 Branching (forgreinet utviklingsforløp)

”Branches” kan benyttes for å lagre en egen uavhengig serie/forgreining med commits/snapshots i en versjonsdatabase. Når man oppretter et nytt Git-prosjekt er man som standard inne i en ”hovedgrein” **master** (master branch tilsvarer ”trunk” i Subversion). Det er veldig enkelt å opprette en ny branch, og å bytte (switch/check out) mellom ulike branches (snapshots) på en sikker måte. Det er ikke noe i veien for å ha flere titalls samtidige branches i et Git-prosjekt. Begrepet HEAD refererer til hvilken commit/versjon som til en hver tid er aktiv i arbeidskopien (working tree).



Figur 9: Your commit history with multiple topic branches (Chocon 2009:63)

Dette gjør det for eksempel mulig å i en branch jobbe med bugfix på en publisert/ferdig versjon av et program, samtidig som det i en annen branch jobbes med utvikling av neste versjon. Det er også forholdsvis enkelt å ”flette sammen” (merge) det som er gjort i en branch inn i en annen branch.



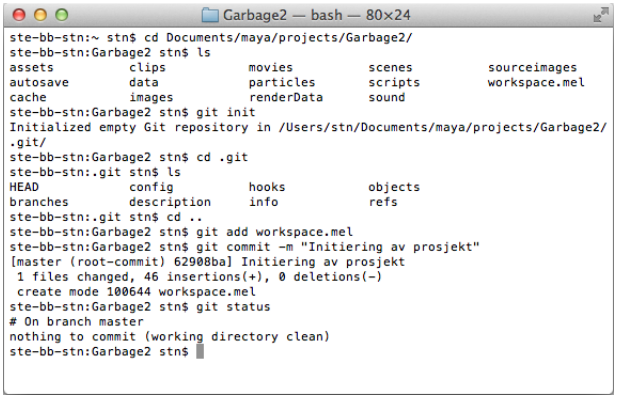
Figur 10: Your history after merging in dumbidea and iss91v2 (Chocon 2009:64)

De mest sentrale operasjonene i forbindelse med branching:

- **Switch**: bytte til en annen branch, dvs. arbeidskopien av prosjektet byttes ut til å inneholde alle filer og versjoner i aktuelle ”snapshot” av prosjektet.
- **Check out**: kan benyttes for å bytte til en spesifikk commit (”snapshot”) i hvilken som helst branch.
- **Merge**: flette inn forandringer i en branch inn i aktiv branch. Etter en merge kan det være at du må rydde opp i eventuelle konflikter, før du gjør en ”merge commit”.
- **Add branch**: oppretter en ny branch på bakgrunn av en gitt commit (snapshot).
- **Add tag**: sette inn en beskrivende tag på en gitt commit. Dette kan for eksempel dreie seg om tag’er for når en ny versjon av et produkt er ferdig (”v.1.0”, ”v.1.0.1”, ”v.1.1”, osv.). Bruk av tag’er kan gjøre det enklere å lese/navigere i versjonshistorikken.

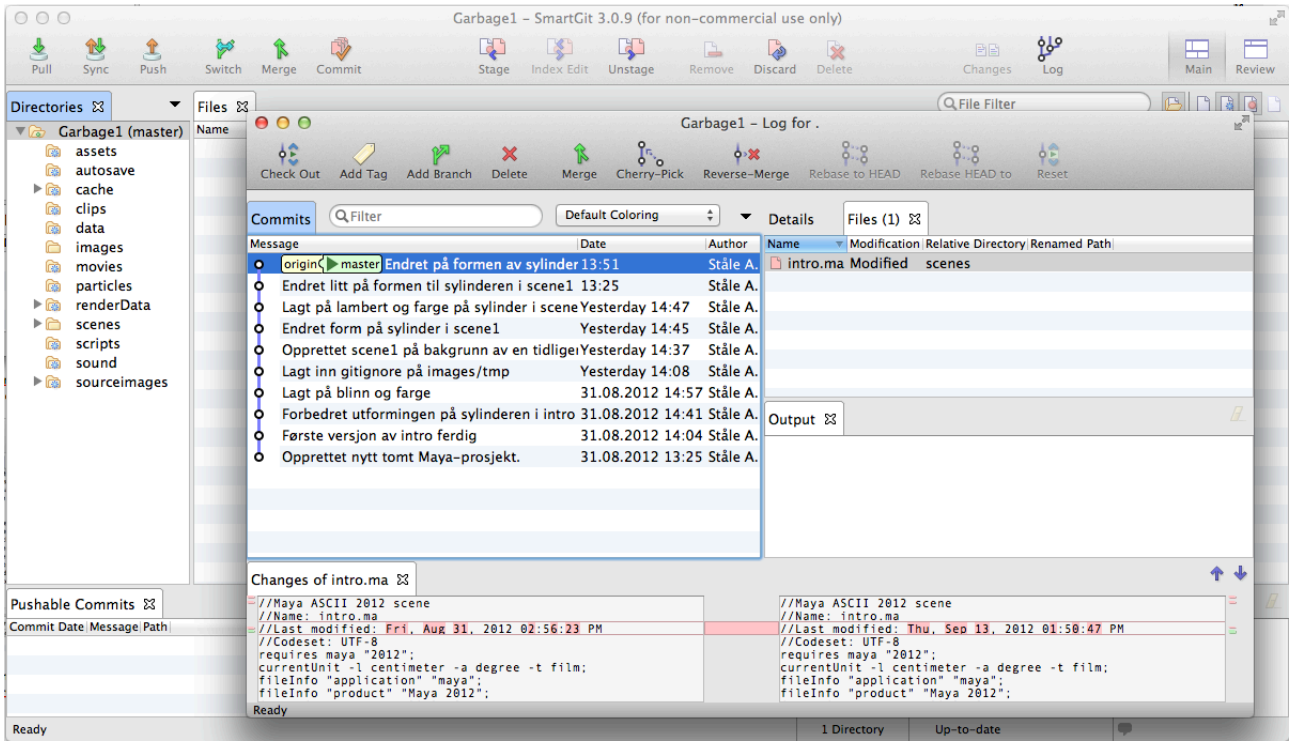
3.5 Git-verktøy

Git er i utgangspunktet et kommandobasert verktøy. Dvs. at alle Git-operasjoner kan kjøres i et terminalprogram ved å skrive kommandoen ”git” etterfulgt av opsjoner og argumenter.



Figur 11: Bruk av kommandobasert git i Terminal-vindu på Mac

Men det finnes også et stort utvalg av Git-applikasjoner med grafiske brukergrensesnitt (”GUI-klienter”) som gjør alle kommandoene tilgjengelig på en visuell og (for de fleste) mer intuitiv måte. I dette kompendiet skal vi benytte GUI-klienten **SmartGit** i eksemplene. Denne GUI-klienten er tilgjengelig både for PC og for Mac, og gratis for ikke-kommersiell bruk. SmartGit er også installert på alle relevante datasaler her på HiNT.



Figur 12: GUI-klienten SmartGit

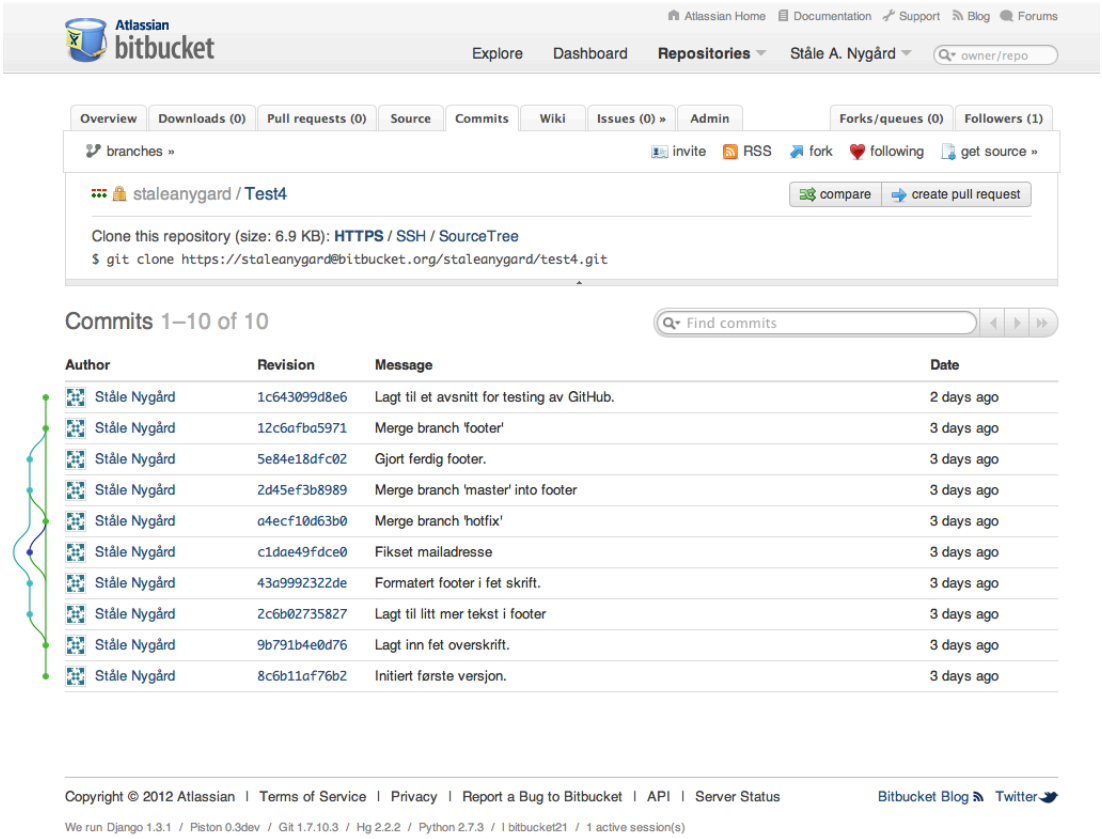
For de som ønsker å se på kommandobasert bruk av Git, og for de som ønsker å laste ned/installere Git på egen datamaskin, finner dere alt dere behøver å vite ved å følge lenkene i kapittel 3.6.

Det siste som kan nevnes i forhold til Git-verktøy, er at det finnes webtjenester som tar på seg ”hosting” av Git-repositories. Slike tjenester tilbyr typisk også i tillegg nettbasert prosjektoppfølging for hvert Git-prosjekt (dokumentasjon, sosiale community, issues tracker, distribusjon av kode, …). Her er en kort presentasjon av to slike tjenester som tilbyr noenlunde samme løsning:

- **GitHub** (https://github.com): En veldig utbredt, stabil og etablert tjeneste med mange medlemmer, og dermed et stort sosialt community (slagord: ”Social coding”). I et gratis GitHub-medlemskap kan

du ha et ubegrenset antall repositories så lenge de er åpent tilgjengelig for alle. Hvis du har behov for private repositories hvor du kun gir utvalgte personer tilgang må du betale månedsavgift (pr. dags dato $7 for inntil 5 private repositories, osv.).

- **BitBucket** (https://bitbucket.org): En nyere tjeneste (noen mener den er en "klone" av GitHub), som ikke har like mange medlemmer som GitHub, men som har økende oppslutning. BitBucket er spesielt populær pga. at de tilbyr et ubegrenset antall private repositories i gratis-medlemskapet (med tilgang for inntil 5 personer). BitBucket tilbyr "unlimited" medlemskap for studenter.



Figur 13: Visning av versjonshistorikk i BitBucket

3.6 Mer informasjon om Git og GUI-klienter

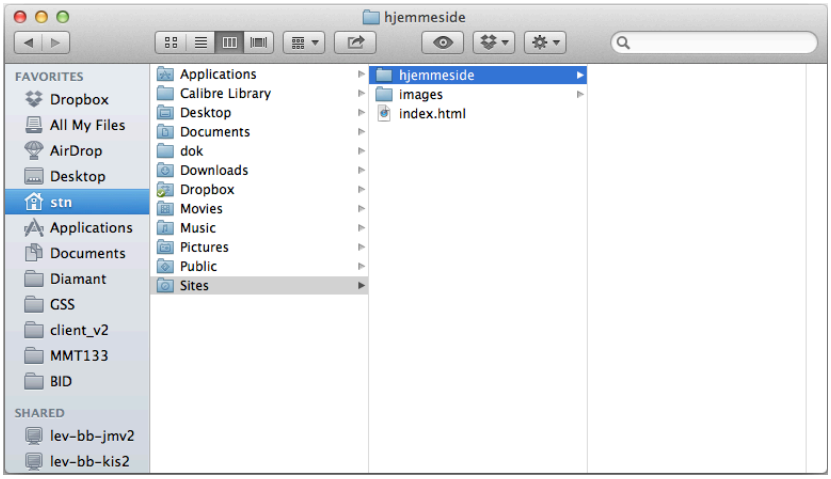
- Offisiell nettside Git: http://git-scm.com/
- Dokumentasjon Git: http://git-scm.com/doc
- Boken "Pro Git" (gratis): http://git-scm.com/book
- Nedlasting SmartGit (GUI-klient): http://www.syntevo.com/smartgit/
- Dokumentasjon SmartGit: http://www.syntevo.com/smartgit/documentation.html
- GUI-klient, alternativ – SourceTree (kun Mac): http://www.sourcetreeapp.com/
- Gratis hosting – BitBucket.org: https://bitbucket.org/

4 Eksempel – versjonskontroll nettside

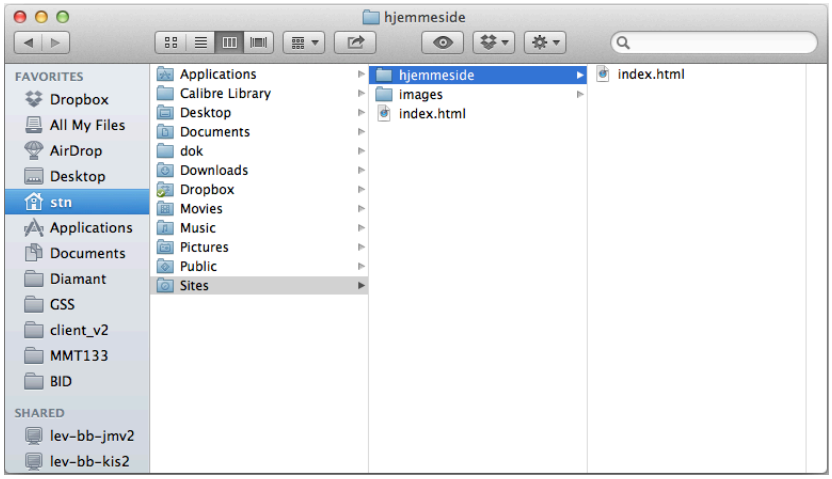
I dette eksemplet skal du bruke en HTML-editor (i eksemplet er Dreamweaver brukt) for å lage en veldig enkel nettside, samtidig som du lager nettsiden skal du gjøre versjonskontroll vha. SmartGit.

Eksemplet viser hvordan du oppretter et Git-repository i SmartGit, hvordan du oppdaterer versjonsdatabasen (**Stage**, **Commit**), hvordan du oppretter nye branches (**Add Branch**), hvordan du bytter mellom branches (**Switch**, **Check Out**), hvordan du fletter sammen branches (**Merge**), hvordan du leser versjonshistorikken (**Log**), hvordan du ser differansen mellom filer i ulike versjoner (**Changes**), og hvordan du setter inn tag'er i versjonshistorikken (**Add Tag**).

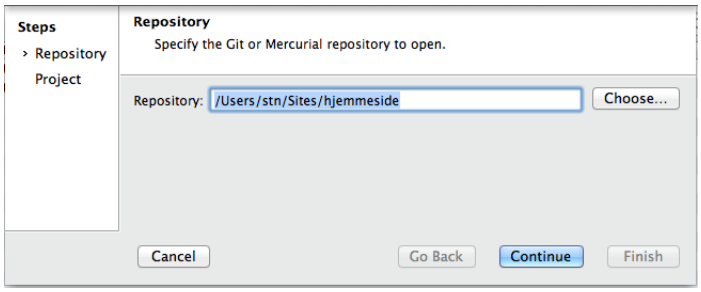
- Opprett en ny mappe for et nytt nettside-prosjekt på egnet sted (bruk Finder på Mac eller Windows Utforsker på PC)



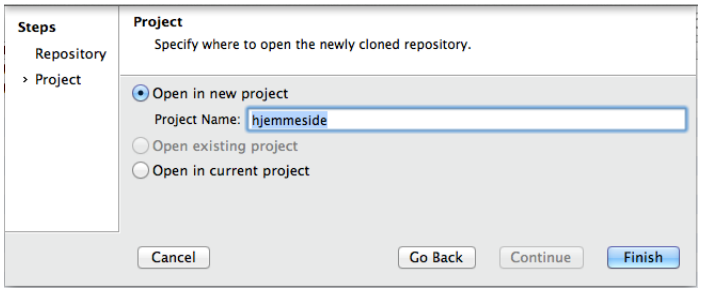
- Åpne din favoritt-HTML-editor (i vårt eksempel Dreamweaver, men en enkel teksteditor som Notepad fungerer også bra), og opprett et nytt HTML-dokument som du lagrer som index.html i den nye prosjektmappen – i Dreamweaver:
  - File > New…
  - I menyen Blank Page, velg Page Type: HTML, Layout: <none>
  - Bekreft med "Create"
  - File > Save
  - I dialogboksen "Save As" browser du deg fram til mappen du nettopp opprettet, og angir Save As: index.html
  - Bekreft med "Save"



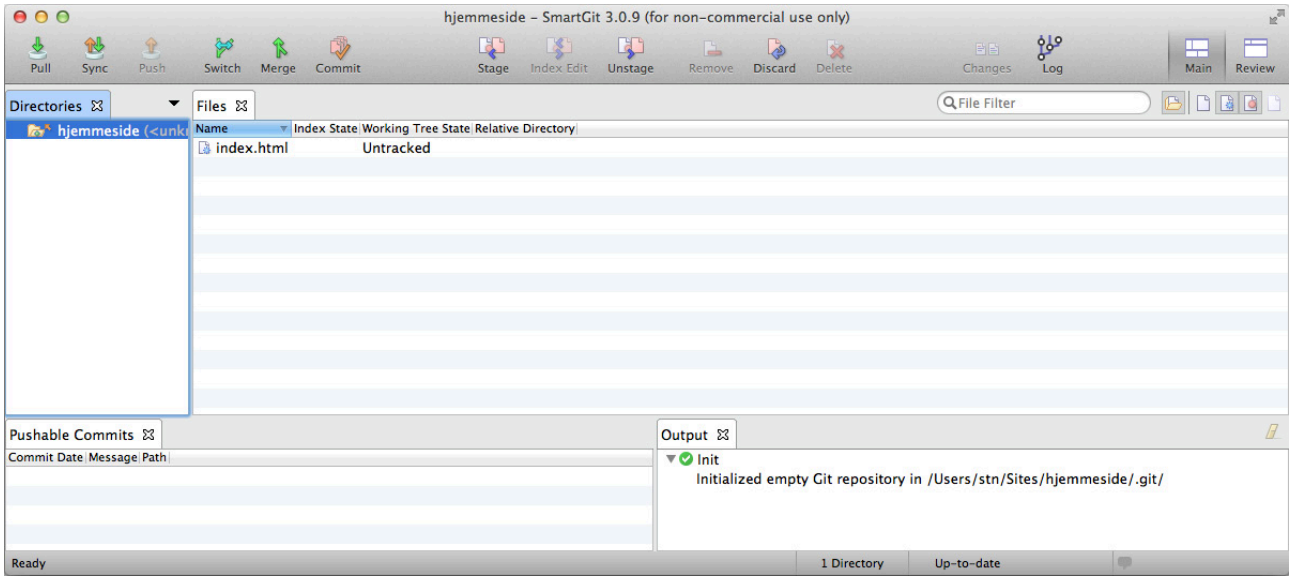
- Åpne SmartGit og initier et nytt Git-repository (versjonsdatabase) for nettsideprosjektet:
    - Project > Open Repository…
    - I neste vindu browse deg fram til (Choose…) rota på nettsideprosjektet



    - Bekreft med ”Continue”
    - Velg ”Git” i neste vindu ved spørsmål om hva slags repository som skal initieres
    - Velg ”Open in new project” i neste vindu, og velg et fornuftig ”Project Name”

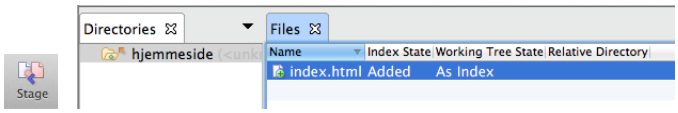


    - Bekreft med ”Finish” – et nytt Git-repository er nå opprettet; klar til håndtere versjonskontroll på alle filer som lagres i mappestrukturen til nettsideprosjektet

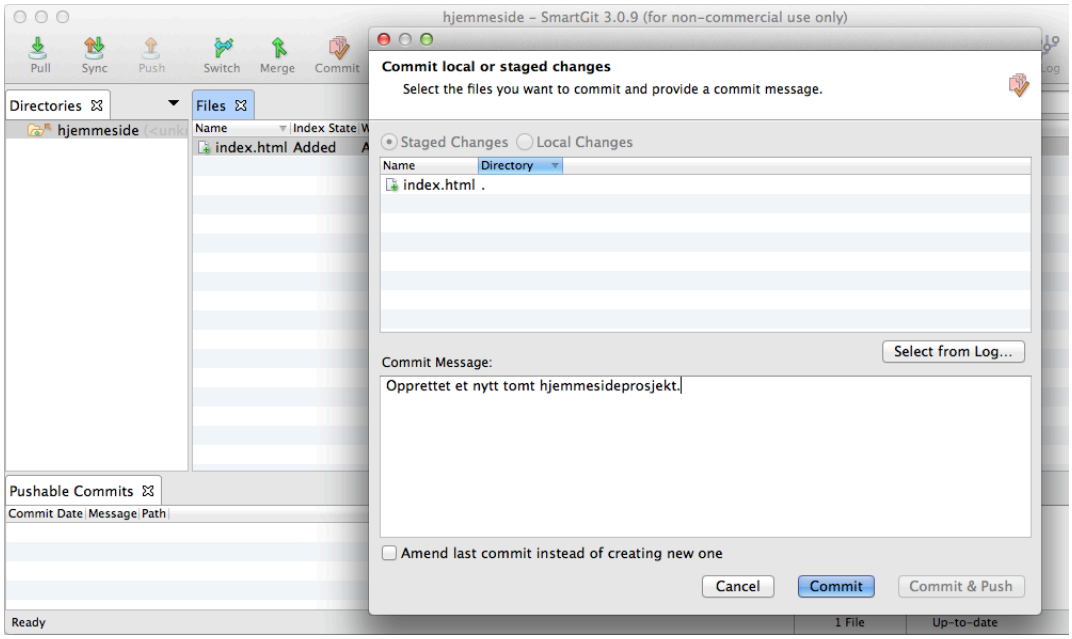


- Commit første versjon av prosjektet:
    - Velg rotmappa (i dette eksemplet ”hjemmeside”) i Directories-paletten til venstre i SmartGit

