Git till vardags

Matthias Nilsson

27 januari 2017

Introduktion

Introduktion

Planen för i dag:

- Commits
- Grenar
- Samarbete och konflikter
- När allt går fel
- Tips & trix

Några ord om kommandorad vs. GUI.

Använd det som funkar bäst för dig.

Commits

Commits

```
commit c1b8042153cc7ebc459bbf9cad9d61bdc318fb64
Author: Matthias Nilsson <matthias.nilsson@kb.se>
Date:   Thu Jan 26 11:18:08 2017 +0100

    Initial commit
```

Commits

Commits låter oss se vad som ändrats och (förhoppningsvis) varför.

Commits

http://www.commitlogsfromlastnight.com/

Ett bra commit-meddelande förklarar vad som gjorts och varför.

```
En kort (<50 tecken) sammanfattning

Därefter en längre beskrivande text, om det behövs.
Försök hålla radlängden till ~72 tecken.

Imperativ form ("Add", inte "Added") på
sammanfattningen matchar stilen på de meddelanden
som Git genererar.
```

Se `http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html` **för mer information.**

Commits

Varje commit som hamnar i `master` ska vara komplett.

Detta innebär att alla tester går igenom, att koden går att köra utan varningar, osv.

Commits

En bra commit är en logiskt sammanhängande ändring.

Orelaterade ändringar ska ligga i andra commits.
Håll uppstädning separat från logiska ändringar.

Commits

Gits ”staging area” gör det här enkelt för oss.

Commits

Gits ”staging area” gör det här enkelt för oss.

`git add --patch` **och** `git add --interactive` **låter oss välja vad vi vill commit:a.**

Commits

OBS

`git commit -m` **kan slänga bort information.**

T.ex. vid en merge med manuellt fixade konflikter.

Commits

Ibland blir det fel.

Commits

Ibland blir det fel.

Stavfel i commit-meddelandet, glömt att lägga till en fil, testerna går inte igenom, osv.

Commits

`git commit --amend` **låter oss ändra på senaste commit:en.**

(För att ändra på mer finns kommandot `git rebase --interactive`, men det är överkurs just nu.)

Övning 1: Snygga commits

Commits

Historiken i Git är en riktad acyklisk graf, där varje commit är en nod.

(Historiken kan förgrenas, men allt rör sig framåt.)

Commits

Vi kan använda `git log` för att titta på grafen:

```
git log --all --decorate --oneline --graph
```

(`tig`[1] är värt att titta på för den som gillar textbaserade verktyg.)

