# Git mini-guide

Kim Lindberg Schwaner
kschw10@student.sdu.dk

4. marts 2012

## Indhold

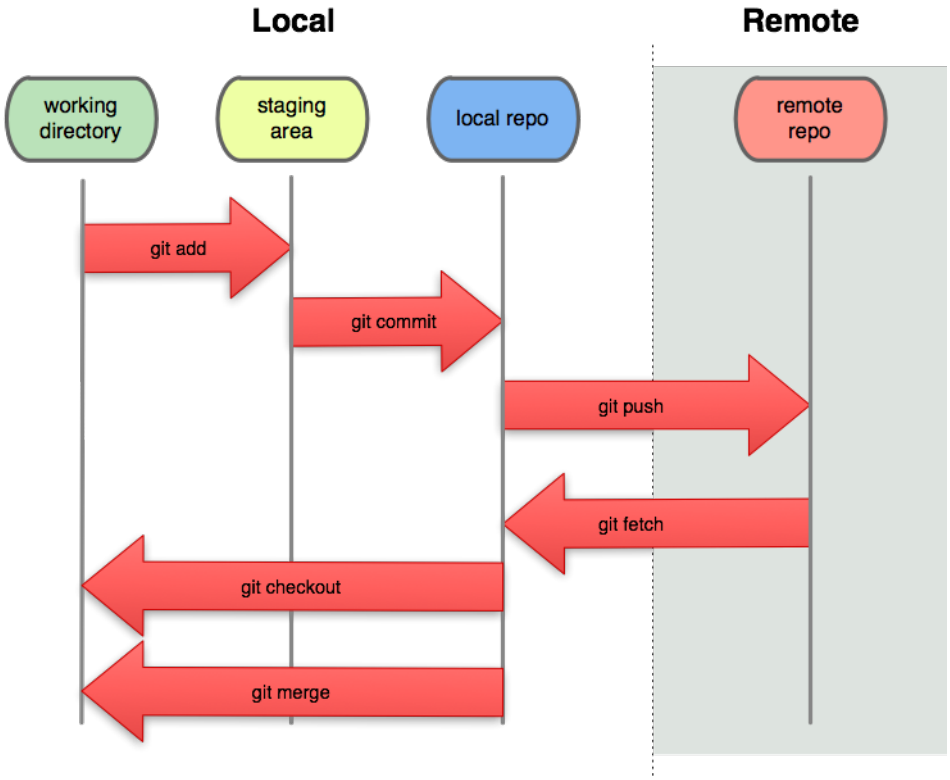
# 1 Hvordan Git virker

I korte træk fungerer Git ved at gemme *snapshots* af det projekt man arbejder på. Som bruger kan man sammenligne forskellige snapshots taget på forskellige tidspunkter, gå frem og tilbage mellem dem, tage dele ud af dem osv.

Har man oprettet en ny fil i et projekt, man arbejder på, siges den at være i *working directory*. Som udgangspunkt er filen ikke *tracked* af Git. Det bliver den først i det man bruger `add` for at tilføje den til *index* aka. *staging area*. Index er en liste over filer, der vil blive *committed* næste gang `commit` bruges. Der tages et snapshot af projektet når ændringer committes og det snapshot kan efterfølgende findes i *repository*.

For at arbejde ud fra ét bestemt snapshot kan man checke det ud fra repository med kommandoen `checkout`. `merge` kan flette ændringer sammen.

Kommandoerne `push` og `fetch` bruges til at synkronisere et lokalt repository med et andet, fjernt repository.




Figur 1: Fil-livscyklus. Fra `www.progit.org/book`

# 2 Opsætning af Git

Git vil gerne kende brugerens navn og email adresse:

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "your_email@youremail.com"
```

Nogle (Github f. eks.) kræver SSH. SSH nøglen *skal* ligge i `%HOME%\.ssh` på Windows eller `~/.ssh` på Unix. Oprettes med:

```
$ cd ~/.ssh
$ ssh-keygen -t rsa -C "your_email@youremail.com"
```

Tryk enter når der spørges om filnavn; så bruges default-navnet. Den SSH nøgle, som bliver lavet, skal registreres på Github under "Account Settings" → "SSH Public Keys" → "Add another public key".

Forbindelse til Github kan testes med:

```
$ ssh -T git@github.com
```

# 3 Nyt repository

For at oprette et nyt repository på en lokal maskine, kan en af to kommandoer bruges:

- `clone`
- `init`

## 3.1 `clone`

Hvis man vil kopiere et repository, der i forvejen findes (f. eks. på Github), er det letteste at bruge `clone`:

```
$ git clone protocol://address.to/repository.git
Cloning into repository...
$ cd repository
$ ls -a
.  ..  .git
```

## 3.2 `init`

Vil man oprette et *helt* nyt repository skal man bruge `init`:

```
$ mkdir new-git-repo
$ cd new-git-repo
$ git init
Initialized empty Git repository in new-git-repo/.git/
$ ls -a
.  ..  .git
```

# 4 Snapshotting

For at gemme *snapshots* af sit projekt med Git har man brug for at kende kommandoerne

- `status`
- `add`
- `commit`
- `reset`
- `rm`

## 4.1 `status`

Kommandoen `status` viser status på filer i *working directory* og *staging area*. Desuden giver den også tit gode hints til hvilke kommandoer der skal bruges for at "komme videre".