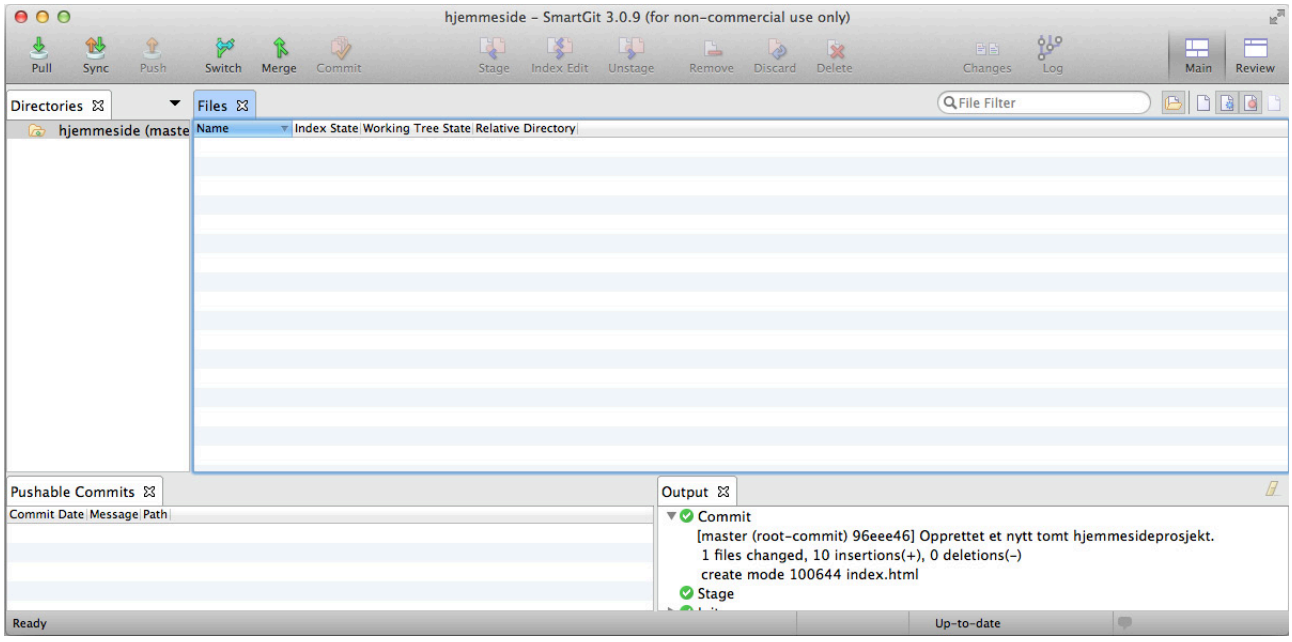
- Marker filen ”index.html” i Files-paletten, og klikk ”Stage” på verktøylinjen for å legge til (add) filen



- Klikk ”Commit” på verktøylinjen for å legge til første versjon (snapshot) av prosjektet i repository
- Skriv inn en beskrivende Commit-kommentar i neste vindu

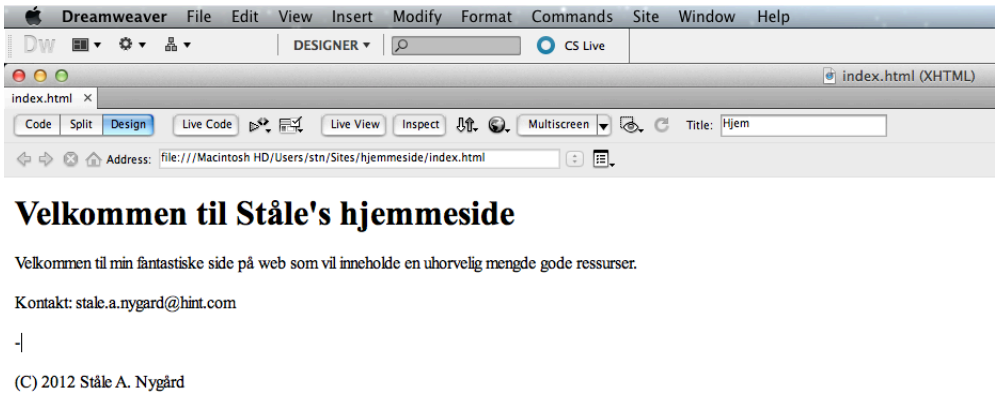


- Bekreft med ”Commit” – første snapshot av nettsideprosjektet er nå lagt inn i Git-repositoriet

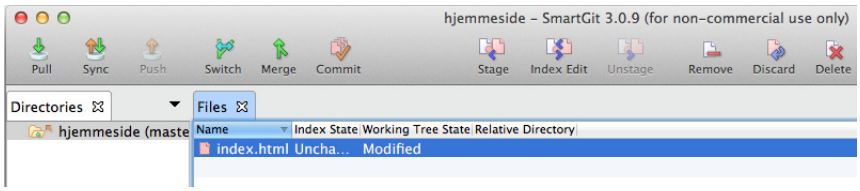


- Gjør noen endringer i index.html og Commit endringene i Git-repositoriet:
  - Bytt over til Dreamweaver/HTML-editoren (den beste arbeidsflyten er å ha både Dreamweaver/HTML-editoren og SmartGit åpne samtidig)

- Legg inn litt innhold/tekst på ”hjemmesiden”

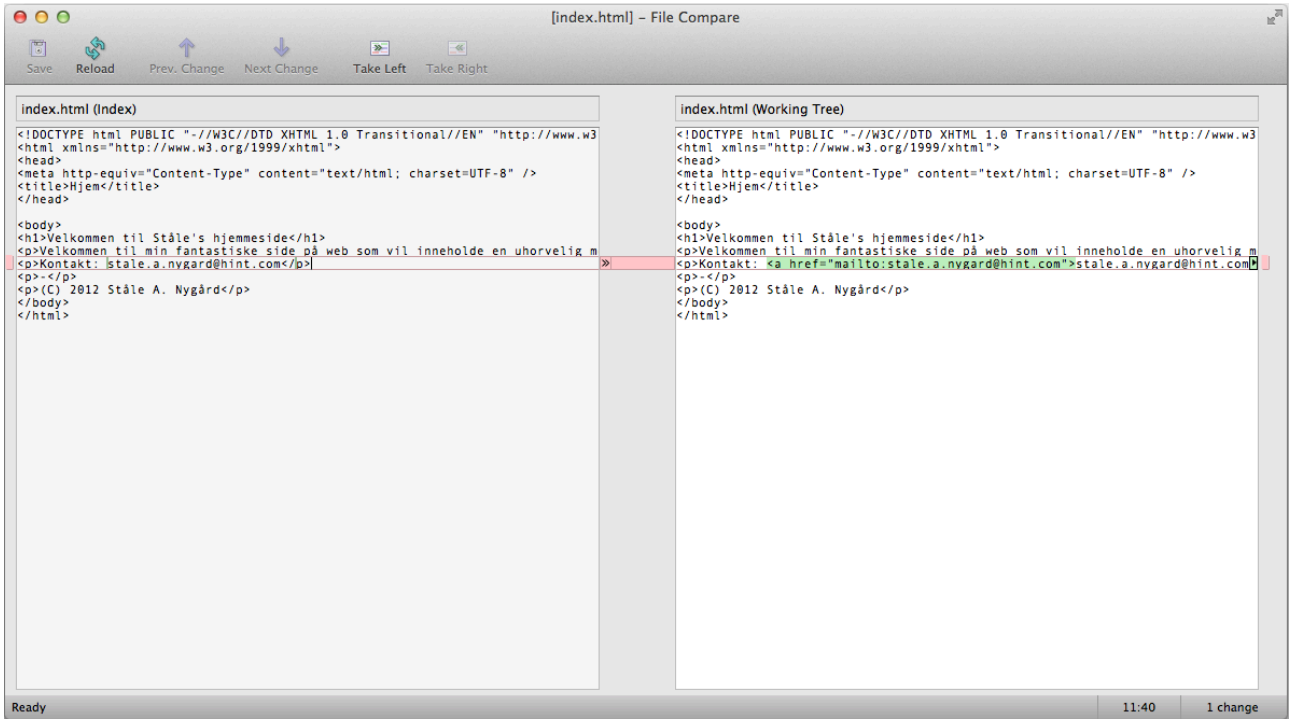


- Lagre endringene i index.html: File > Save
- Bytt til SmartGit og klikk eventuelt på rotmappa til nettsideprosjektet i Directories-paletten til venstre – den endrede index.html-filen skal nå komme opp i Files-paletten som ”Modified”

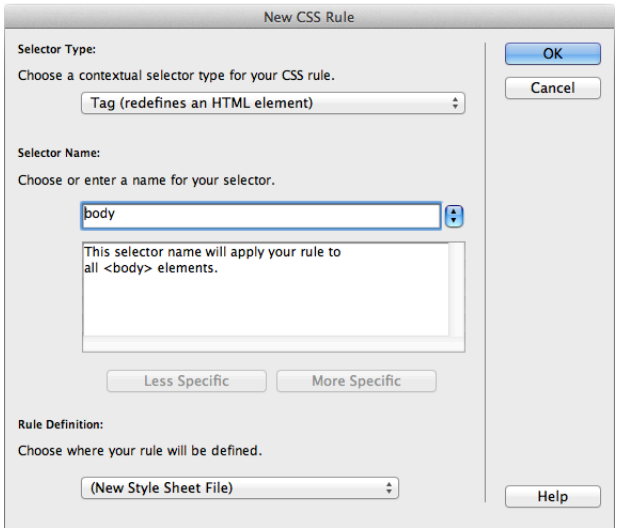


- Klikk på index.html i Files-paletten
- Klikk ”Stage” på verktøylinjen
- Klikk ”Commit” på verktøylinjen (i SmartGit kan man også velge Commit direkte uten å klikke Stage – operasjonen Stage vil da automatisk gjøres først i tillegg til Commit)
- Gi en beskrivende Commit-kommentar, og bekreft med ”Commit”
- Du har nå lagret versjon/smapshot nummer 2 i Git-repositoriet

- Gjør flere endringer i index.html, og sjekk forandringene:
  - Bytt til Dreamweaver/HTML-editoren igjen og gjør noen flere forandringer i index.html
  - Lagre endringene som ble gjort (File > Save)
  - Bytt til SmartGit og klikk på den modifiserte index.html i Files-paletten (**ikke** gjør en Commit enda)
  - Klikk ”Changes” på verktøylinjen
  - Se hvordan endringene som er gjort på denne filen siden siste commit blir presentert i neste vindu ”File Compare” (du kan lukke vinduet og kjøre en Commit etterpå)

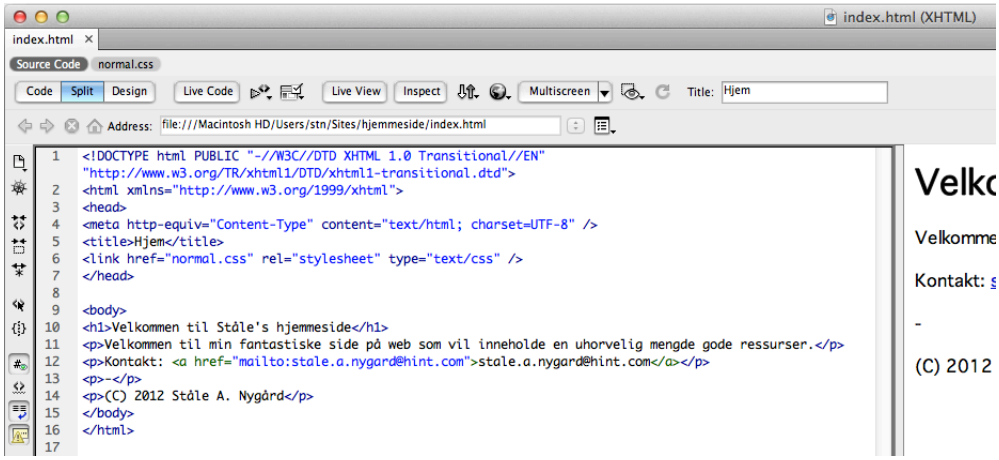


- Opprett et nytt eksternt CSS-stilark, og commit til Git-repository:
  - Bytt til Dreamweaver/HTML-editoren, og opprett et nytt eksternt stilark (i Dreamweaver kan dette gjøres ved å klikke ”New CSS Rule” i CSS Styles-paletten til høyre, for så å velge New Style Sheet File)

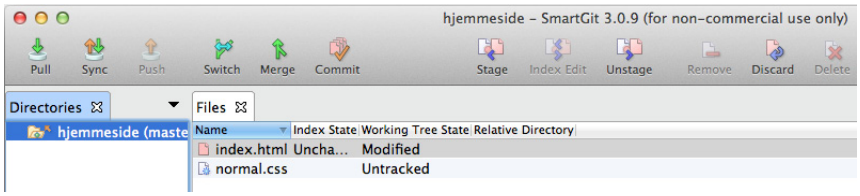


  - Lagre det nye stilarket som normal.css i samme mappe som index.html

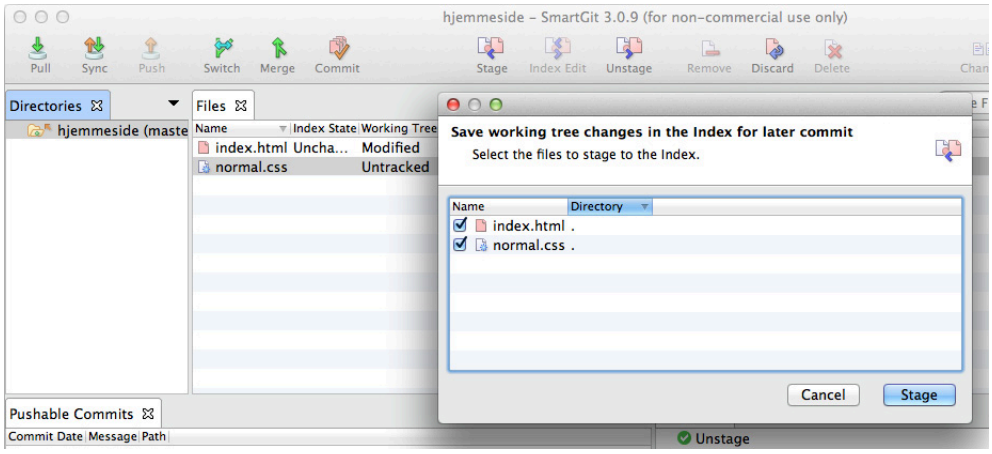
- Legg inn en referanse (link) til stilarket i index.html slik at de to er koblet sammen, og lagre endringene i index.html også



- Bytt til SmartGit og klikk eventuelt på rota av prosjektet i Directories-paletten – du skal nå finne index.html som Modified og normal.css som Untracked i Files-paletten



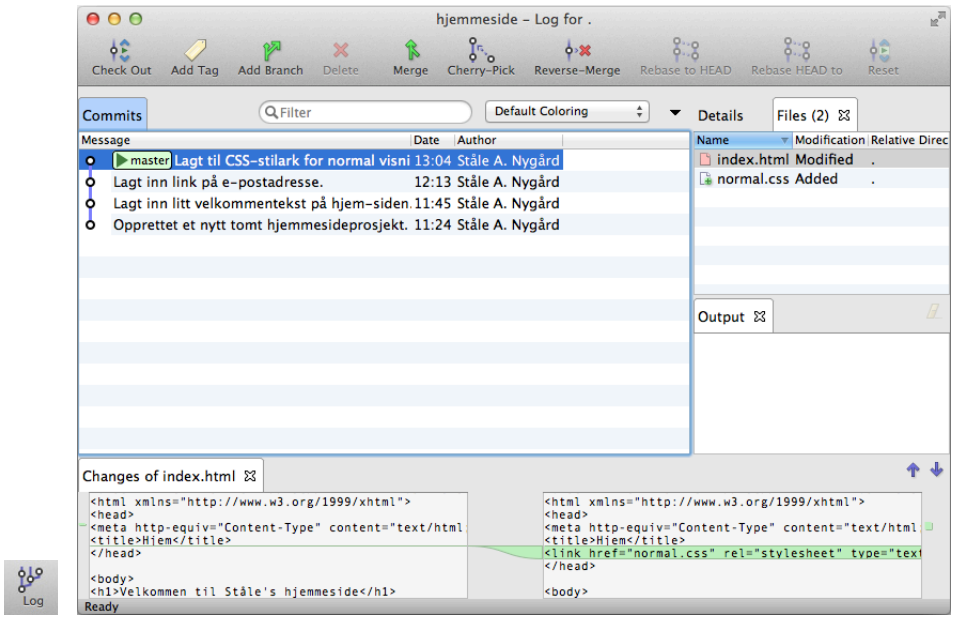
- Med rotmappa merket i Directories-paletten, klikk ”Stage” i verktøylinjen for å klargjøre begge filene for commit



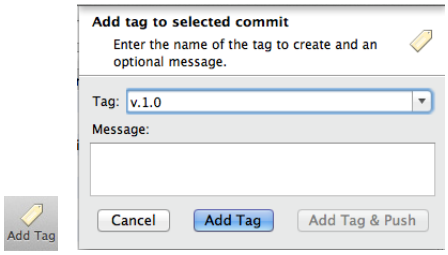
- Klikk ”Commit” for å legge inn ny versjon i Git-repositoriet

- Gjør ferdig og tag første versjon av nettsiden:
  - Gjør eventuelt endringer i index.html og normal.css til du er fornøyd med første versjon av nettsiden (avslutt med commit)
  - I SmartGit klikk på rotmappa til prosjektet i Directories-paletten til venstre

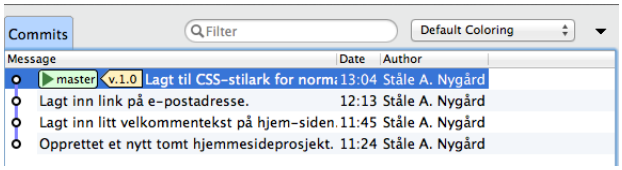
- Klikk ”Log” i verktøylinjen for å få fram versjonshistorikken så langt



- Klikk (merk) den øverste linjen i Commits-paletten (som tilsvarer siste versjon av prosjektet)
- Klikk ”Add Tag” på verktøylinjen
- Angi en Tag i neste vindu som angir at dette er versjon 1 (en tag kan ikke inneholde mellomrom og enkelte andre spesialtegn)

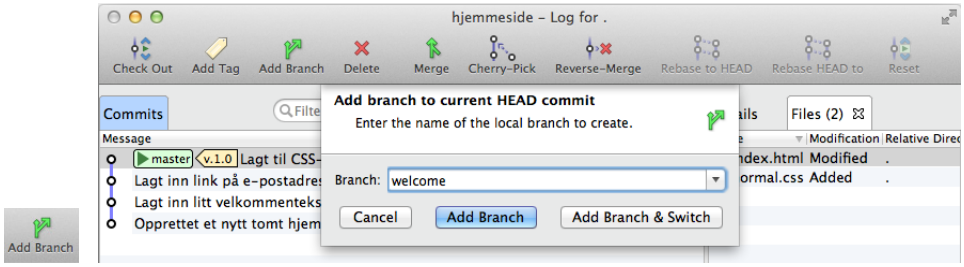


- Bekreft med ”Add Tag”

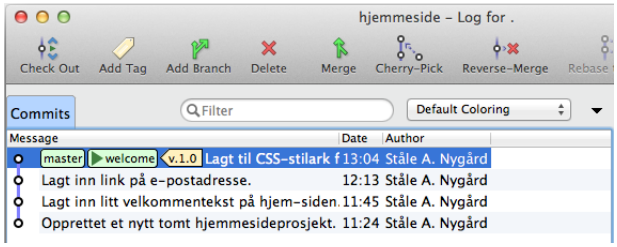


- Opprett en ny ”branch” for utvikling/testing av ny funksjonalitet:
  - I dette eksemplet tenker vi oss at vi skal forsøke å lage en dynamisk velkomsthilsning vha. Javascript (forskjellig velkomsthilsning til besøkende ved ulike tider på døgnet) – dette skal gjøres ved å opprette en ny ”branch” (ny utviklingsgrein) med utgangspunkt i versjon 1 av prosjektet
  - I SmartGit: Klikk rotmappa på prosjektet i Directories-paletten
  - Klikk ”Log” på verktøylinjen
  - Merk øverste linje i Commits-paletten i neste vindu (”v.1.0”-tag’en)
  - Klikk ”Add Branch” på verktøylinjen

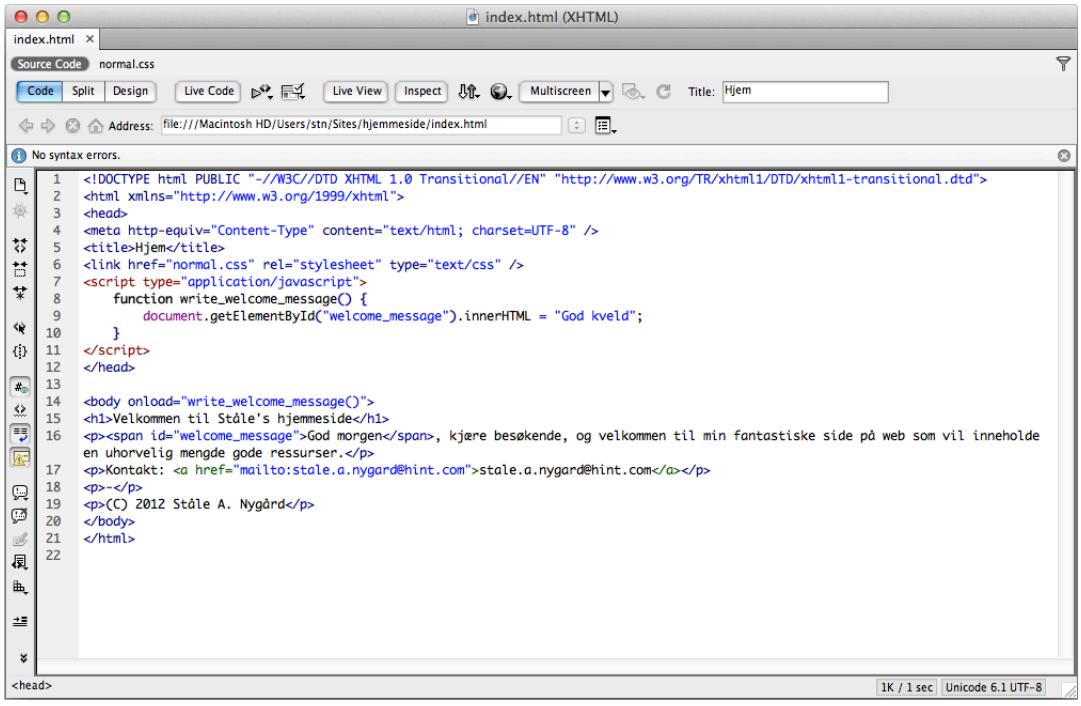
- I neste vindu angir du branch-navn: welcome