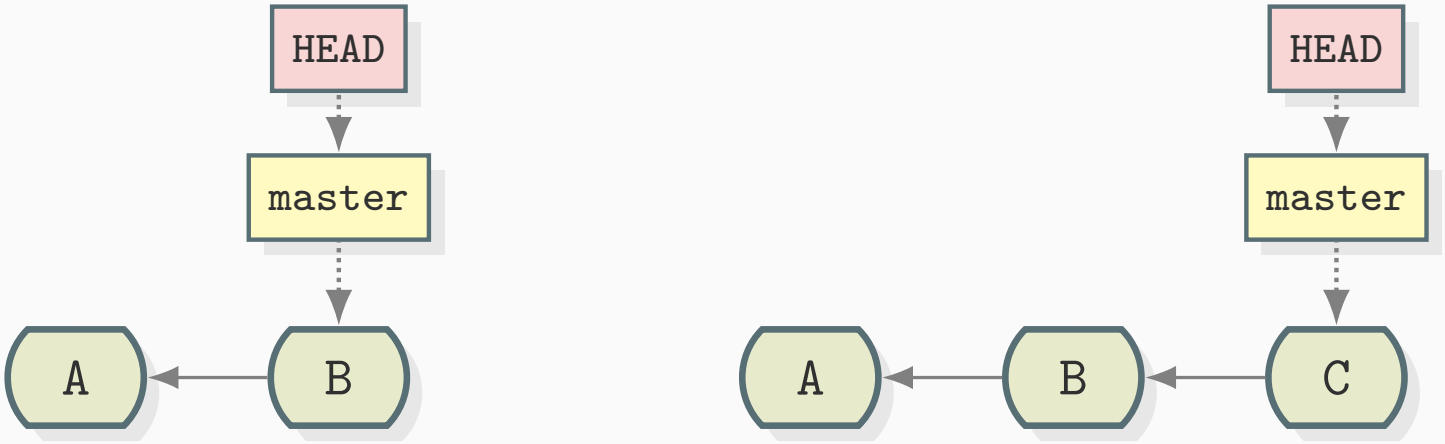
[1] `https://github.com/jonas/tig`

Commits

`HEAD` är den commit vi står på just nu.

`HEAD` ändras t.ex. när vi skapar nya commits eller när vi byter branch.

Commits



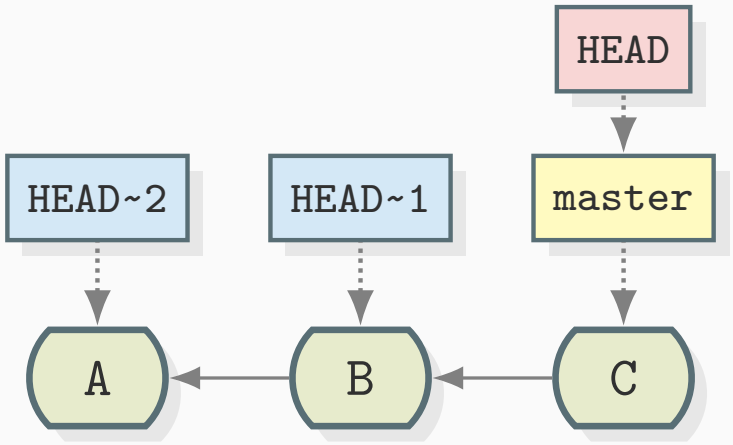
Före och efter `git commit`

Commits

Förutom att referera till commits med deras commit hash kan vi även referera till dem relativt till `HEAD`.

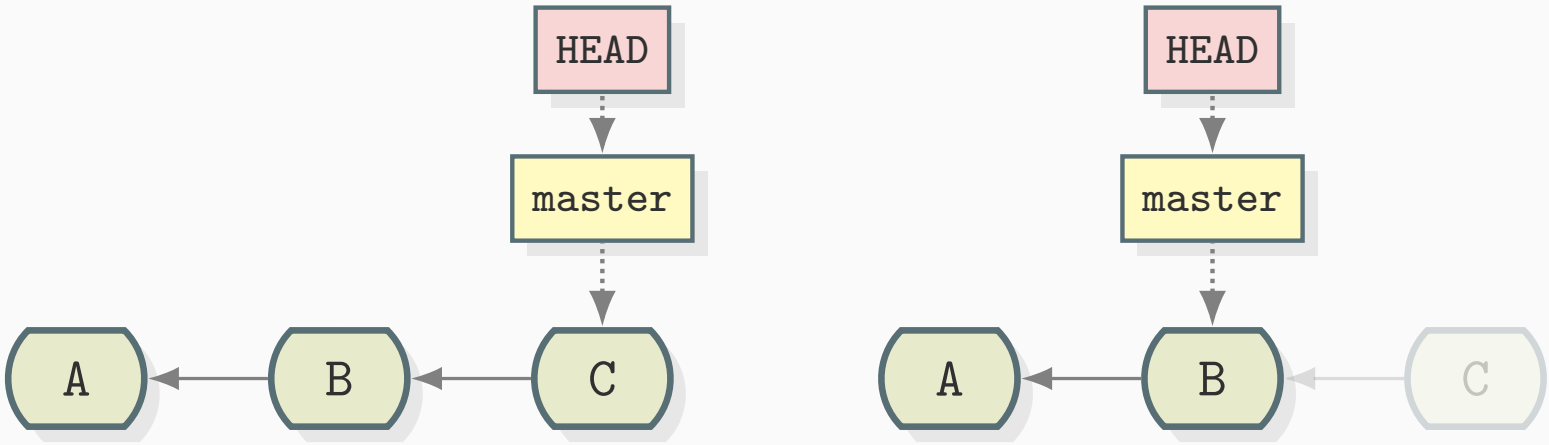
`HEAD~1` **är samma sak som att säga ”commit:en som är ett steg innan** `HEAD`**”.**