```
$ touch newprogram.c
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add  file ..." to include in what will be committed)
#
#  newprogram.c
nothing added to commit but untracked files present (use "git add" to track)
```

## 4.2 `add`

For at tilføje filer til *staging area* bruges `add`.

```
$ touch newprogram.c
$ git add newprogram.c
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached  file ..." to unstage)
#
#  new file:   newprogram.c
```

## 4.3 commit

Et *snapshot* kan gemmes med `commit`. Kun ændringer i *staging area* bliver committed.

```
$ touch newprogram.c
$ git add newprogram.c
$ git commit -m "This is a commit message"
[master (root-commit) 6c0e79b] This is a commit message
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newprogram.c
```

Hvis ikke en commit-tekst tilføjes med switchen `-m` spørger git selv efter den ved at åbne den tekst editor, som er specificeret med `config --global core.editor`. Det kan være en fordel hvis man ønsker en mere beskrivende commit-tekst. Ved ikke-trivielle commits bør følgende format[1] tilstræbes:

```
Capitalized, short (50 chars or less) summary

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Write your commit message in the present tense: "Fix bug" and not "Fixed
bug."  This convention matches up with commit messages generated by
commands like git merge and git revert.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet, preceded by a
  single space, with blank lines in between, but conventions vary here

- Use a hanging indent
```

## 4.4 reset

Har man føjet filer til *staging area* men ønsker at *un-stage* dem, skal `reset HEAD -- filnavn` bruges. F. eks:

```
$ touch newprogram1.c
$ touch newprogram2.c
$ git add newprogram*.c
$ git status
# On branch master
# Changes to be committed:
```

[1] Kilde: `http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html`

```
#   (use "git reset HEAD  file ..." to unstage)
#
#  new file:   newprogram1.c
#  new file:   newprogram2.c
#
$ git reset HEAD -- newprogram2.c
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD  file ..." to unstage)
#
#  new file:   newprogram1.c
#
# Untracked files:
#   (use "git add  file ..." to include in what will be committed)
#
#  newprogram2.c
```

En anden meget nyttig måde at bruge `reset` på, er hvis man ønsker at gendanne sit *working directory* til en bestemt commit. Ofte vil det være HEAD (toppen af den *branch* man står i). Hvis working directory skal overskrives er switchen `--hard` nødvendig:

```
$ git reset --hard HEAD
HEAD is now at xxxxxx
```

## 4.5 `rm`

Denne kommando sørger for at smide specificerede filer helt ud af *staging area*. Som udgangspunkt slettes pågældene filer også helt fra disken, med mindre `--cached` switchen tilføjes!

```
$ touch temp-file.c
$ touch program.c
$ git add *.c
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached  file ..." to unstage)
#
#  new file:   program.c
#  new file:   temp-file.c
#
$ git rm temp-file.c
error: 'temp-file.c' has changes staged in the index
(use --cached to keep the file, or -f to force removal)
$ git rm -f temp-file.c
rm 'temp-file.c'
```

# 5 Branching og merging

En Git branch kan ses som pointer, som peger på én bestemt commit i en række af commits. En speciel pointer, HEAD, peger altid på den branch man står i. Skifter man branch med `checkout` flyttes HEAD pointeren. Følgende kommandoer giver mulighed for at manipulere branches:

- `checkout`
- `branch`
- `merge`

## 5.1 branch

Til at oprette, se (`-v` eller `-a`) og slette (`-d`) branches bruges `branch`.

```
$ git branch newbranch
$ git branch anotherbranch
$ git branch -a
  anotherbranch
* master
  newbranch
$ git branch -d newbranch
Deleted branch newbranch (was 87a363c)
$ git branch -a
  anotherbranch
* master
```

## 5.2 checkout

For at skifte mellem branches skal man bruge `checkout`. *Working directory* bliver reset til sidste commit på den branch, der checkes ud.

```
$ git checkout anotherbranch
Switched to branch 'anotherbranch'
```

## 5.3 merge

Kommandoen `merge` kan flette branches sammen i den nuværende branch. Hvis der er konflikter skal disse løses manuelt før et merge kan gennemføres.

```
$ touch code.c
$ git add code.c
$ git commit -m "A commit"
[anotherbranch 2f9a917] A commit
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 code.c
$ git checkout master
Switched to branch 'master'
$ touch program.c
```

```
$ git add program.c
$ git commit -am "Program committed"
# On branch master
nothing to commit (working directory clean)
$ git merge anotherbranch
Updating 87a363c..2f9a917
Fast-forward
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 code.c
```

# 6 Yderligere info

For mere info, se

- `man git`
- `www.progit.org/book`
- `www.gitref.org`