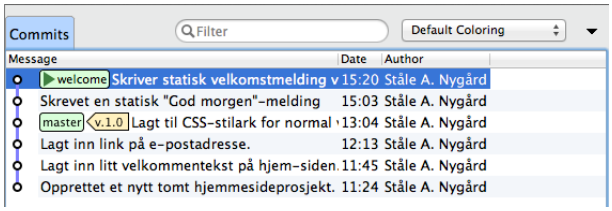
- Bekreft med ”Add Branch & Switch” – i Commits-historikken vil nå pilen angi at ”welcome” er aktiv branch (dette angir såkalt HEAD i arbeidskopien/working tree)



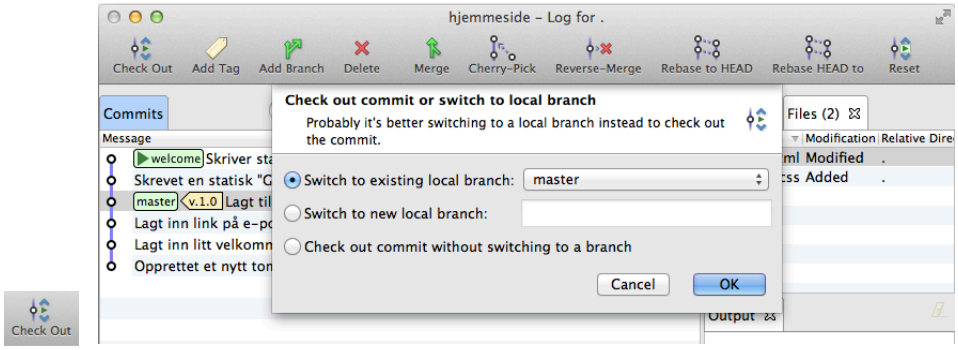
- Lukk Log-vinduet
- Bytt til Dreamweaver/HTML-editoren, og gjør noen forandringer, og noen commits som reflekterer at du jobber med velkomst-problematikken



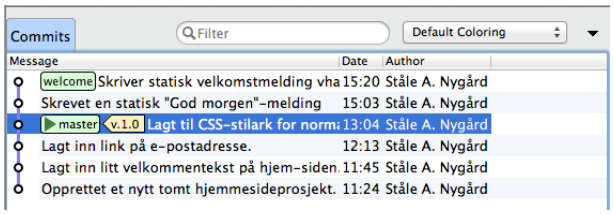
- Etter et par commits ser kanskje versjonshistorikken ut noe slikt i SmartGit (klikk ”Log”):



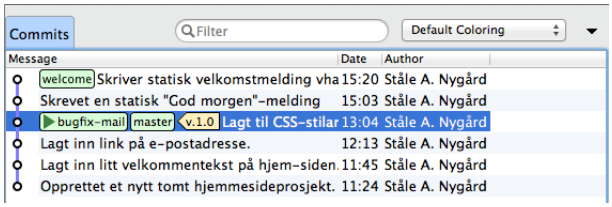
- Opprett en ny branch for bugfix:
  - Vi tenker oss nå at du har funnet ut at det er en feil på den publiserte versjonen av nettsiden (v.1.0) – nemlig at e-postadressen er feil – dvs. du må legge det uferdige arbeidet med den dynamiske velkomsthilsningen til side, ”rulle tilbake” til den publiserte versjonen av nettsiden, og rette opp e-postadressen
  - Bytt til SmartGit, og eventuelt commit modifiseringer som er gjort
  - Klikk på rotmappen til prosjektet i Directories-paletten, og klikk ”Log” på verktøylinjen
  - I neste vindu merker du linjen med versjon 1 i Commits-paletten (i dette eksemplet linje 3), og klikker ”Check Out” på verktøylinjen
  - I neste vindu velger du ”Switch to existing local branch: master”



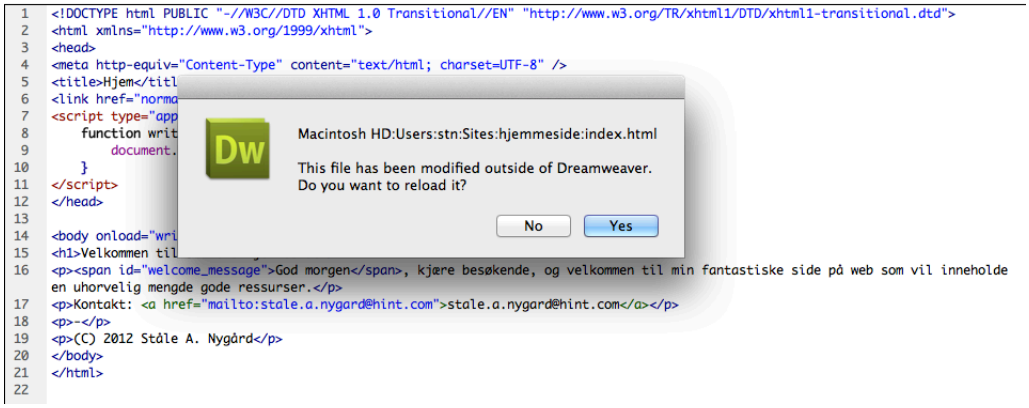
  - I versjonshistorikken som følger ser du nå at ”master” er aktiv branch i og med at det er en pil foran denne (angir HEAD i working tree)



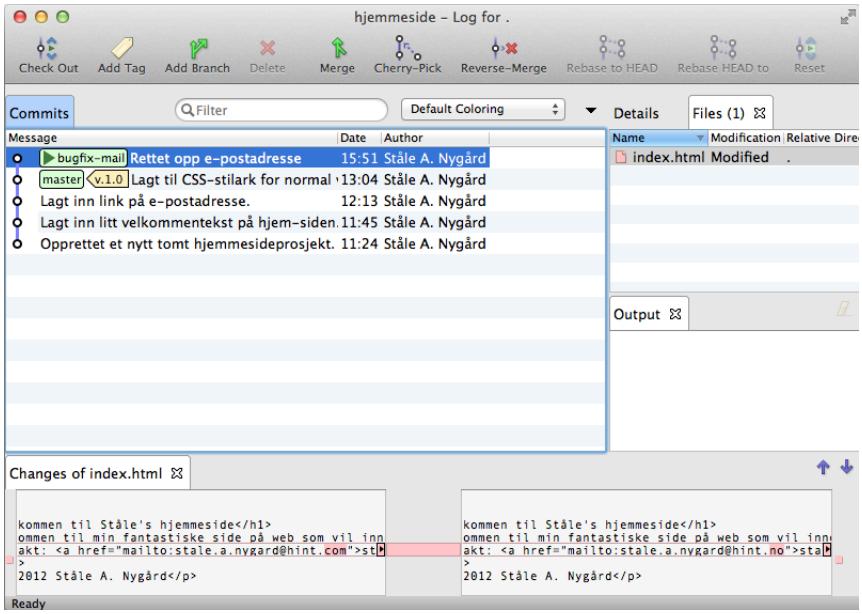
  - Med master/v.1.0 markert, klikk ”Add Branch” på verktøylinjen
  - Velg navnet ”bugfix-mail” i neste vindu, og bekreft med ”Add Branch & Switch” slik at ”bugfix-mail” blir aktiv branch



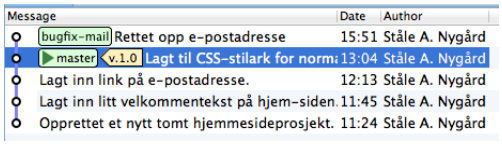
- Bytt til Dreamweaver/HTML-editoren, og åpne index.html på nytt (i Dreamweaver vil du automatisk få beskjed dersom åpne filer er endret, så her kan du trygt svare ”Yes” i den dialogboksen som dukker opp)



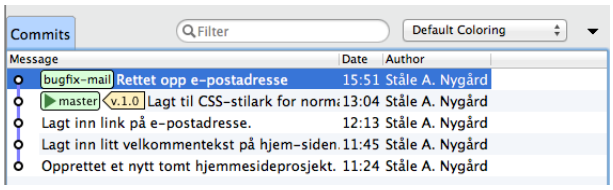
- Når du åpner index.html på nytt vil du se at de siste endringene ikke er der lengre – du har gått tilbake til versjon 1
- Gjør noen endringer (forandre på e-postadressen) i index.html , lagre endringene, og gjør en commit i SmartGit
- Versjonshistorikken i SmartGit vil nå se nogenlunde slik ut:



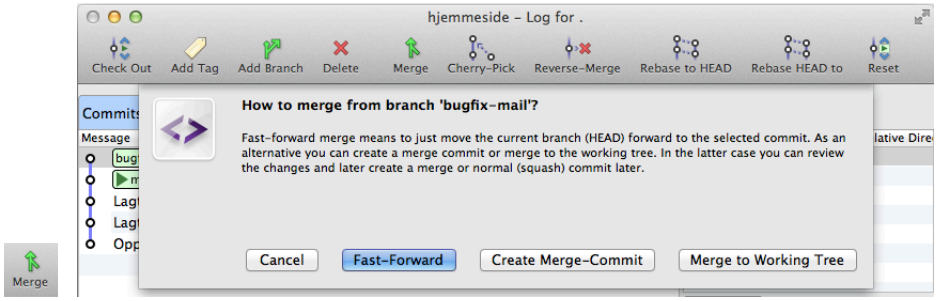
- Flett inn (merge) bugfix inn i master branch:
  - Vi sier oss nå fornøyd med bugfix’en, og skal derfor opprette en ny versjon av den publiserte nettsiden
  - Bytt til master/v.1.0 branch (Check out) slik at denne er aktiv branch (i og med at det er denne du skal oppdatere med bugfix-branch’en)



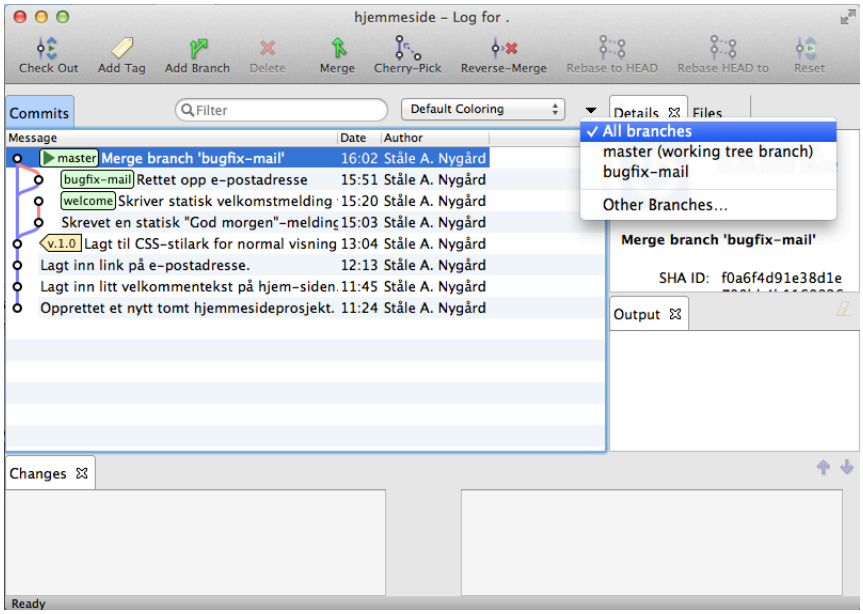
- Merk den siste commit'en til "bugfix-mail" branch'en i Commits-paletten



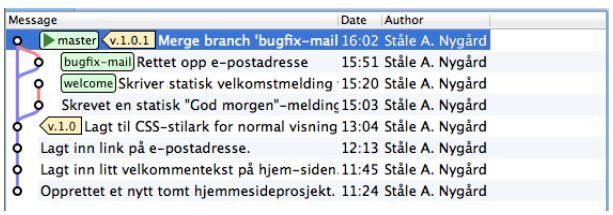
- Klikk "Merge" på verktøylinjen
- I neste vindu klikker du "Create Merge-Commit" i og med at vi ønsker å flette endringene gjort i "bugfix-mail" inn i aktiv branch "master"



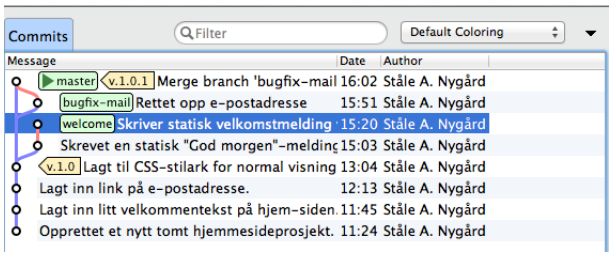
- For å se en fullstendig versjonshistorikk over alle branch'er klikker du "nedtrekkspilen" øverst til høyre i Commits-paletten, og velger "All branches" – versjonshistorikken skal nå se nogenlunde slik ut:



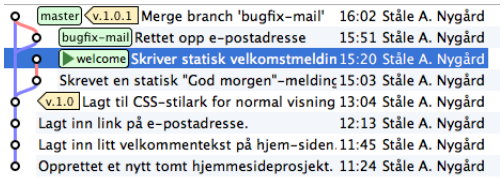
- Lag en ny tag "v.1.0.1" på den nye versjonen (merk at tag'er ikke er en nødvendighet; kun et hjelpemiddel for å gjøre versjonshistorikken mer lesbar)



- Bytt til welcome-branch'en og gjør ferdig denne utviklingsgreinen:
  - Åpne versjonshistorikken (klikk "Log" på verktøylinjen)
  - Merk øverste linje som hører til i welcome-branch'en, i dette eksemplet linje 3



  - Klikk "Check Out" på verktøylinjen
  - Velg "Switch to existing local branch: welcome" i neste vindu, og bekreft med "OK"
  - "welcome" skal nå være aktiv branch (den vil ha et pilsymbol foran seg)

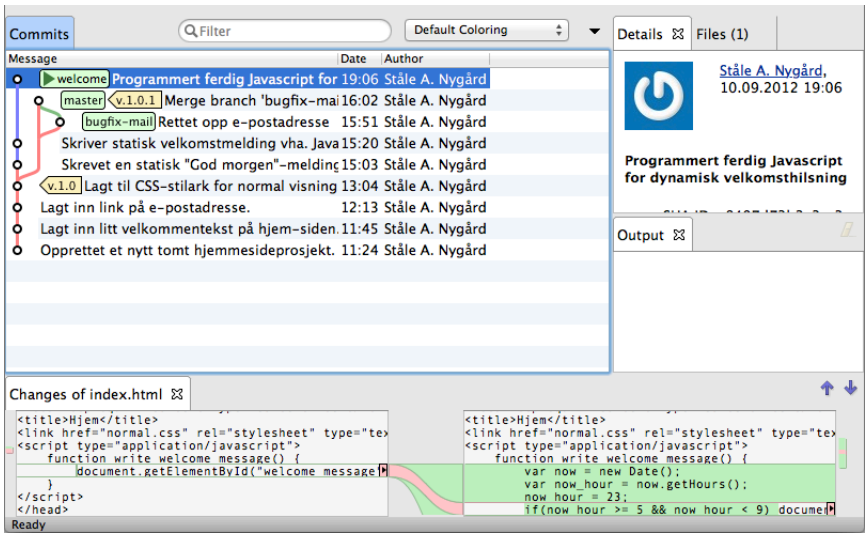


  - Bytt til Dreamweaver/HTML-editoren og åpne index.html på nytt (innholdet i denne skal nå være slik det var på slutten under arbeidet med den dynamiske velkomsthilsningen – merk at e-postadressen er feil igjen…)
  - Gjør ferdig programmeringen av den dynamiske velkomsthilsningen

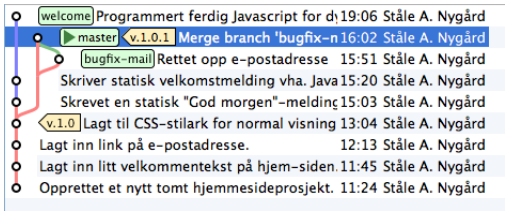
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Hjem</title>
<link href="normal.css" rel="stylesheet" type="text/css" />
<script type="application/javascript">
    function write_welcome_message() {
        var now = new Date();
        var now_hour = now.getHours();
        if(now_hour >= 5 && now_hour < 9) document.getElementById("welcome_message").innerHTML = "God morgen";
        else if(now_hour >= 9 && now_hour < 12) document.getElementById("welcome_message").innerHTML = "God dag";
        else if(now_hour >= 12 && now_hour < 18) document.getElementById("welcome_message").innerHTML = "God ettermiddag";
        else if(now_hour >= 18 && now_hour <= 23) document.getElementById("welcome_message").innerHTML = "God kveld";
        else document.getElementById("welcome_message").innerHTML = "Hei i natten";
    }
</script>
</head>