Commits



Relativa commits

Commits



Före och efter `git reset HEAD~1`

Grenar

Grenar

Git tillåter (och uppmuntrar) oss att dela upp arbetet i ”grenar” (branches).

Varje repo börjar med en `master`-gren.

Grenar

Grenar är billiga att skapa och använda.

De låter oss:

- arbeta på flera orelaterade features parallellt
- skapa en ny gren för att experimentera utan att riskera att ha sönder saker
- hantera releaser

Grenar

För att skapa en ny gren:

```
$ git branch my-branch && git checkout my-branch
```

Genväg:

```
$ git checkout -b my-branch
```

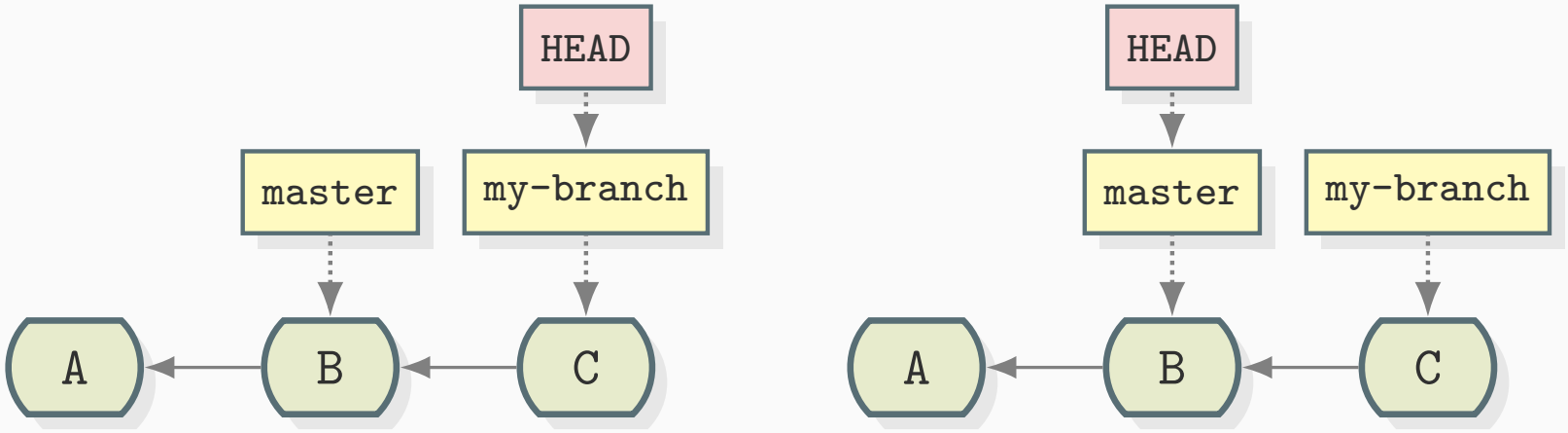
För att publicera den nya grenen:

```
$ git push -u origin my-branch
```

(`-u` innebär att Git kopplar ihop din lokala gren med den på `origin`. Se även `git checkout -t`.)