<body onload="write_welcome_message()">
<h1>Velkommen til Ståle's hjemmeside</h1>
<p><span id="welcome_message">God morgen</span>, kjære besøkende, og velkommen til min fantastiske side på web som vil
inneholde en uhorvelig mengde gode ressurser.</p>
<p>Kontakt: <a href="mailto:stale.a.nygard@hint.com">stale.a.nygard@hint.com</a></p>
<p>-</p>
<p>(C) 2012 Ståle A. Nygård</p>
</body>
</html>
```

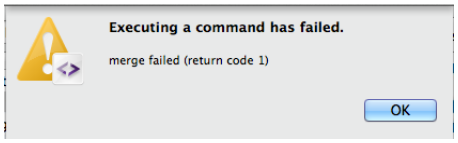
- Bytt til SmartGit og commit den ferdige versjonen

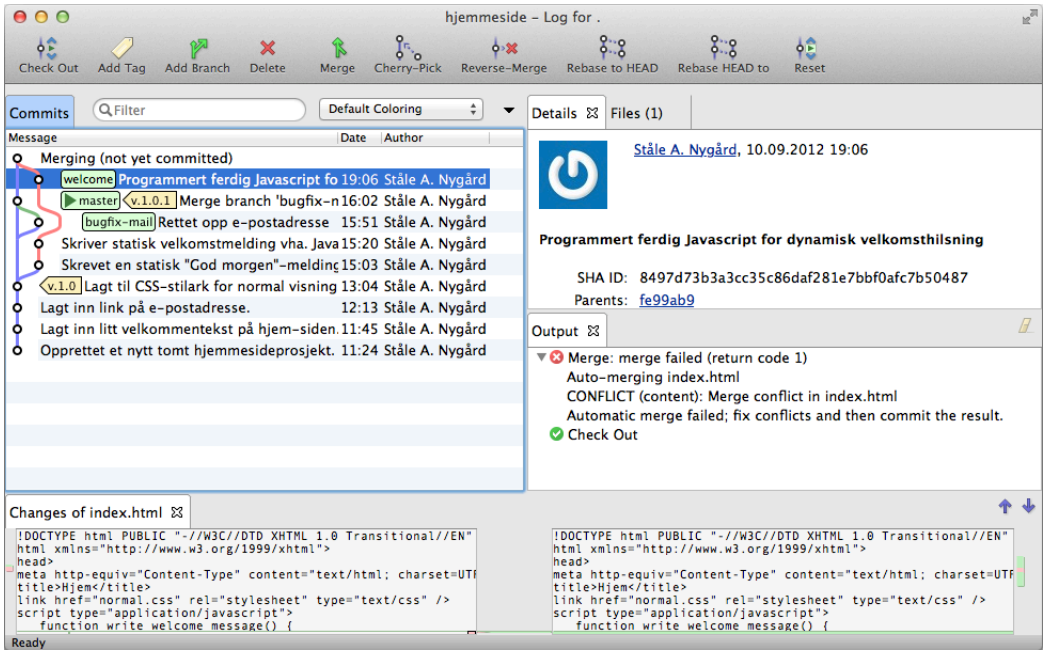


- Flett inn welcome branch'en inn i master branch'en som versjon 1.1:
  - Bytt til master/v.1.0.1-branch'en (merk i Commits-paletten, og klikk "Check Out")

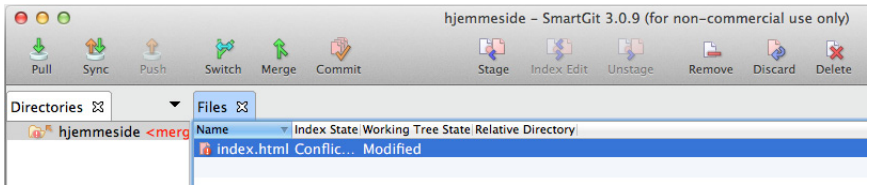


  - Merk øverste linje i welcome-branch'en i Commits-paletten
  - Klikk "Merge" i verktøypaletten (for å merge welcome-branch'en inn i aktiv branch som er master)
  - Klikk "Create Merge-Commit" i neste vindu
  - Ulikhetene mellom index.html i welcome i forhold til i master er i vårt tilfelle så store at Git ikke automatisk greier å flette inn forandringene, noe som skyldes at man på den ene siden ønsker å ta med seg forandringene på e-postadressen som er gjort i master, og at man på den andre siden ønsker å ta med seg den dynamiske velkomstteksten som er gjort i welcome – det har derfor oppstått en konflikt som må løses "manuelt" ved å redigere index.html

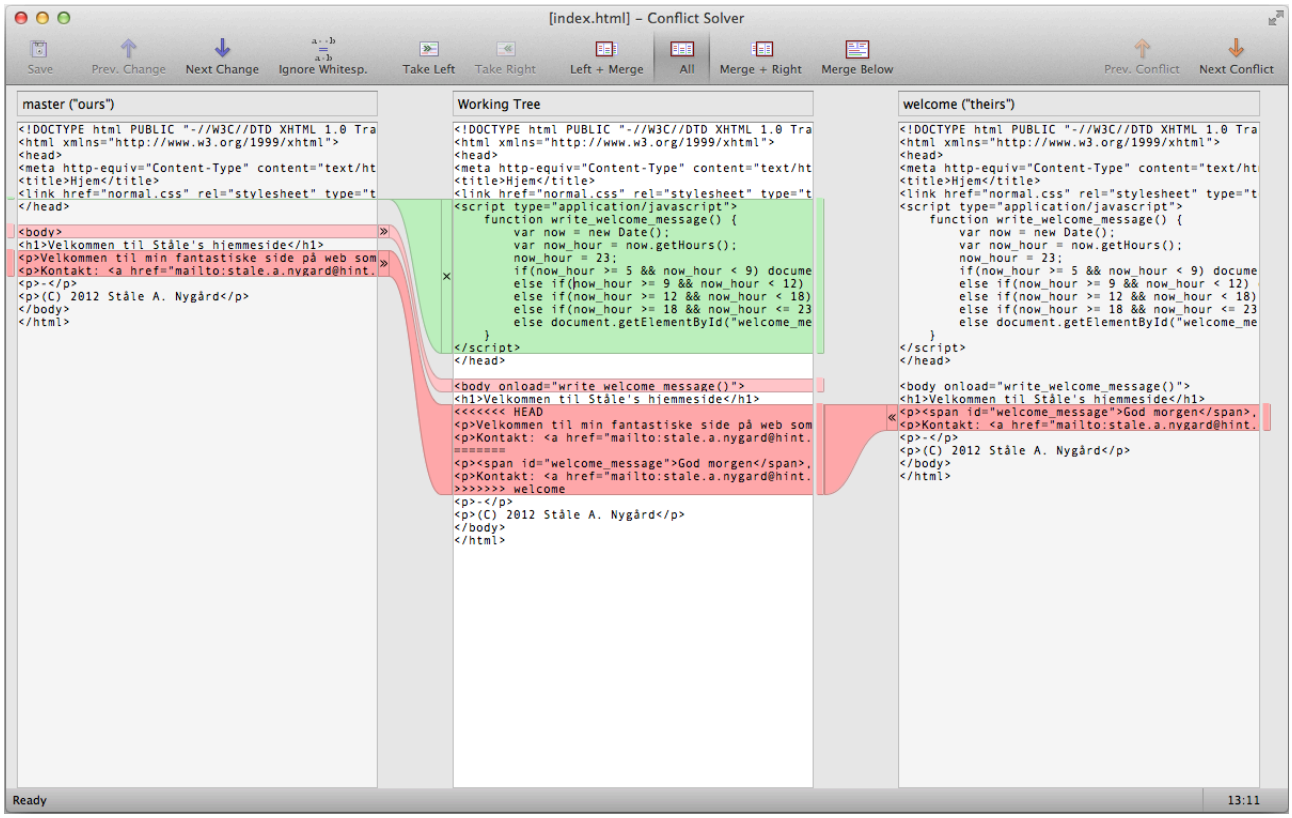




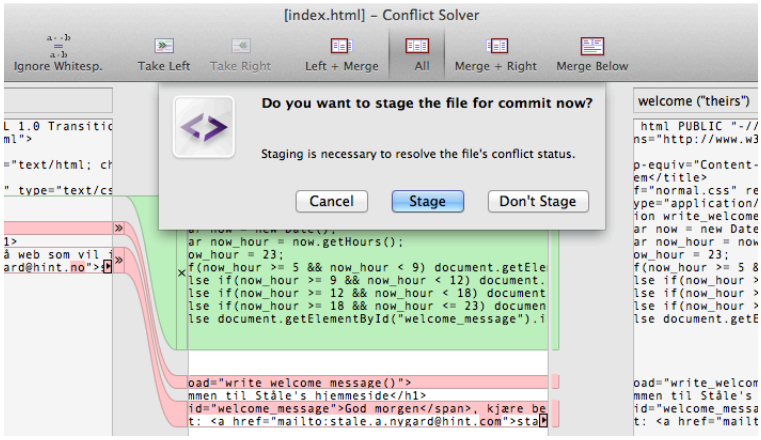
- Lukk versjonshistorikk-vinduet (Log)



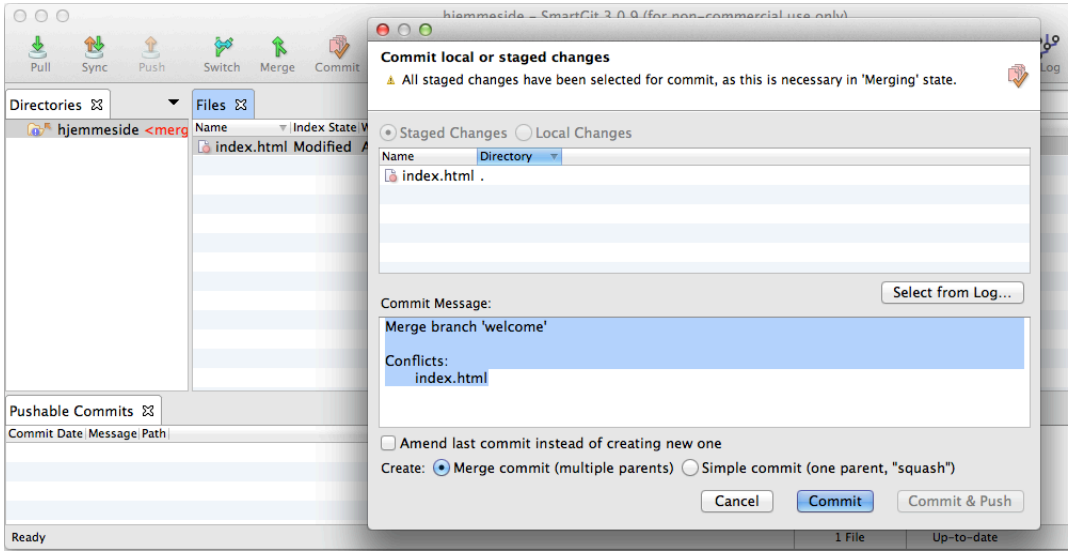
- Dobbeltklikk på den modifiserte index.html i Files-paletten for å åpne Conflict Solver-vinduet



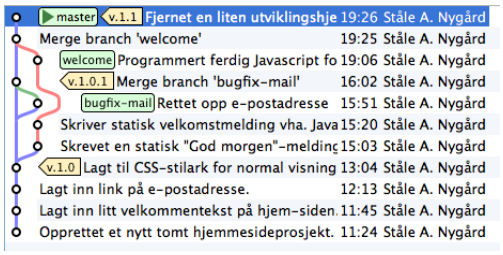
- Forklaring: versjonen i midten (Working Tree) er den som ligger lagret i arbeidskopien (denne kan du for så vidt åpne direkte i Dreamweaver/HTML-editoren og løse konfliktene der), versjonen til venstre er master-versjonen (som du skal merge til), og versjonen til høyre er welcome-versjonen (som du skal merge fra)
- Versjonen i midten er den endelige versjonen du skal bygge opp, og den viser at det i utgangspunktet er 3 konflikter; en grønn kodebit (denne regner Git med er OK i og med at den kommer fra welcome – men kodebiten kan likevel fjernes hvis det er ønskelig ved å klikke på ”x” til venstre) og 2 røde (disse kodebitene har hver sin versjon både i master og welcome – her kan man velge hvilken kodebitene man vil bruke ved å klikke ”>>” og ”<<”)
- I den nederste kodebiten har Git ikke noe forslag til hvilken som skal velges, så her må vi velge enten den til venstre (”>>”) eller den til høyre (”<<”) – i dette eksemplet ser vi faktisk at ingen av dem er helt riktig, så vi må velge en av dem og gjøre de siste konfliktsløsningene direkte i Dreamweaver/HTML-editoren
- Vi velger å benytte versjonen av den nederste kodebiten som er i welcome-branch’en (”<<”)
- Bekreft med ”Save”
- Lukk ”Conflict Solver”-vinduet, og klikk ”Don’t Stage” i neste vindu (vi har jo ikke løst konflikten skikkelig enda…)



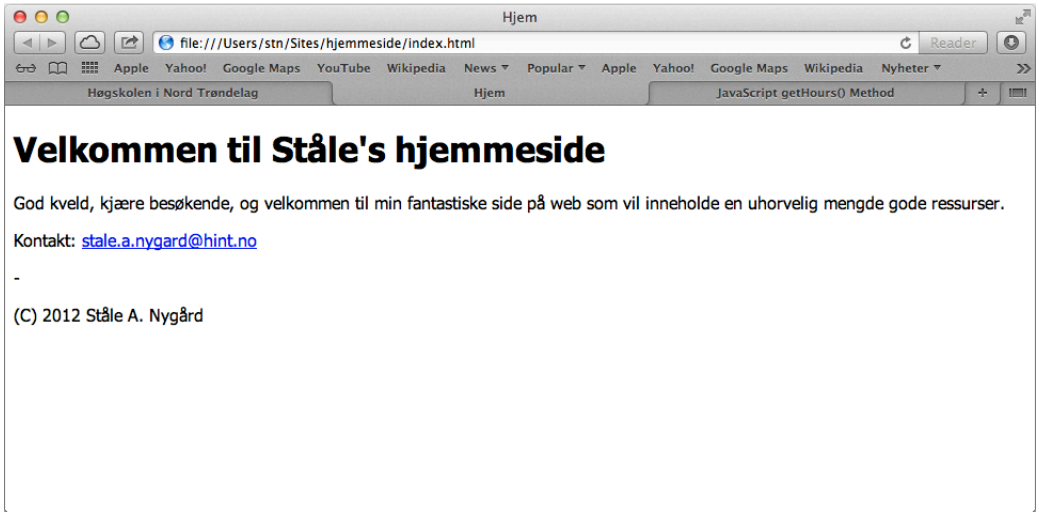
- Bytt til Dreamweaver/HTML-editoren og åpne index.html på nytt
- Rett opp slik at e-postadressen blir riktig igjen
- Lagre index.html
- Bytt til SmartGit og commit (dette vil bli en Merge Commit)



- Tag den nye versjonen som ”v.1.1” (jeg måtte gjøre en liten ekstra-commit for å luke bort en liten bug først)



- Slik ser da nettsiden i dette eksemplet ut i nettleseren kl. 19:26:

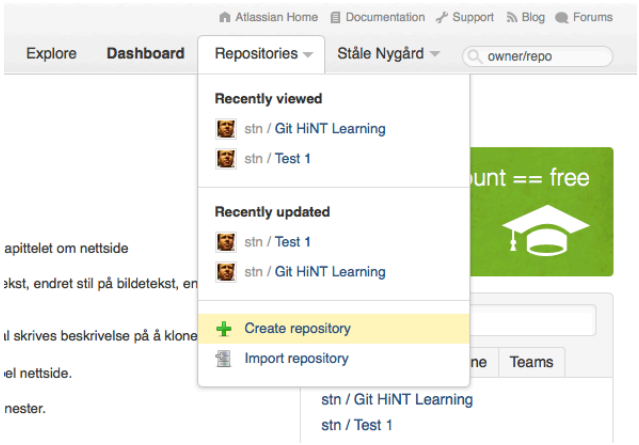


5 Eksempel – versjonskontroll med BitBucket

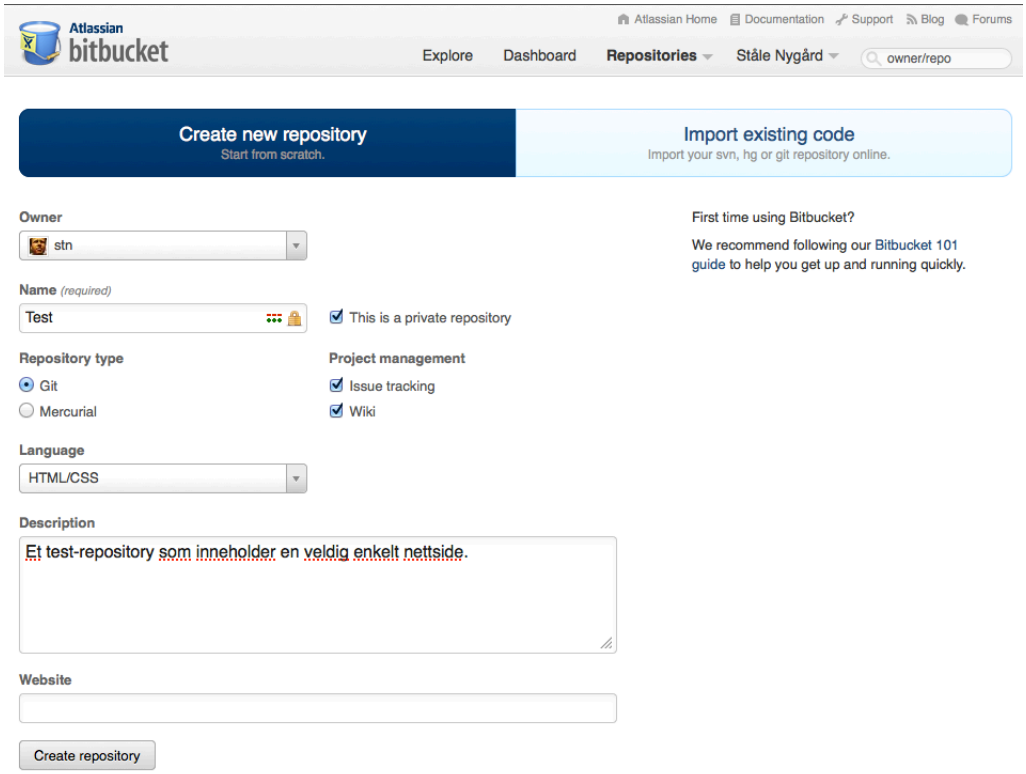
I dette eksemplet skal du bruke nettjenesten BitBucket som Git-server for å host'e nettsideprosjektet som ble opprettet i forrige kapittel.

Eksemplet viser hvordan du oppretter et nytt tomt repository i BitBucket, hvordan du importerer et lokalt repository til BitBucket-repository'et (**Add Remote**, **Push**), hvordan du jobber både lokalt og remote (origin) mot BitBucket (**Pull**, **Push**), hvordan du kopierer/kloner et prosjekt på BitBucket til lokal datamaskin (**Clone**), og hvordan du bruker **issue tracker**'en som prosjektstøtteverktøy på BitBucket.

- Opprett et nytt repository i BitBucket:
  - Logg deg eventuelt inn med din brukerkonto på BitBucket: https://bitbucket.org/ (hvis du ikke allerede har konto må du selvfølgelig opprette dette også først)
  - Velg "Create repository" i rullegardinmenyen "Repositories" i toppen

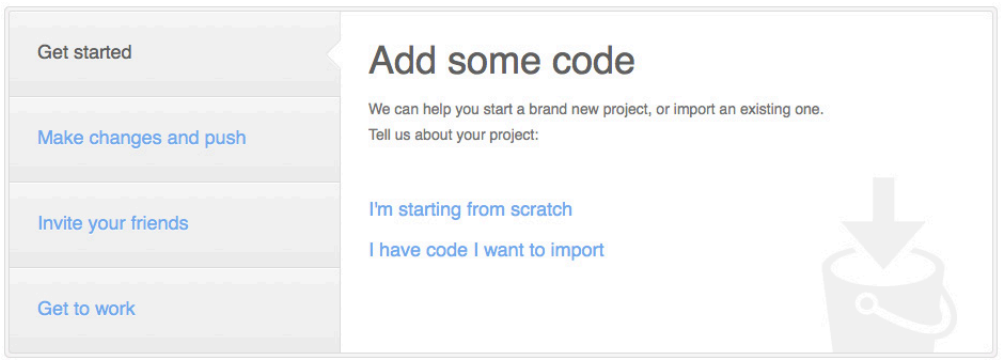


  - I neste bilde registrerer du informasjon om repository'et – gi et passende navn (Test), velg Git som type, HTML/CSS som language, og kryss av på både private repository, issue tracking og Wiki

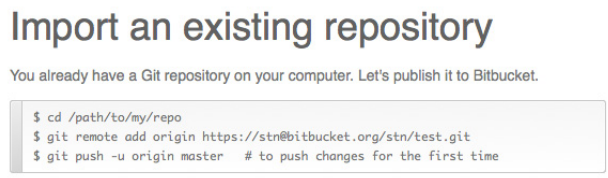


- Bekreft med ”Create repository” – du har nå opprettet et nytt repository på BitBucket sin server på adressen https://bitbucket.org/<username>/<repository-name>

- Importer et lokalt repository til serveren:
  - Gå eventuelt inn i startsiden til det nye repository’et – du finner det listet opp når du klikker ”Repositories” i toppmenyen (når du er innlogget)
  - På forsiden til det nye repository’et klikker du først ”Get started” og deretter ”I have code I want to import” under Add some code

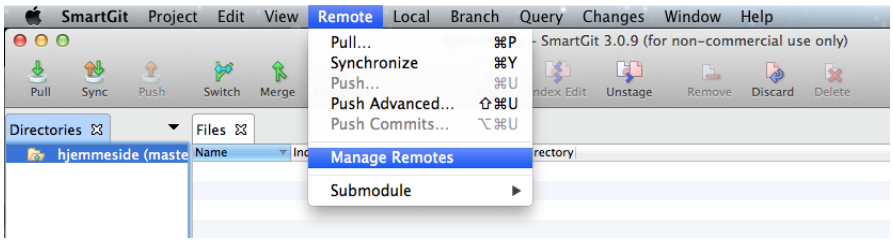


- Du vil nå få fram informasjon om hvordan du får importert et eksisterende lokalt repository inn i ditt repository som ligger på BitBucket sin server (dette er git-kommandoer som kan kjøres i et Terminal-vindu – vi skal benytte en GUI-klient)

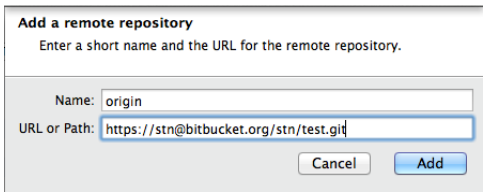


- Merk og kopier https-adressen til repository'et (høyreklikk > Copy)

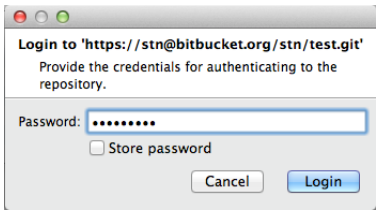
- Åpne/bytt til SmartGit
- Åpne eventuelt et lokalt repository i et eget vindu (i dette eksemplet benyttes det som ble opprettet i kapittel 0)
- I SmartGit velg: Remote > Manage Remotes



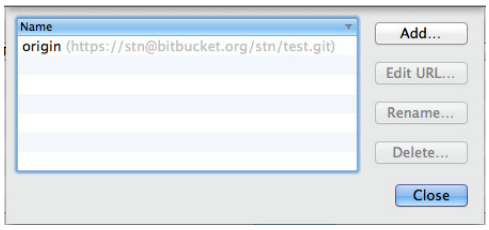
- I neste vindu klikk "Add…"
- I neste vindu lar du det stå Name: "origin", og limer inn https-adressen som URL or Path



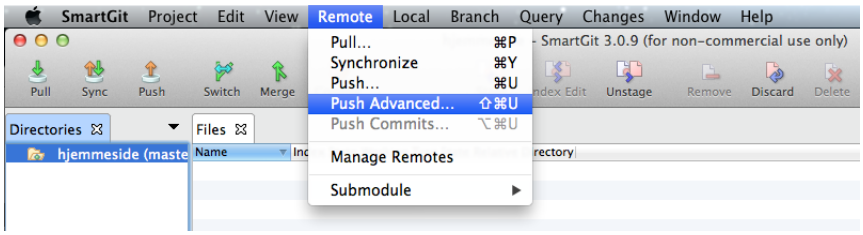
- Bekreft med "Add"
- I neste vindu skriver du inn ditt BitBucket-passord (brukernavnet står som en del av https-adressen)



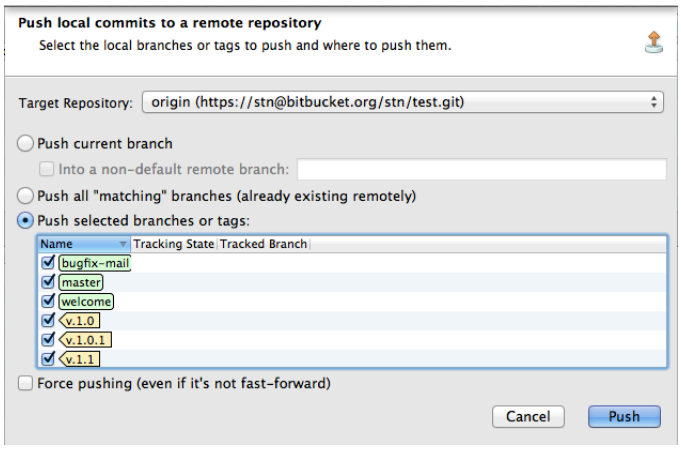
- Bekreft med ”Login” – du vil nå finne repository’et ditt på BitBucket koblet til det lokale repository’et via det logiske navnet ”origin” (det er fullt mulig å koble til flere)



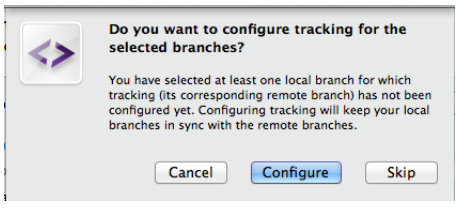
- Lukk Manage Remotes-vinduet – ”Close”
- Klikk rotmappa til Git-prosjektet i Directories-paletten til venstre
- I og med at det lokale repository’et i dette eksemplet består av flere branch’er velges en ”advanced push” første gangen: Remote > Push Advanced…



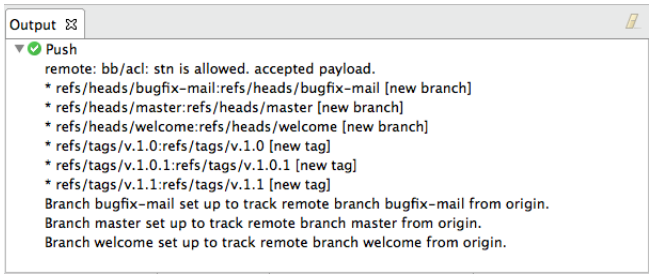
- I neste vindu velger du Target Repository: origin, velger Push selected branches or tags og huker av for alle branch’er og tag’er



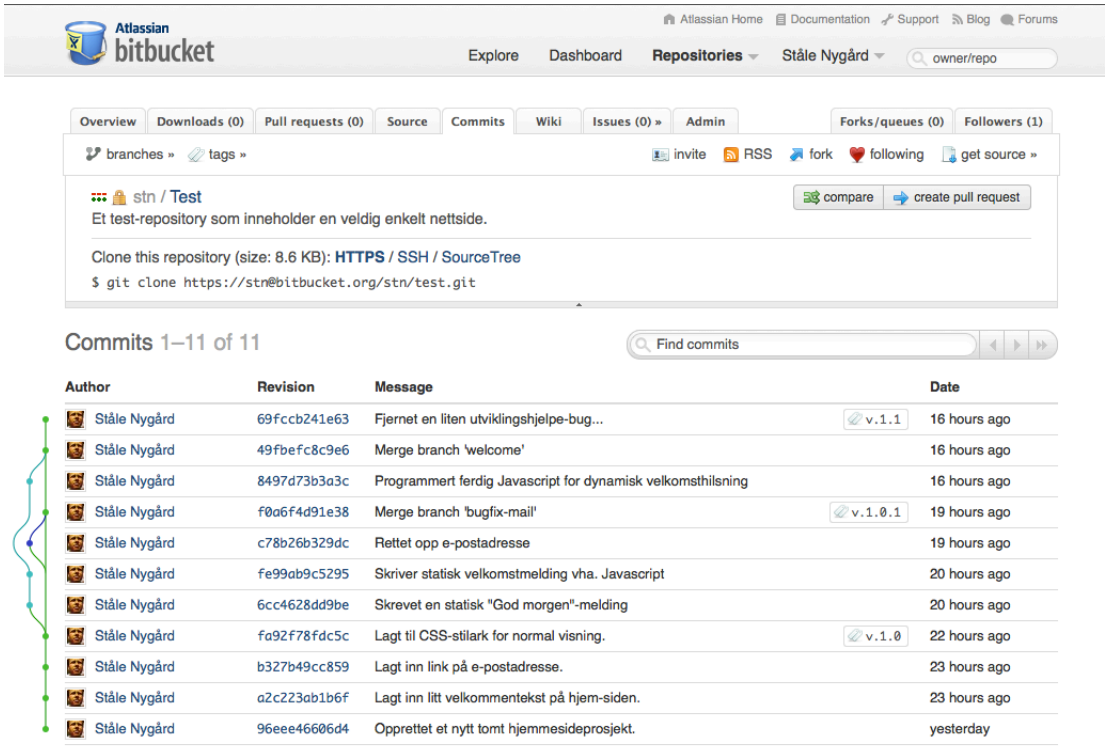
- Bekreft med ”Push”
- I neste vindu klikker du ”Configure” for å sette opp håndtering av synkronisering mellom remote (origin) og lokal versjon



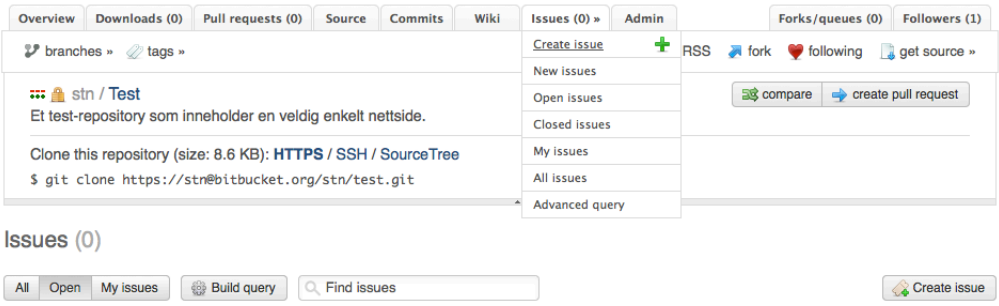
- Hele Git-prosjektet, inkludert filer og versjonsdatabase, sendes/importeres nå forhåpentligvis til BitBucket-serveren (dette kan ta litt tid) – klart til å deles med andre



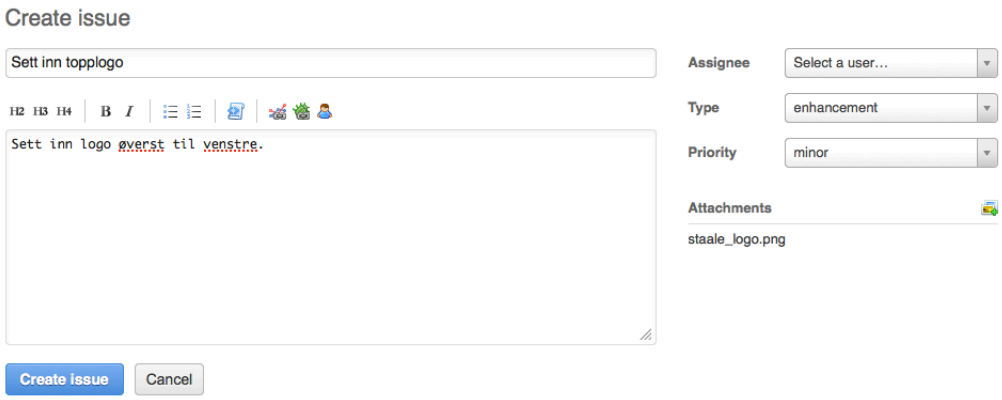
- Bytt til nettleseren og åpne repository'et på nytt for å kontrollere at importen ble vellykket



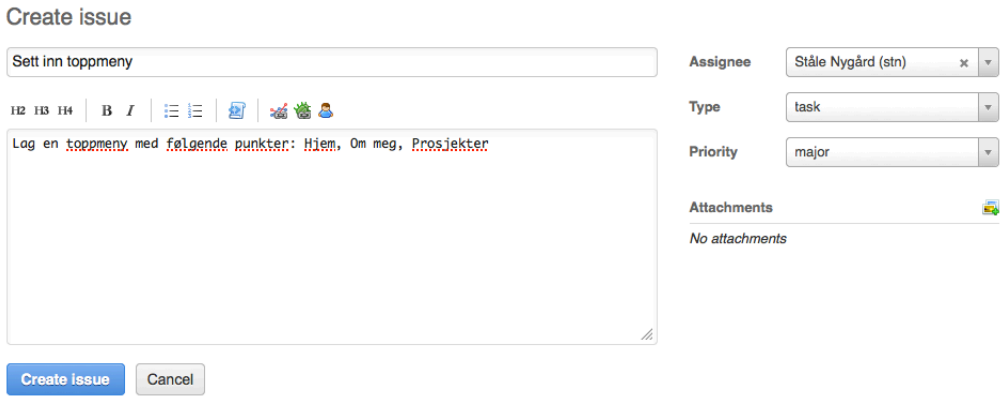
- Registrer et par "issues" i issue tracker'en:
  - I BitBucket: velg "Create Issue" i nedtrekksmenyen "Issues" (eventuelt klikk knappen "Create issue" i Issues-listen på Issues-siden)



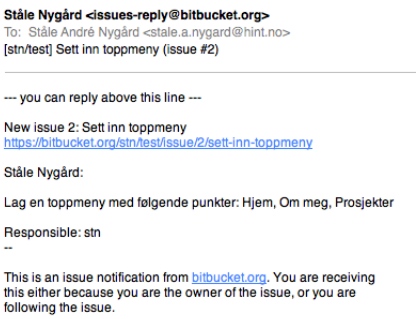
- I neste bilde registrerer du informasjon om saken/problemet som kan/skal løses, i dette eksemplet: Title: Sett inn topplogo, Assignee: ingen spesielle har ansvaret…, Type: enhancement, Priority: minor – pluss at det i vårt eksempel er valgt et bilde som skal lastes opp som vedlegg



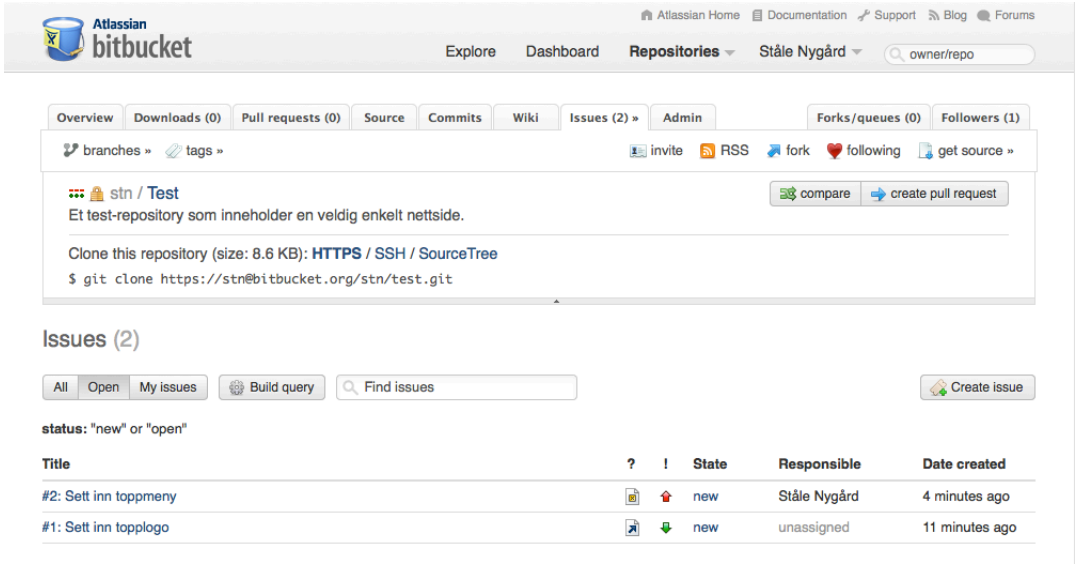
- Bekreft med ”Create Issue”
- Sett inn en ny issue ”Sett inn toppmeny”, med deg selv som ansvarlig (Assignee), task som type og major som prioritet



- Hvis du har standardoppsettet på BitBucket vil du nå få en notification-mail tilsendt fra BitBucket (dette kan ta litt tid)

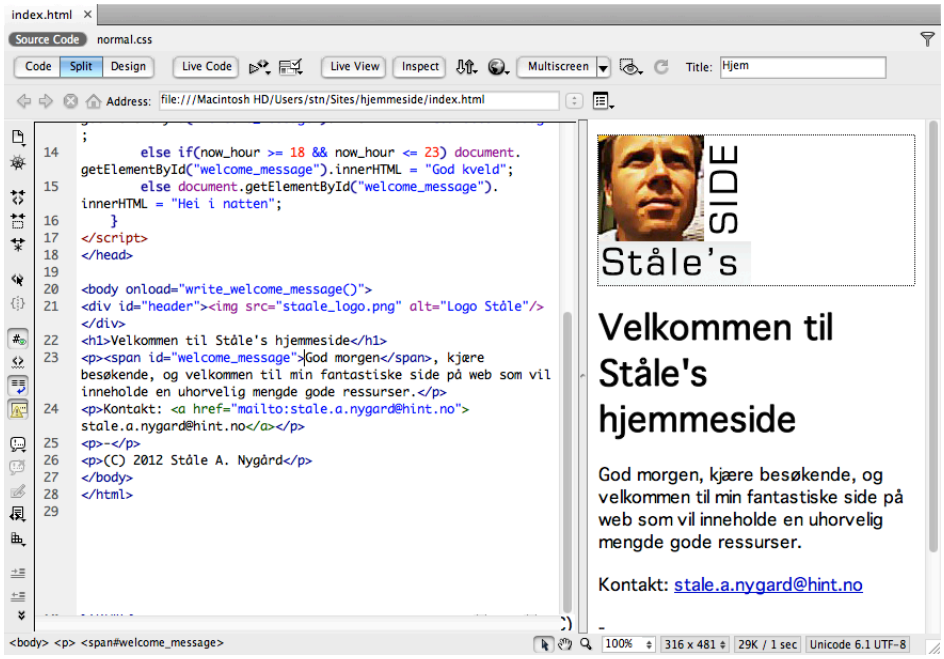


- Klikk ”Issues” i toppmenyen for å få fram en liste over issues



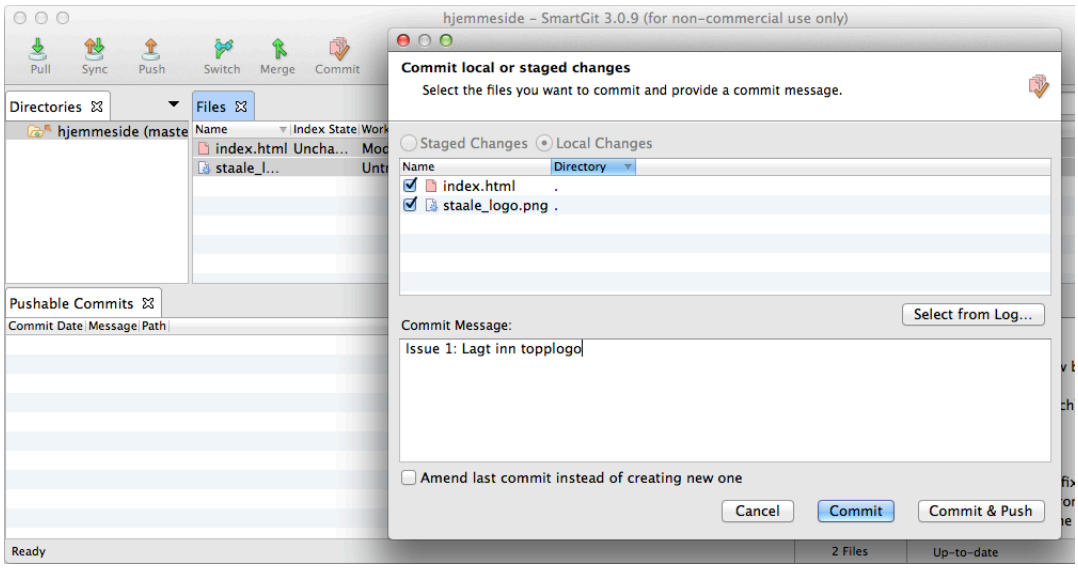
  - Merk: Issues er ikke en integrert del av Git, dette er kun en separat prosjektoppfølgingstjeneste som BitBucket (og andre lignende tjenester) tilbyr

- Gjør noen lokale endringer og synkroniser med serveren:
  - Sjekk eventuelt ut master branch’en på hjemmesideprosjektet
  - Åpne/bytt til Dreamweaver/HTML-editoren og åpne index.html
  - Sett inn en logo øverst til venstre på siden

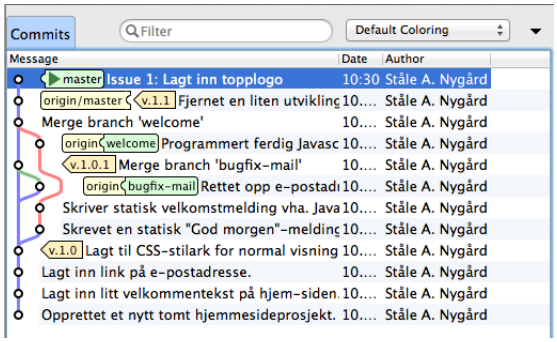


  - Lagre endringene i index.html

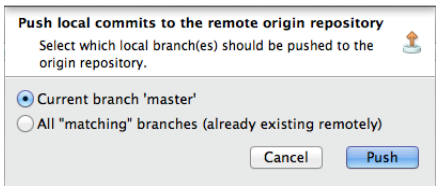
- Bytt over til SmartGit, og commit endringene



- Klikk rotmappa i Directories-paletten til venstre og klikk ”Log” for å få fram versjonshistorikken

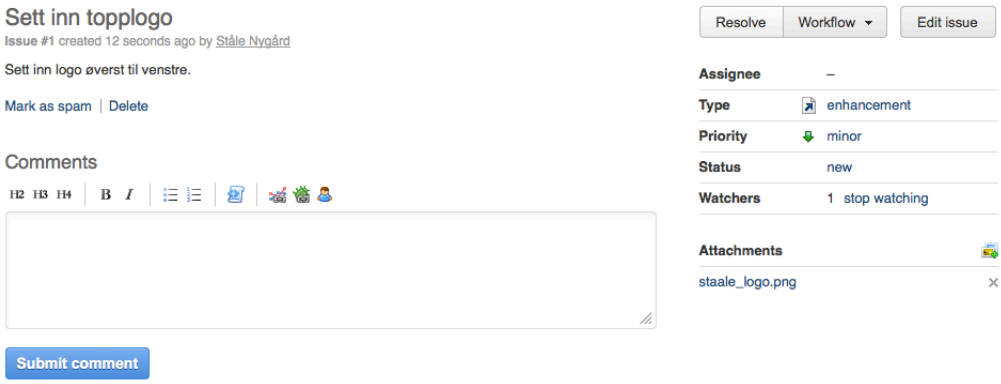


- Commits-grafen viser at lokal- og server-versjon ikke er synkronisert – klikk ”Push” (eventuelt ”Synchronize”) for å overføre lokale endringer til serveren (det er ikke nødvendig å push’e til serveren for hver lokal commit – typisk push’es det når man avslutter arbeidet lokalt, eller dersom man har gjort endringer som samarbeidende personer skulle ha pull’et inn i sin lokale arbeidskopi)

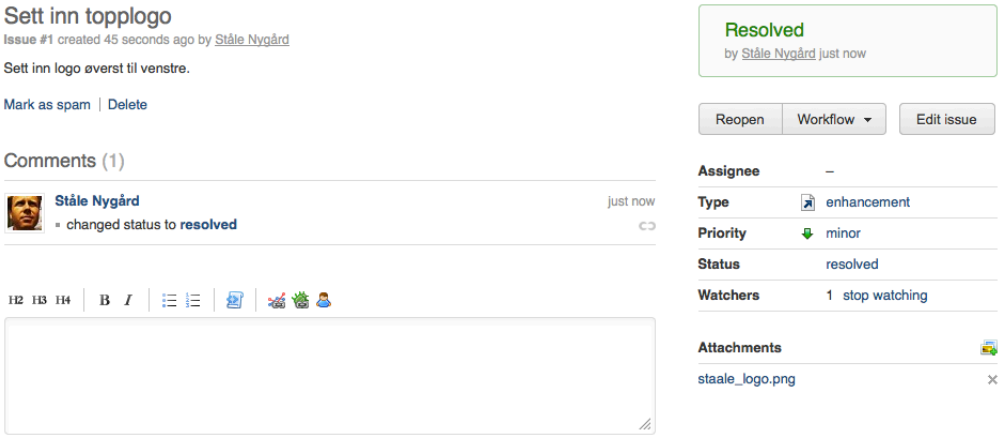


- Sjekk versjonshistorikken både lokalt (SmartGit – ”Log”) og på BitBucket (”Commits”)
- I BitBucket: Klikk ”Issues” for å få fram listen over issues
- Klikk issue #1 i listen (”Sett inn topplogo”)

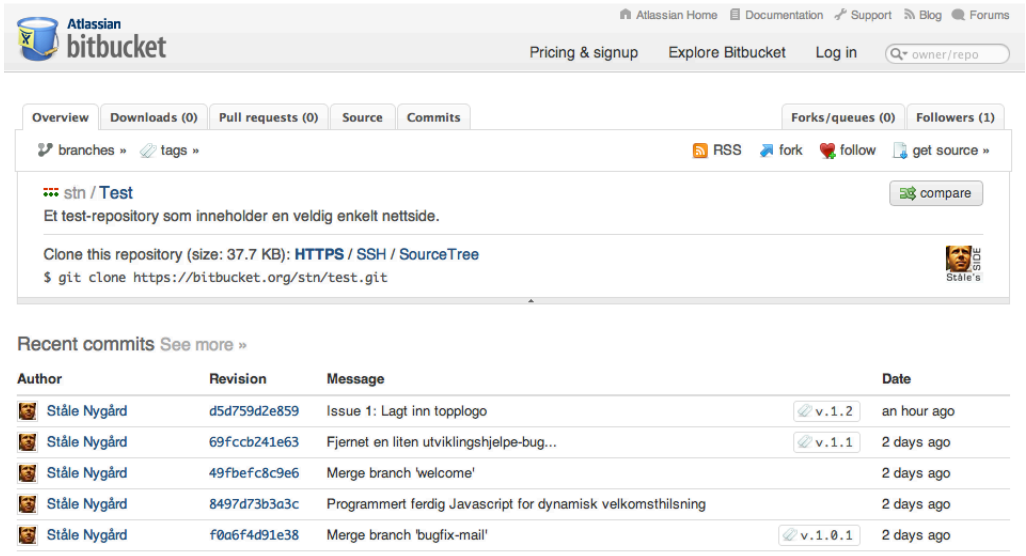
- Klikk ”Resolve”-knappen til høyre for tittelen



- Denne saken er nå ”løst”, mail-notification sendes ut, og saken fjernes fra standardlisten over issues

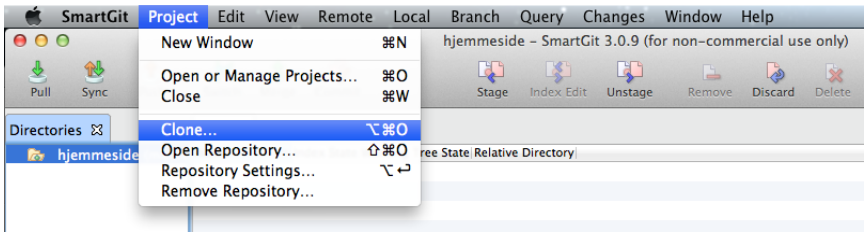


- Klone eksisterende prosjekt:
  - Hjemmesidenprosjektet som er brukt som eksempel er offentlig tilgjengelig – åpne nettsiden: https://bitbucket.org/stn/test

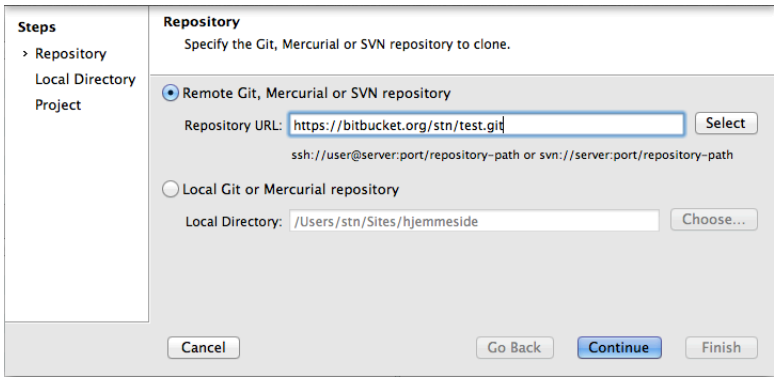


  - Under ”Clone this repository” finner du https-linken til repository’et, merk og kopier adressen (https://bitbucket.org/stn/test.git)
  - Åpne/bytt til SmartGit