`HEAD` **följer med när vi byter branch.**

Grenar

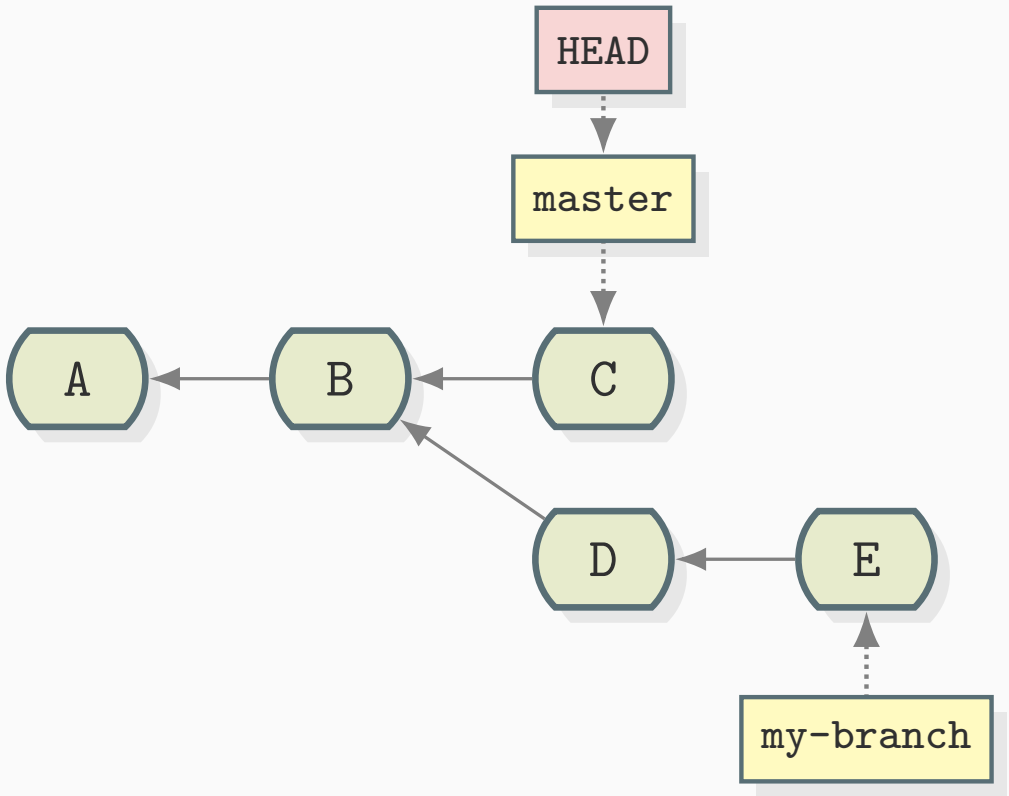


Före och efter `git checkout master`

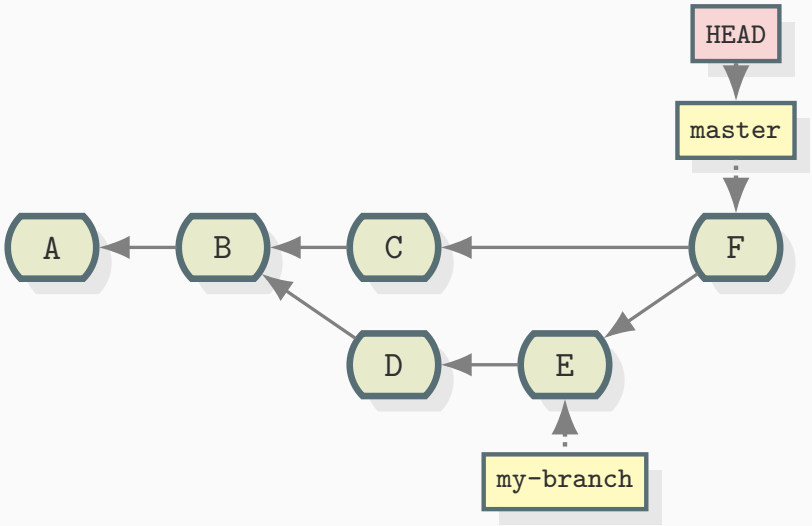
Grenar

För att samla ihop commits från olika grenar använder vi `git merge` **eller** `git rebase`**.**