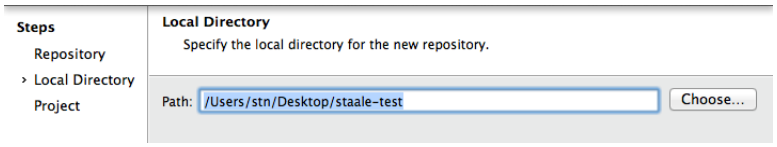
- Project > Clone…



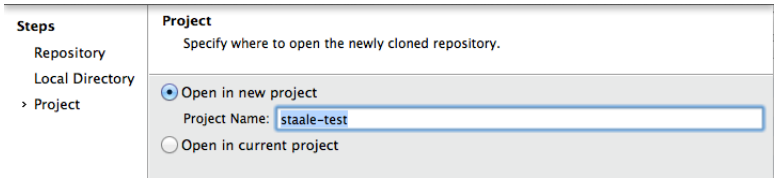
- I neste vindu velger du ”Remote Git, Mercury or SVN repository”, og skriver/limer inn https-adressen som Repository URL



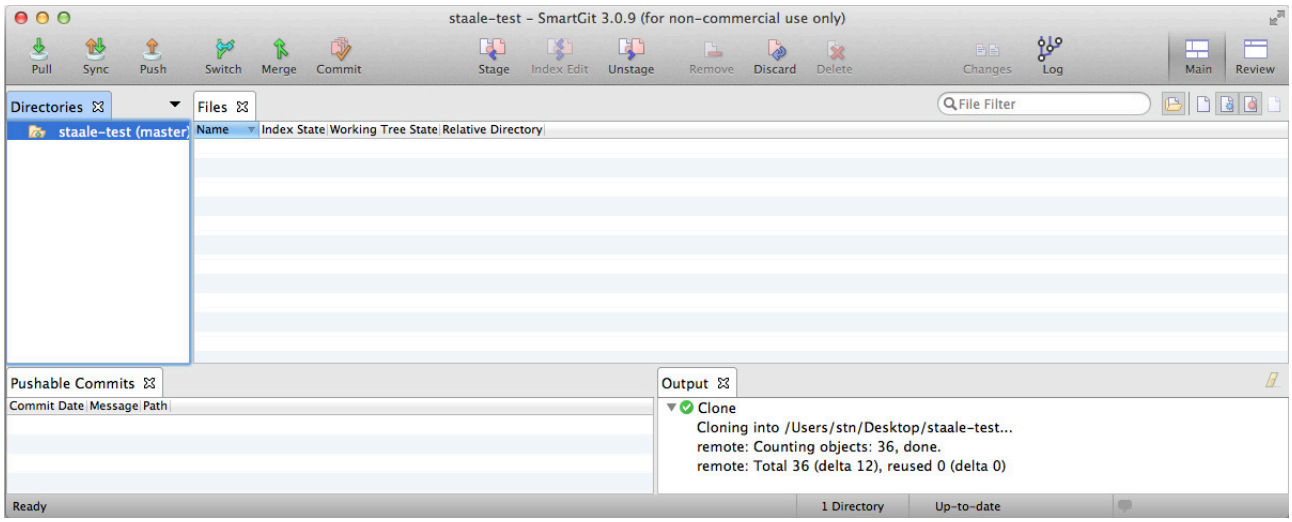
- Bekreft med ”Continue”
- I neste vindu velger du en fornuftig plassering på lokal datamaskin for lagring av arbeidskopien av Git-prosjektet (”Choose…” – bruk eventuelt ”New Folder”-knappen i Choose-vinduet for å opprette ny mappe)



- Bekreft med ”Continue”
- I neste vindu velger du ”Open in new project”



- Bekreft med ”Finish” – du har nå hentet en lokal arbeidskopi av Git-prosjektet og kan jobbe videre lokalt med både filer og versjonsdatabase (merk: du vil ikke ha rettigheter til å push’e de lokale endringene til origin/server)

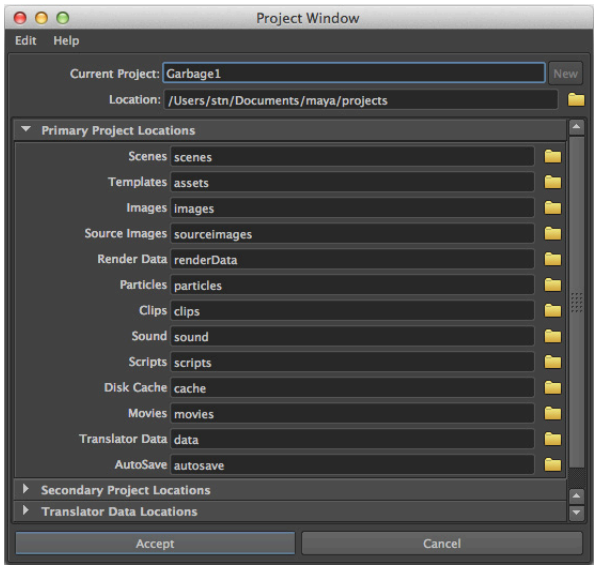


6 Eksempel – versjonskontroll Maya

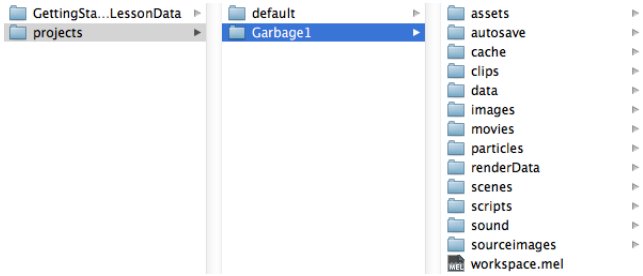
I dette eksemplet skal du bruke 3D-modelleringsverktøyet Maya for å lage noen enkle scener, samtidig som du gjør versjonskontroll på Maya-prosjektet med SmartGit.

Eksemplet viser hvordan du oppretter et nytt Git-repository i SmartGit for et Maya-prosjekt, hvordan du oppdaterer versjonsdatabasen (**Stage**, **Commit**), hvordan du leser versjonshistorikken (**Log**), hvordan du får Git til å ignorere versjonshåndtering av en mappe (**.gitignore**), og hvordan du henter ut en gammel versjon av en Maya-scene og lagrer den med et nytt filnavn i aktiv versjon av prosjektet (**Save As**).

- Åpne Maya og opprett et nytt Maya-prosjekt:
  - File > Project Window…
  - Klikk ”New”
  - Velg en fornuftig ”Location”
  - Gi prosjektet et fornuftig navn (”Current Project”) og bekreft med ”Accept”

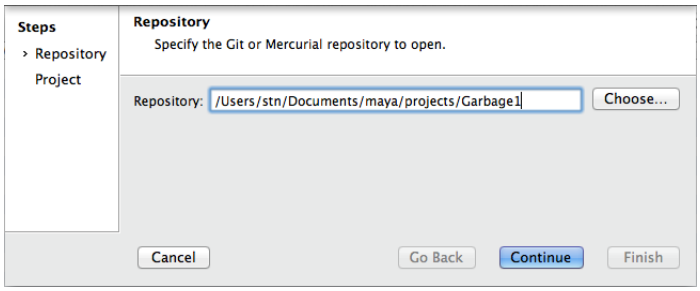


  - Det er nå opprettet et sett med undermapper (”scenes”, ”assets”, osv.) under rota på Maya-prosjektet (i dette eksemplet heter rotmappa ”Garbage1”)

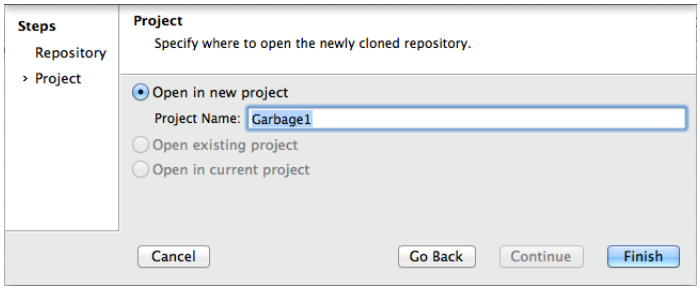


- Åpne SmartGit og initier et nytt Git-repository for Maya-prosjektet:
  - Project > New Window
  - Merk av ”Open an existing local or create a new repository” og klikk ”OK”

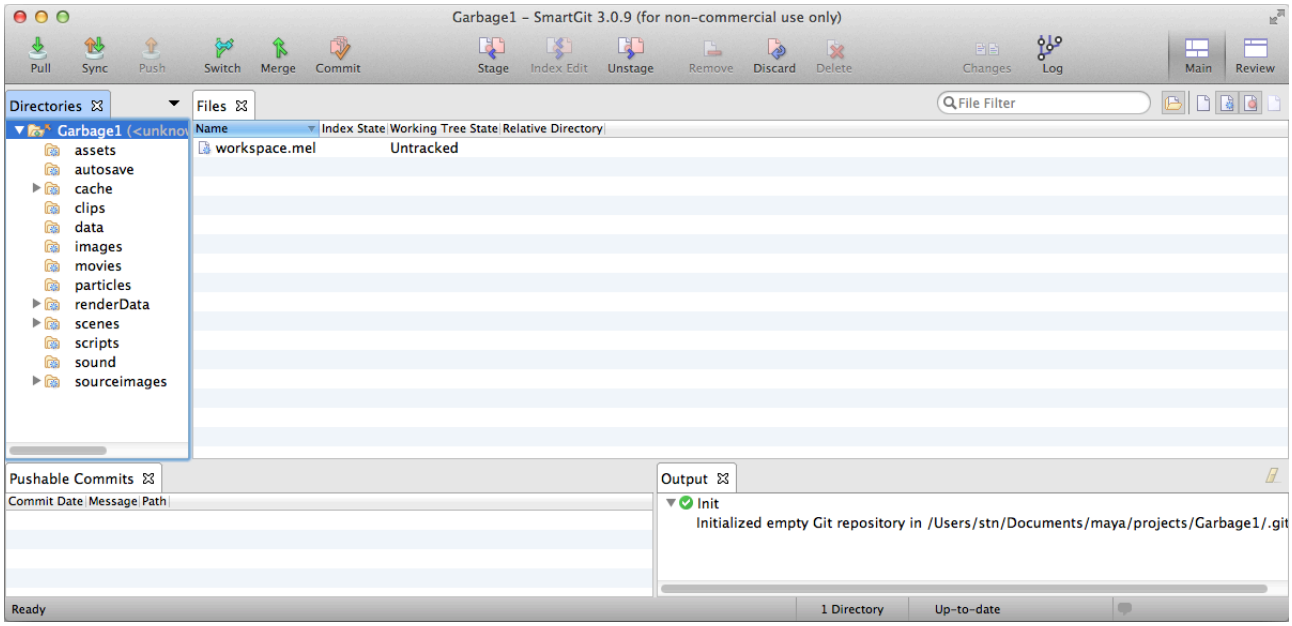
- I neste vindu browse deg fram til (Choose…) rota på Maya-prosjektet



- Bekreft med ”Continue”
- Velg ”Git” i neste vindu ved spørsmål om hva slags repository som skal initieres
- Velg ”Open in new project” i neste vindu, og velg et fornuftig ”Project Name”

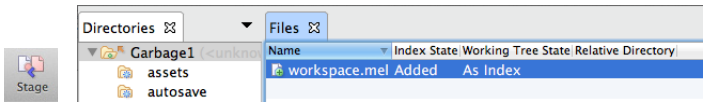


- Bekreft med ”Finish” – et nytt Git-repository er nå opprettet; klar til håndtere versjonskontroll på alle filer som lagres i mappestrukturen til Maya-prosjektet

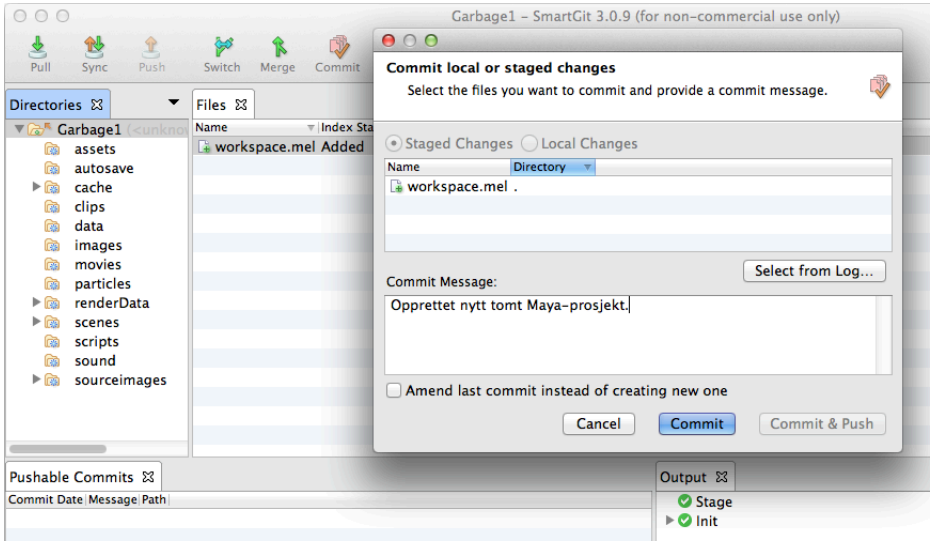


- Commit første versjon av prosjektet:
  - Velg rotmappa (i dette eksemplet ”Garbage1”) i Directories-paletten til venstre

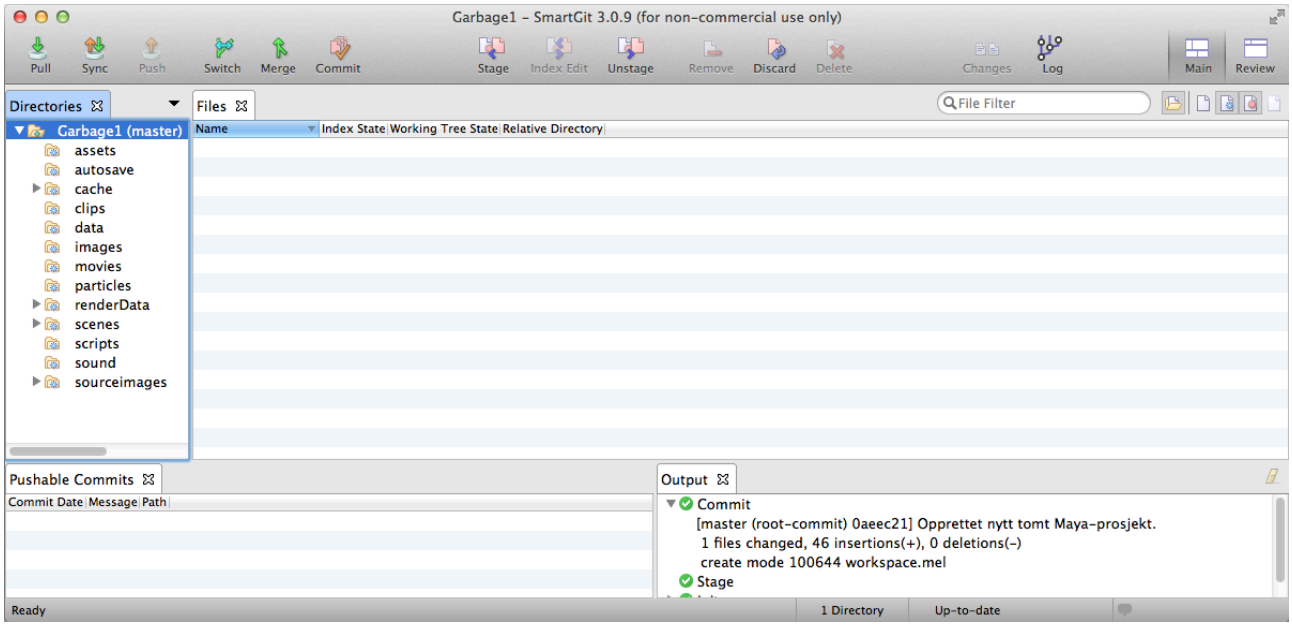
- Marker filen ”workspace.mel” i Files-paletten, og klikk ”Stage” på verktøylinjen (”workspace.mel” er den eneste filen som automatisk blir opprettet i et helt nytt Maya-prosjekt)



- Klikk ”Commit” på verktøylinjen
- Skriv inn en beskrivende Commit-kommentar i neste vindu

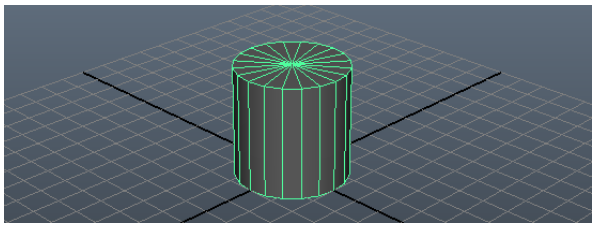


- Bekreft med ”Commit” – første ”shapshot” av Maya-prosjektet er nå lagt inn i Git-repositoryet

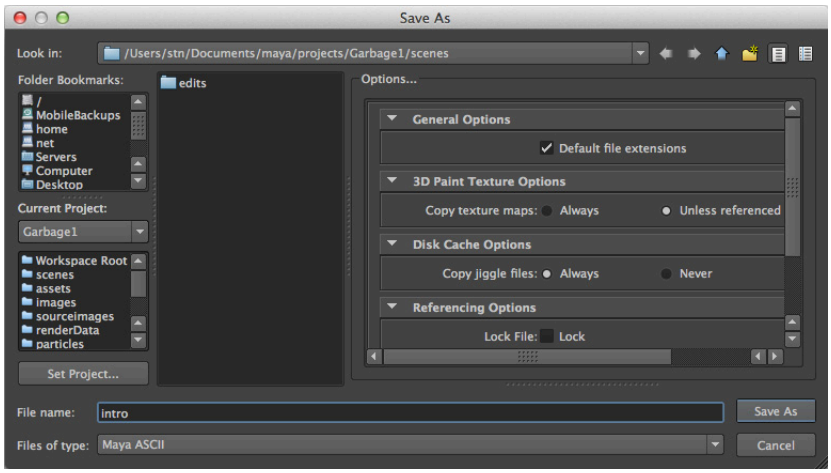


- Lag en ny Maya-scene, og commit til Git-repository:
  - Bytt over til Maya-programmet (den beste arbeidsflyten er å ha både Maya og SmartGit åpne samtidig)

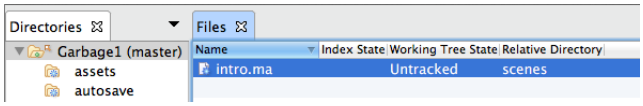
- Opprett eventuelt en ny scene (File > New Scene) og lag en enkel 3D-modell



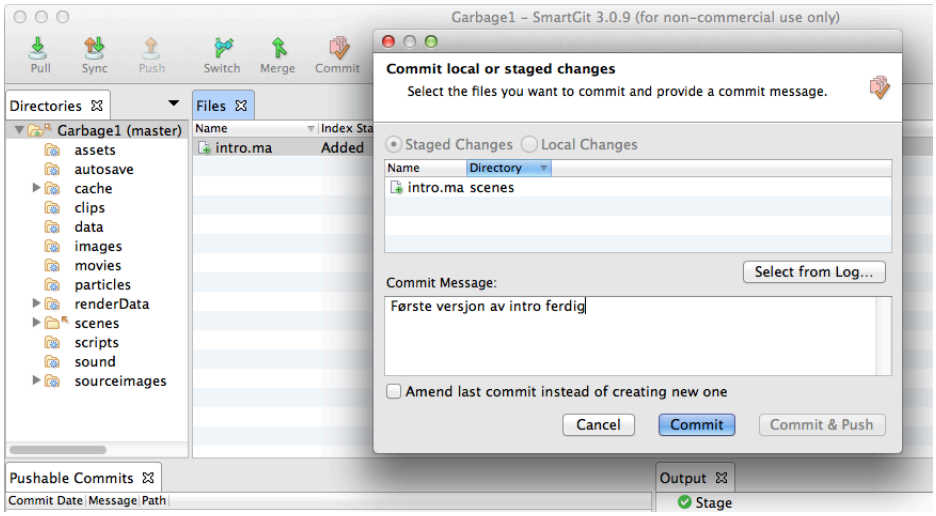
- Lagre scenen (File > Save Scene As…) med et fornuftig navn (File name), med ”Maya ASCII” som Files of type og i standard mappe (Look in: …/scenes) – bekreft med ”Save As”



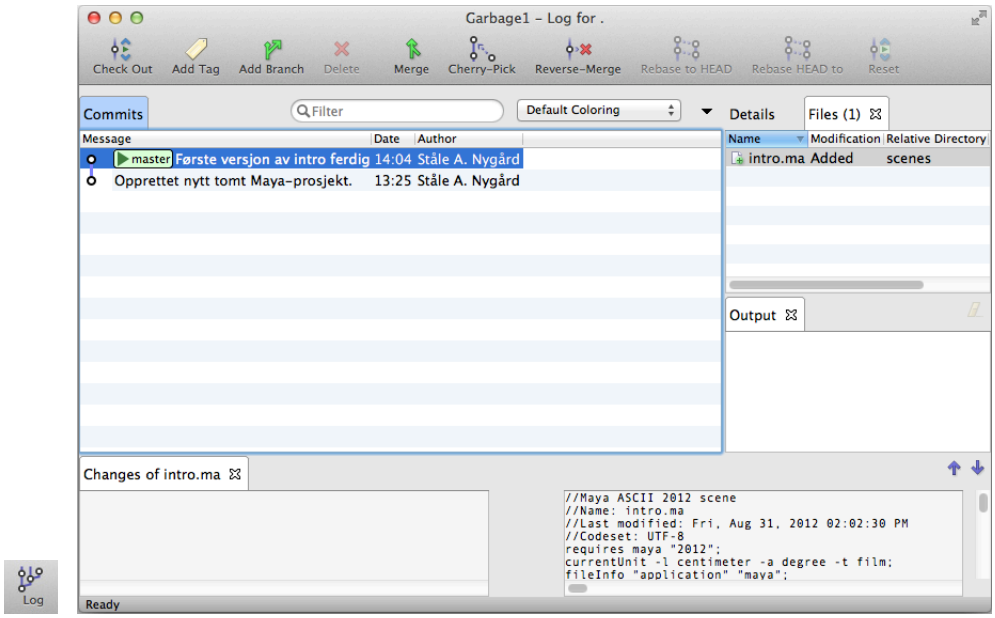
- Bytt til SmartGit, klikk rotmappa i Directories-paletten til venstre, og marker den nye scene-filen i Files-paletten til høyre (Git har enda ikke fått ”beskjed” om å track’e endringer på denne filen; dvs. den er ”Untracked”)



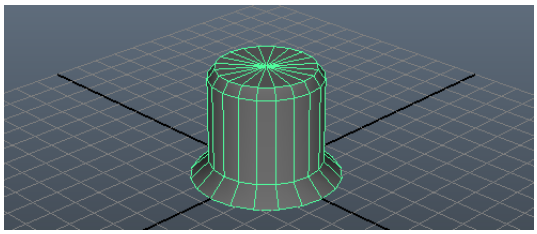
- Klikk ”Stage” og deretter ”Commit” for å legge inn den nye scenen i Git-repositoryet



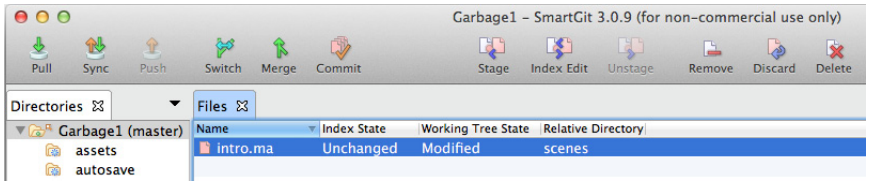
- Andre ”snapshot” av Maya-prosjektet er nå tatt – klikk først rotmappa i Directories-paletten, og deretter”Log” på verktøylinjen for å se en graf over commit-historikken



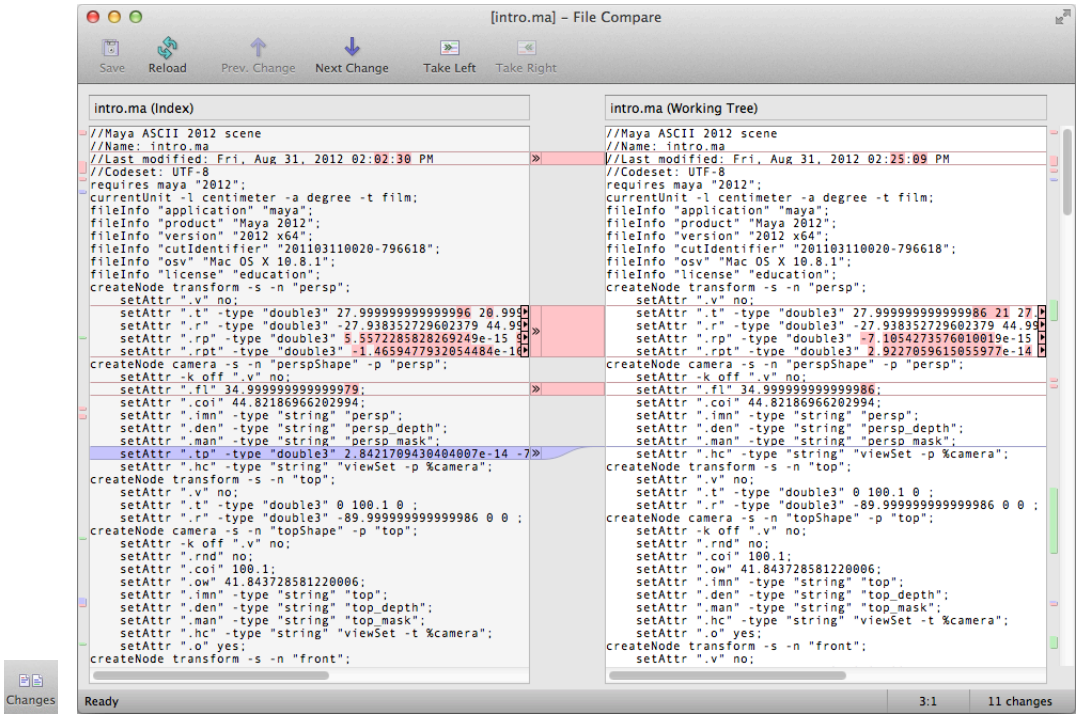
- Gjør endringer i Maya-scene, og commit til Git-repository:
  - Bytt til Maya, og gjør noen små enderinger i den eksisterende scenen



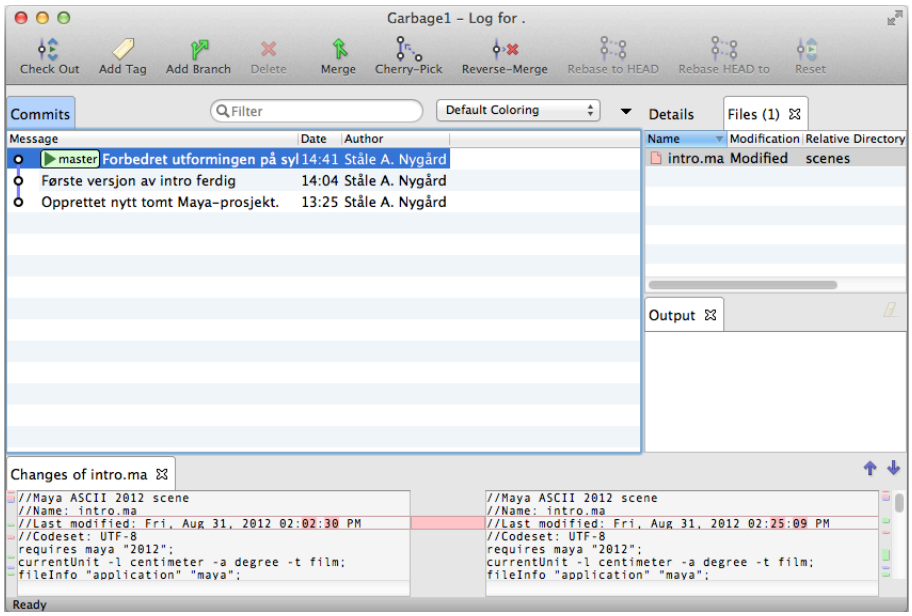
  - Lagre scenen (File > Save Scene)
  - Bytt til SmartGit og klikk eventuelt på rotmappa til Maya-prosjektet i Directories-paletten til venstre – den endrede scene-filen skal nå komme opp i Files-paletten som ”Modified”



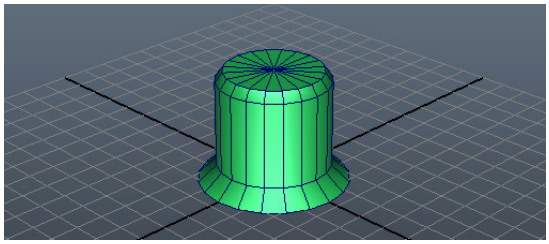
  - Hvis du velger filen i Files-paletten og klikker ”Changes” på verktøylinjen vil du kunne se hvilke forandringer som er gjort på den sist lagrede versjonen av filen i forhold til siste versjon som ble ”commit’et” (i en ASCII Maya-scene er dette rimelig ”kryptisk”, men i annen type kildekode kan denne muligheten være veldig snedig)