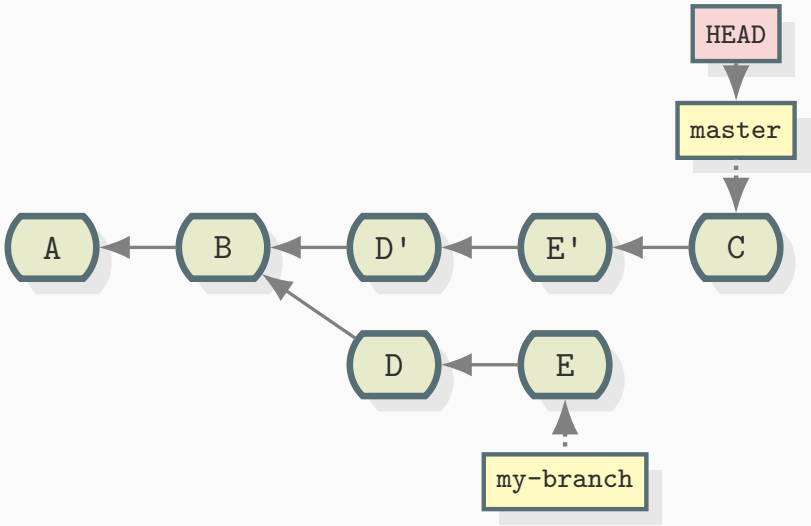
Grenar



Grenar



`git merge --no-ff my-branch`



`git rebase my-branch`

Tips

- **Använd `git merge --no-ff` för att spåra vilken gren ändringarna kommer ifrån.**
- **Använd `git rebase` när det är irrelevant.**

Övning 2: Merge och rebase

Samarbete och konflikter

Samarbete och konflikter

Att arbeta tillsammans med andra i samma gren kan vara problematiskt.

Samarbete och konflikter

```
$ git push
To git.kb.se:matnil/test.git
 ! [rejected]        master -> master (fetch first)
 error: failed to push some refs to 'git@git.kb.se:matnil/test.git'
 hint: Updates were rejected because the remote contains work that you do
 hint: not have locally. This is usually caused by another repository
 pushing
 hint: to the same ref. You may want to first integrate the remote changes
 hint: (e.g., 'git pull ...') before pushing again.
 hint: See the 'Note about fast-forwards' in 'git push --help' for
 details.
```

Samarbete och konflikter

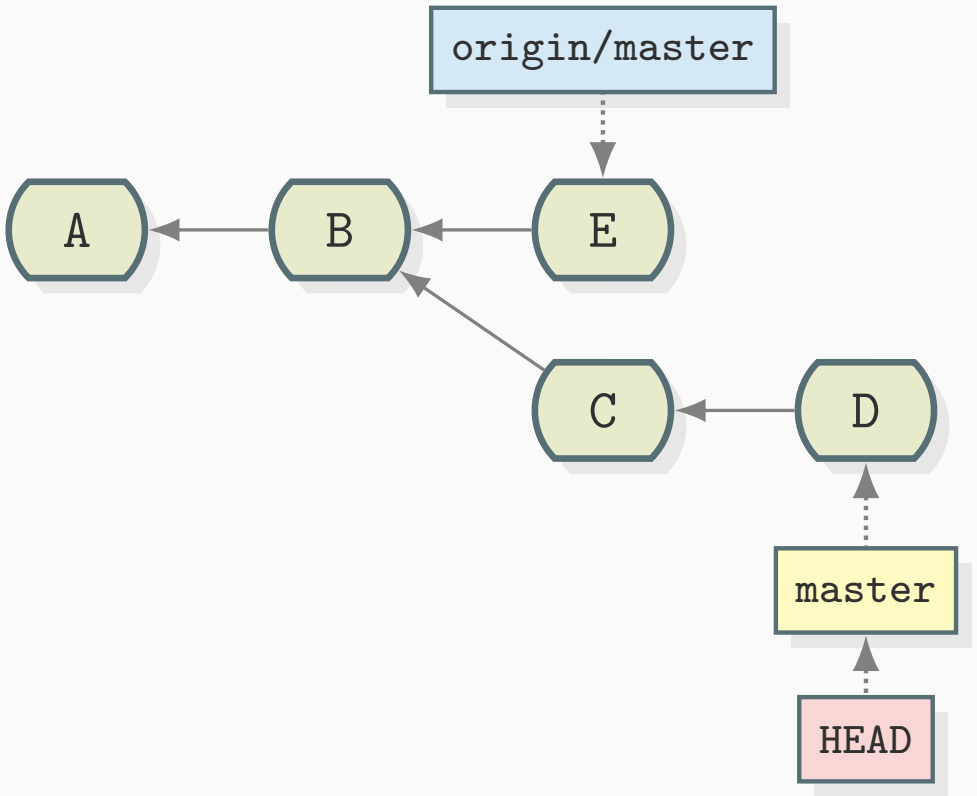
`git fetch` **uppdaterar det lokala repots bild av** `origin`**.**