- Lukk ”File Compare”-vinduet, og gjør et nytt ”snapshot” av Maya-prosjektet (Stage, Commit)

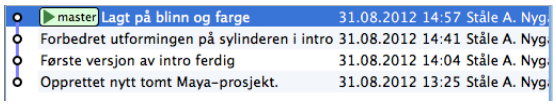


- Bytt til Maya og legg på litt enkel texture

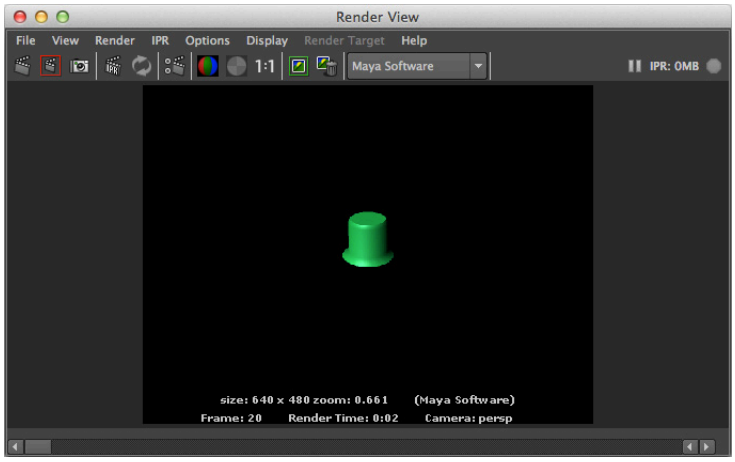


- Bytt til SmartGit og gjør ny commit

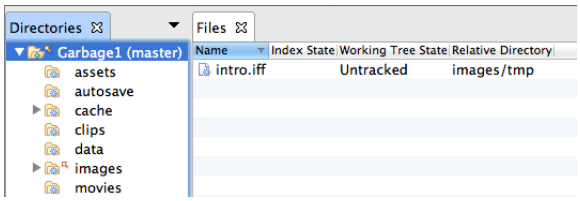
- Ta en titt på versjonshistorikken igjen (”Log”)



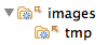
- Legg inn Git-ignore på tmp-mappe:
  - Bytt til Maya-prosjektet igjen, og gjør en Render av Maya-scene’n (for eksempel via Render > Render Current Frame)



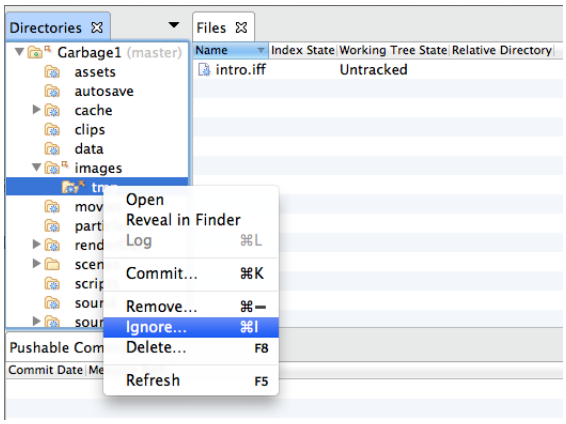
  - Bytt til SmartGit – velg rotmappa i Directories-paletten til venstre for se alle endringene i Git-prosjektet



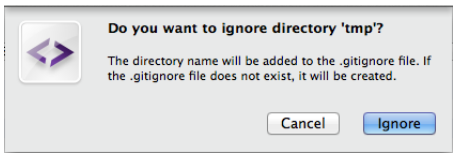
  - Vi ser at det i render’en har blitt generert en midlertidig fil intro.iff i mappen tmp som igjen ligger i mappen images – å ta versjonskontroll av denne mappen (images/tmp/) er ikke interessant for oss, så vi ønsker at Git-systemet skal ignorere denne mappen i sin versjonskontroll
  - Klikk pilen foran ”images”-mappen i Directories-paletten for å se undermappene



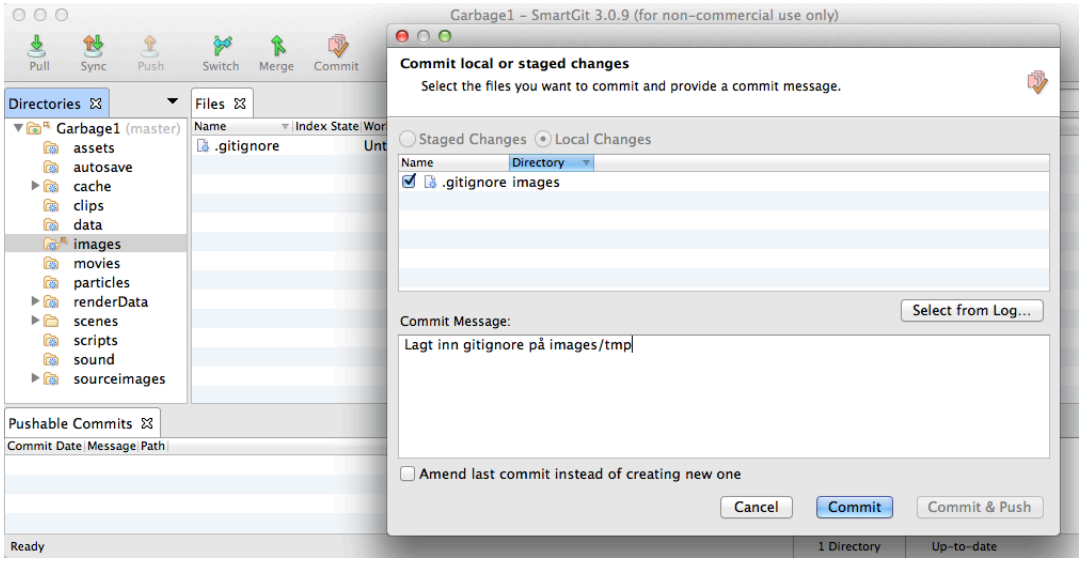
- Høyreklikk på ”tmp”-mappen og velg ”Ignore…” i menyen



- Bekreft med ”Ignore” i neste vindu

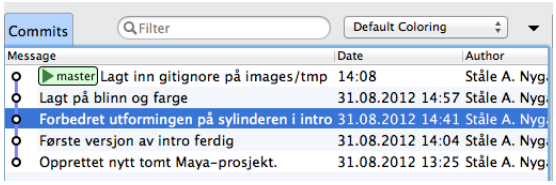


- Det har nå blitt opprettet en fil ”.gitignore” i images-mappen som forteller Git-systemet at undermappen ”tmp” skal ignoreres – gitignore-filen bør være en del av versjonsdatabasen så gjør en commit slik at den legges til (Stage, Commit)

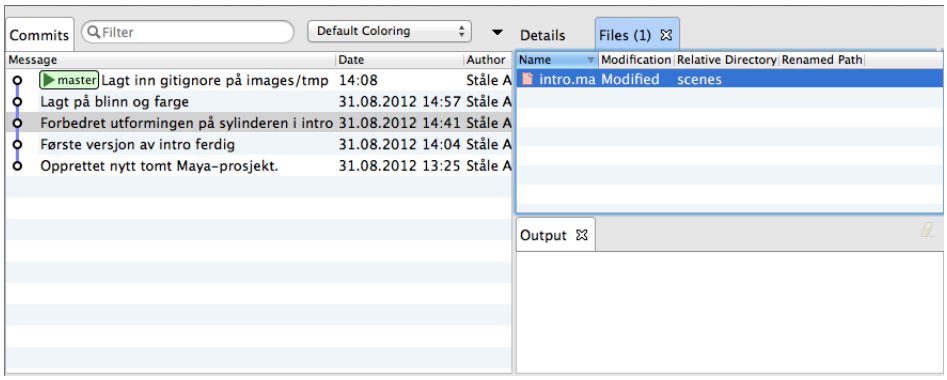


- Ta kopi av Maya-scene i tidligere versjon:
  - Vi tenker oss nå at du skal lage en ny Maya-scene med utgangspunkt i en Maya-scene slik den så ut i en tidligere versjon – det vil da være enklest å finne riktig versjon av filen i loggen og lagre den med et annet navn i stedet for å sjekke ut hele prosjektet til aktuell versjon
  - Bytt til SmartGit, og velg rotmappa til Git-prosjektet i Directories-palettene til venstre
  - Klikk ”Log” på verktøylinjen for å få fram versjonshistorikken

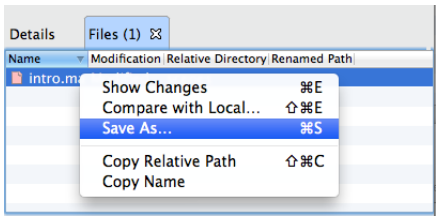
- Klikk den linjen i Commits-paletten hvor du vet at ønsket versjon av Maya-scene ble commit'et



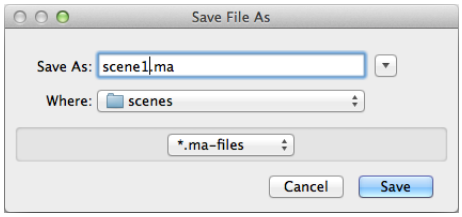
- Klikk "Files"-fliken til høyre for Commits-paletten slik at den blir aktivert



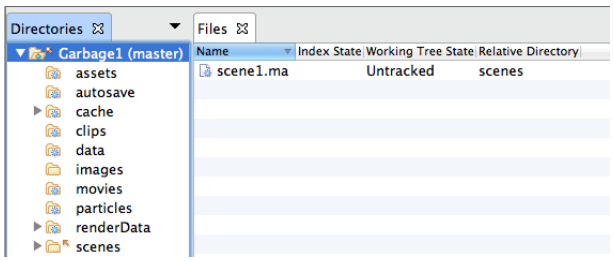
- Høyreklikk filnavnet (i dette eksemplet "intro.ma") og velg "Save As…" i menyen



- Gi den nye scenen et passende navn (i vårt eksempel "scene1.ma") og lagre den i riktig mappe ("scenes")

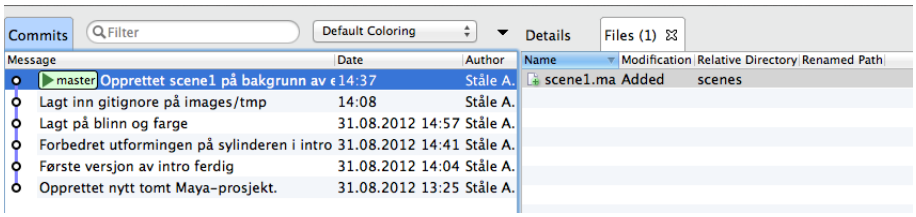


- Lukk Log-vinduet – den nye Maya-scenen skal nå komme opp i Files-paletten som Untracked (det kan hende du må velge View > Refresh for at SmartGit skal gjøre en oppdatering)

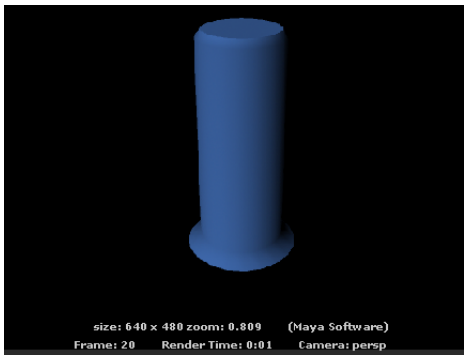


- Legg inn den nye Maya-scenen i versjonsdatabasen (Stage, Commit)

- Ta en titt på versjonshistorikken (”Log”)



- Bytt til Maya og åpne den nye Maya-scenen (”scene1.ma”) for å kontrollere at du har fått riktig versjon
- Gjør noen enkle endringer i scenen, legg på litt enkel texture, gjør en render for å sjekke at Git ignorerer nytt innhold i images/tmp-mappen



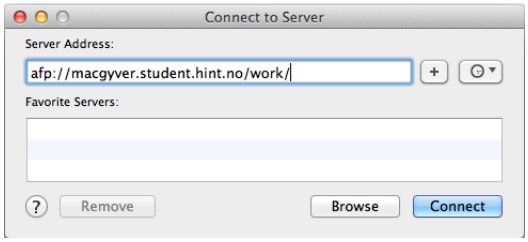
- Gjør til slutt en commit for å oppdatere Git-repository’et

7 Eksempel – nettverksshare som Git-server

I dette eksemplet skal du koble til et delt nettverksshare på Mac-lab'en, og bruke dette som en Git-server for å host'e Maya-prosjektet som ble opprettet i forrige kapittel.

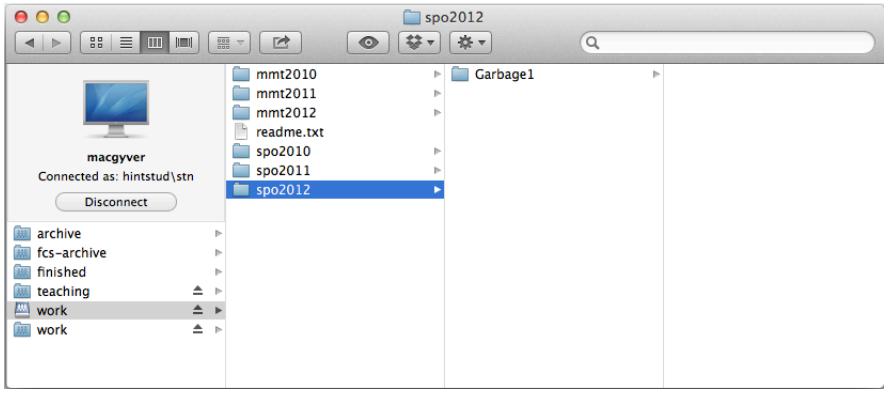
Eksemplet viser hvordan du oppretter et nytt tomt repository på serveren vha. Finder og git-kommandoer i Terminal-programmet (**git --bare init**), hvordan du importerer et lokalt repository til et tomt repository på et nettverksshare (**Add Remote**, **Push**), og hvordan du oppdaterer lokal versjon for å sikre at den er synkronisert med remote/server (**Pull**, **Sync**).

- Koble til nettverksshare:
  - I dette eksemplet viser vi hvordan vi kobler til et nettverksshare på Mac-lab'en, men tilnærmet samme fremgangsmåte kan benyttes på PC-lab'ene; men da må man koble til et område på en annen server enn "MacGyver"
  - Åpne Finder
  - Go > Connect to Server…
  - Skriv inn Server Address: afp://macgyver.student.hint.no/work/



  - Bekreft med "Connect" (det er mulig du må skrive inn brukernavn og passord nå – i så fall det samme som på datasalen ellers)

- Opprett et nytt tomt Git-repository på nettverksshare'et:
  - Bruk Finder og opprett en ny mappe (File > New Folder) for et nytt Git-prosjekt på MacGyver på en fornuftig plass (i dette eksemplet i mappen "spo2012/Garbage1/" (merk at alle i utgangspunktet har lese/skrive-rettigheter på work-share'et)

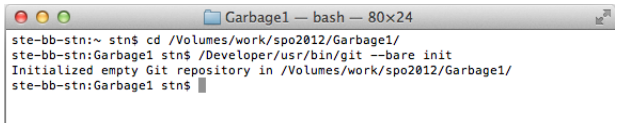


  - Åpne Terminal-programmet (Applications > Utilities > Terminal)
  - Skriv inn følgende 2 kommandoer (hver kommando etterfølges av et trykk på Enter-tasten, /path/to/project/ må erstattes med din mappesti – for eksempel /spo2010/spillprosjekt10/ - eller /spo2012/Garbage1/ som er benyttet i vårt eksempel):

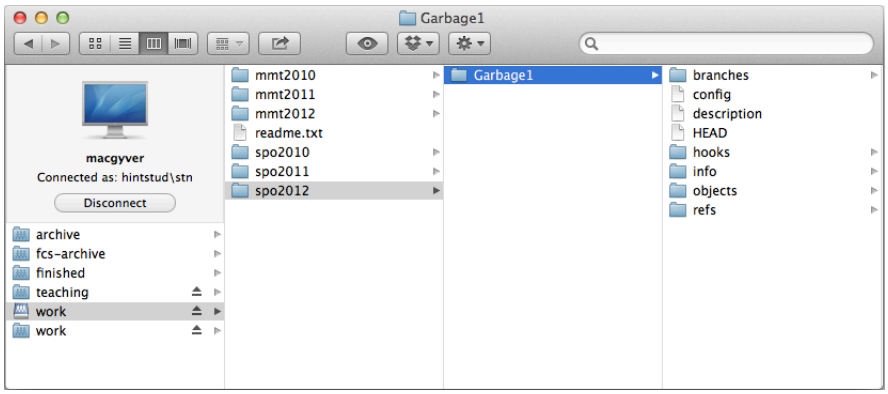
```
$ cd /Volumes/work/path/to/project/
$ git --bare init
```

Hvis den siste kommandoen ikke fungerer kan du prøve å oppgi absolutt sti til git-kommandoen i stedet:

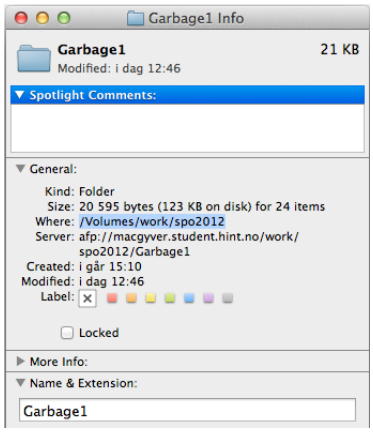
```
$ /Developer/usr/bin/git --bare init
```

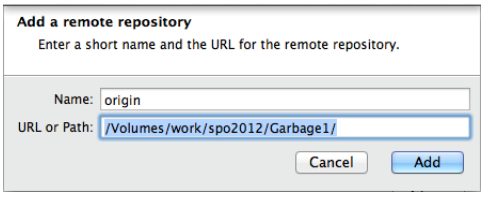


- Bytt til Finder, åpne mappen du nettopp laget, og sjekk at Git-prosjektet har blitt opprettet (det har blitt opprettet diverse mapper og filer som vil inneholde versjonsdatabasen)



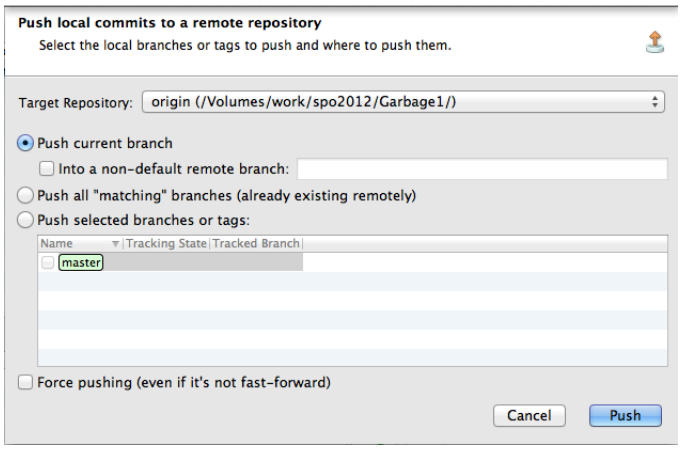
- Importer et lokalt repository til serveren (nettverksshare'et):
  - Åpne/bytt til SmartGit
  - Åpne eventuelt et lokalt repository i et eget vindu (i dette eksemplet benyttes det Maya-prosjektet som ble opprettet i kapittel 0)
  - I SmartGit velg: Remote > Manage Remotes
  - I neste vindu klikk "Add…"
  - I neste vindu lar du det stå Name: "origin", og skriver inn hele banen/stien til mappen med det tomme Git-prosjektet du nettopp opprettet – i vårt eksempel /Volumes/work/spo2012/Garbage1/ (tips: hvis du høyreklikker på mappen i Finder og velger "Get Info" i menyen finner du stien til mappen under "Where")



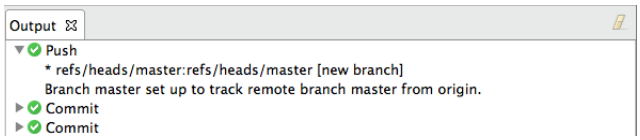


  - Bekreft med "Add"
  - I neste vindu ser du at origin har blitt definert som en et nytt "remote" repository – klikk"Close" for å lukke vinduet
  - Klikk rotmappa til Git-prosjektet i Directories-paletten til venstre
  - Importer det lokale repository'et til origin-repository'et: ved første push kan det være greit å velge en "advanced push": Remote > Push Advanced…

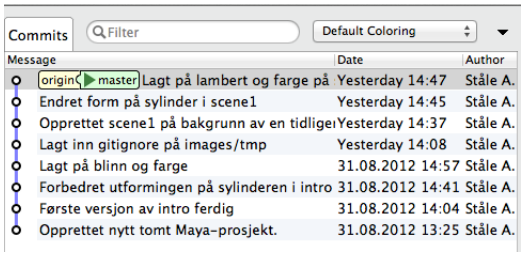
- I neste vindu velger du Target Repository: origin – og velger ”Push selected branches or tags” hvis du skal importere flere branch’er og tag’er til origin-repository’et (i dette tilfellet er det kun en master branch så da vil standardvalget ”Push current branch” gjøre samme jobben)



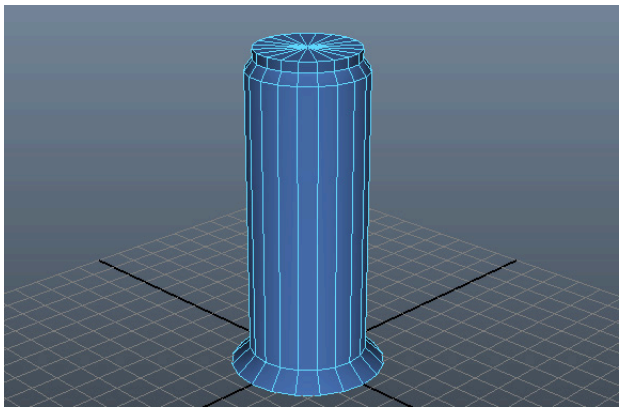
- Bekreft med ”Push”
- I neste vindu klikker du ”Configure” for å sette opp håndtering av synkronisering mellom remote (origin) og lokal versjon
- Hele Git-prosjektet, inkludert filer og versjonsdatabase, kopieres/importeres nå forhåpentligvis over til serveren (dette kan ta litt tid) – klart til å deles med andre



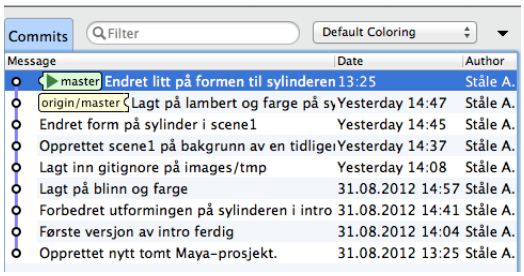
- Åpne versjonshistorikken (”Log”) for å sjekke at lokal og remote (origin) repository er synkronisert



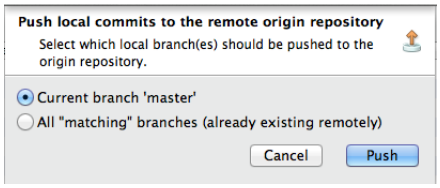
- Gjør noen små forandringer i Maya-prosjektet



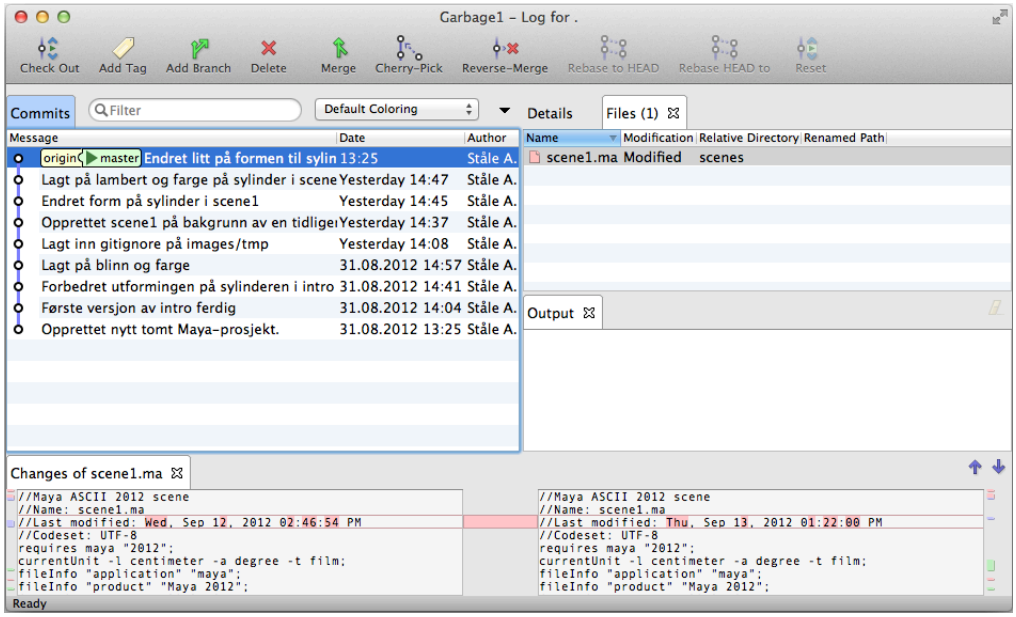
- Bytt til SmartGit og gjør en ny commit for å versjonere forandringene
- Ta en titt på versjonshistorikken igjen (”Log”) – du vil nå se at lokal og remote (origin) versjon ikke er synkronisert



- Lukk Log-vinduet, og oppdater serveren (origin) ved å klikke ”Push” på verktøylinjen

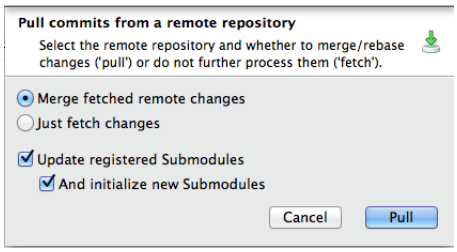


- I neste vindu velger du ”Current branch ’master’” og bekrefter med ”Push”
- Serveren er nå oppdatert (se ”Log”-vinduet)

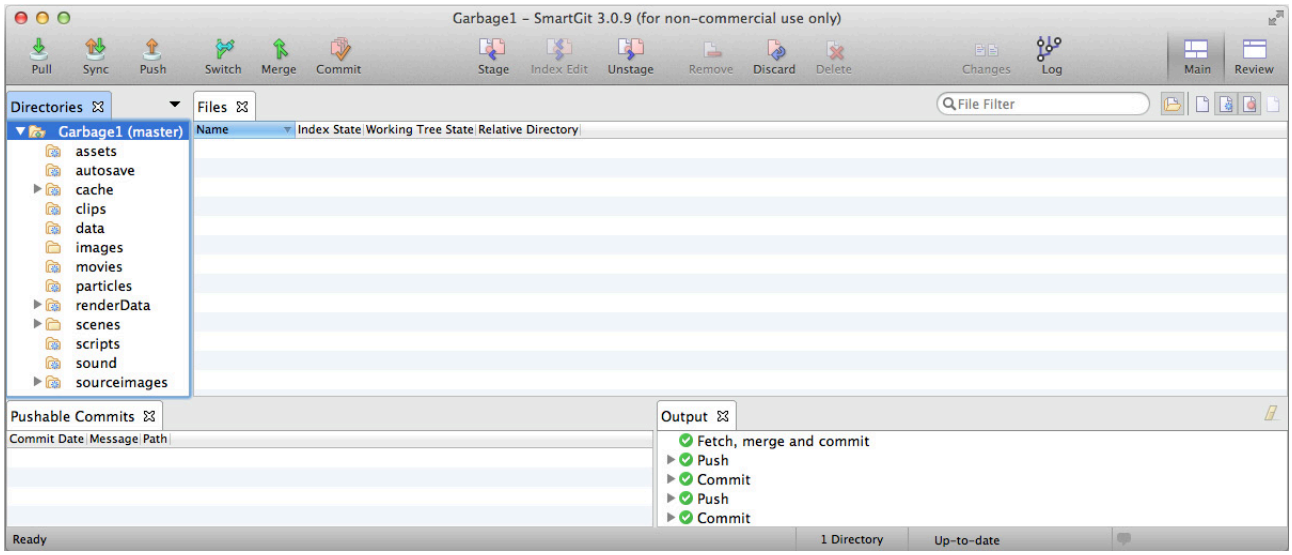


- Hvis det er flere som samarbeider om å benytte samme remote repository; med jevnlige oppdateringer av serveren, kan man innimellom sjekke om det er nødvendig å oppdatere sin lokale arbeidskopi ved å klikke ”Pull” eller ”Sync” – klikk rotmappa til Git-prosjektet i Directories-paletten og klikk ”Pull”

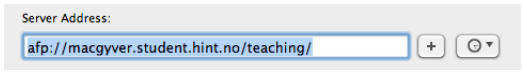
- I neste vindu kan du la du standardvalgene stå for å eventuelt ”flette inn” (merge) forandringer som er gjort på serveren i forhold til lokal arbeidskopi



- Bekreft med ”Pull”

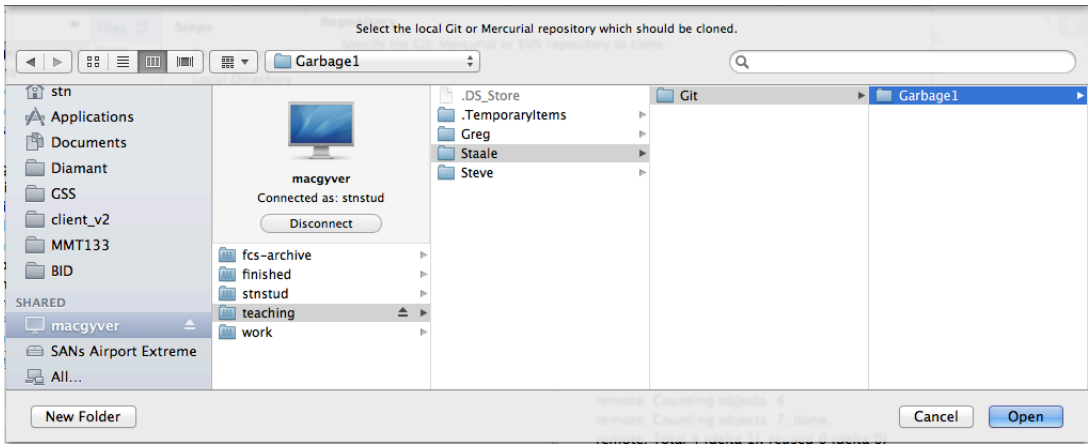


- Klone eksisterende prosjekt på server:
  - Maya-prosjektet som er brukt som eksempel er tilgjengelig på MacGyver (read only for studenter) på ”teaching”-share’et – i Finder: Go > Connect to Server…
  - I neste vindu skriver du inn Server Address: afp://macgyver.student.hint.no/teaching/

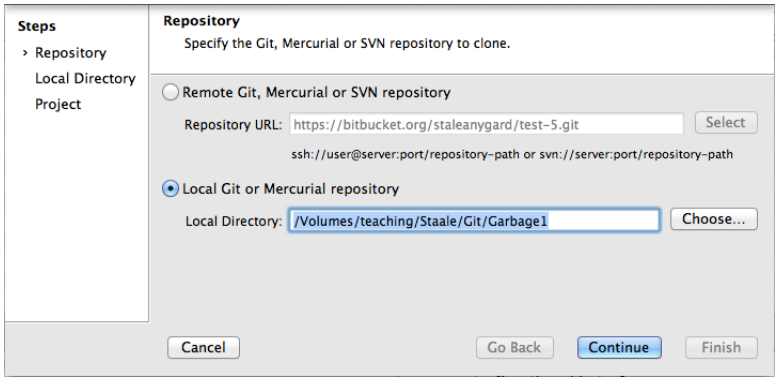


  - Bekreft med ”Connect”
  - Åpne/bytt til SmartGit
  - I SmartGit: Project > Clone…
  - I neste vindu velger du ”Local Git or Mercurial repository”, og klikker ”Choose…”

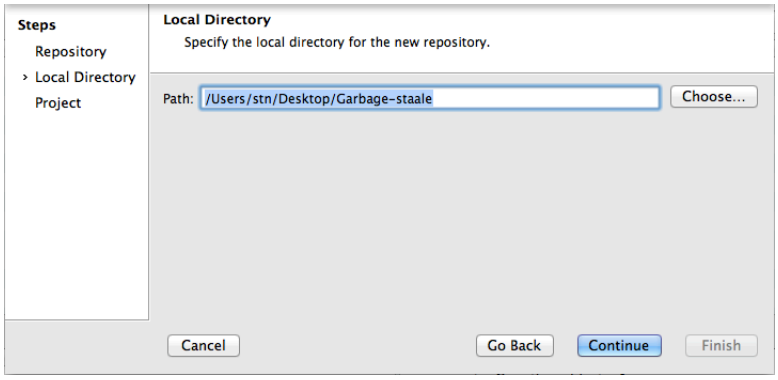
- Browse deg fram til mappen /Staale/Git/Garbage1/ på teaching-share'et



- Bekreft med "Open"

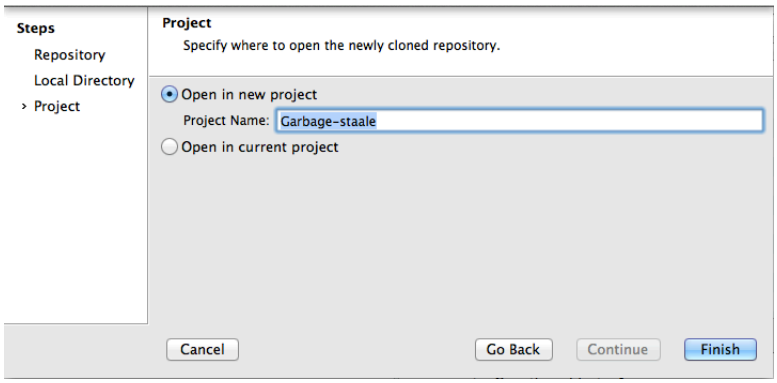


- I Local Directory skal det nå stå "/Volumes/teaching/Staale/Git/Garbage1/" – bekreft med "Continue"
- I neste vindu velger du en lokal plassering av din arbeidskopi av Git-prosjektet – klikk "Choose…" og velg en egnet plassering

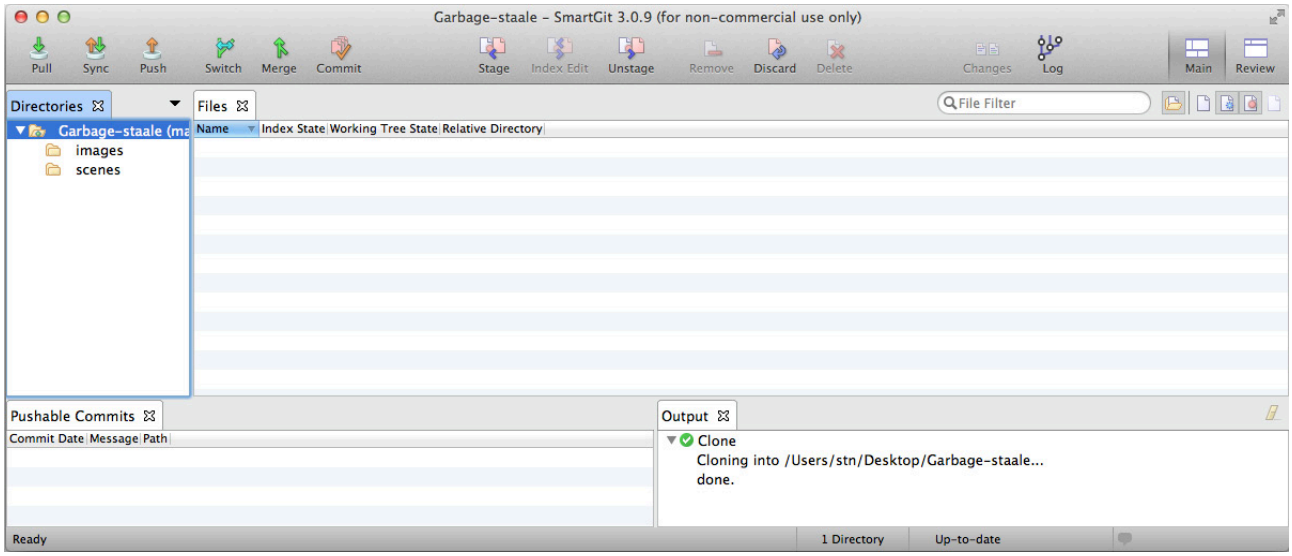


- Bekreft med "Continue"

- I neste vindu velger du ”Open in new project”, og gir prosjektet et egnet Project Name



- Bekreft med ”Finish” – du har nå hentet en lokal arbeidskopi av Maya-prosjektet og kan jobbe videre lokalt både med filer og versjonsdata – i Maya: File > Set Project… (merk: du har i dette tilfellet ikke rettigheter til å push’e endringer til remote/origin)



8 Referanser

- Chacon, S. (2009). *Pro Git*, Apress.
- Syntevo (2012). *SmartGit Quickstart Guide*, syntevo GmbH, www.syntevo.com.
- Atlassian (2012). BitBucket Documentation, Atlassian, på URL: https://confluence.atlassian.com/display/BITBUCKET/bitbucket+Documentation+Home.