Samarbete och konflikter

```
$ git fetch
Fetching origin
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git.kb.se:matnil/test
   efe053d..e37aaf7  master     -> origin/master

$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 2 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
  nothing to commit, working tree clean
```

Samarbete och konflikter



Samarbete och konflikter

`git pull` **är en genväg för**

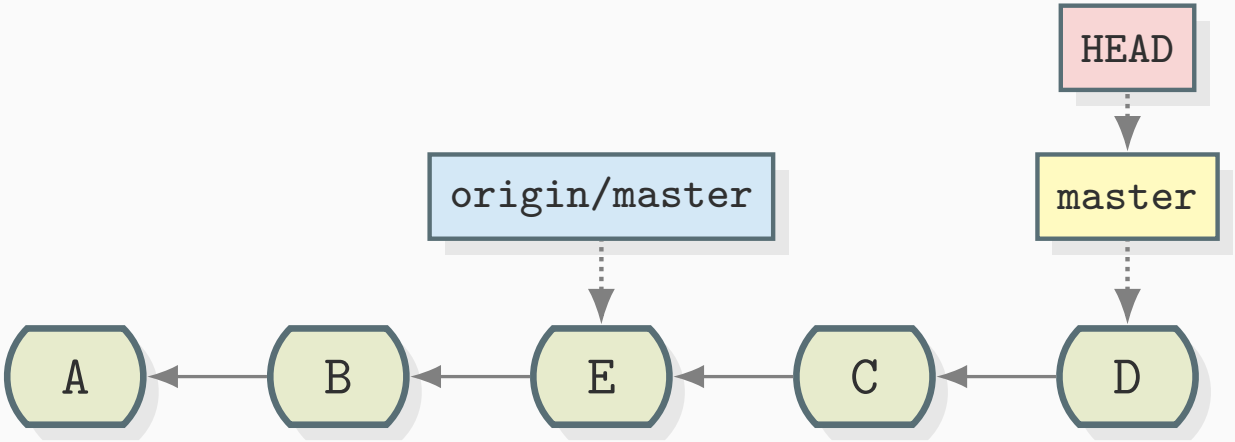
`git fetch && git merge`**.**

Ibland är det precis vad vi vill, men andra gånger (som i det här fallet) kommer det skapa en onödig merge-commit.

Samarbete och konflikter

`git pull --rebase` **låter oss undvika det.**

Samarbete och konflikter



Efter `git pull --rebase`

Rant

`git pull` **ska aldrig resultera i en merge-commit.**

Det tillför inget till historiken.

Samarbete och konflikter

Om flera personer arbetar i samma repo är konflikter inte ovanligt.

Konflikter (i koden) är inget att vara rädd för.

Övning 3: Konflikthantering

Kommunikation underlättar vårt arbete.

Samarbete och konflikter

För snåriga konflikter kan externa verktyg underlätta.

Ett sådant är `meld`[1].

[1] `http://meldmerge.org/`

När allt går fel

När allt går fel

Git anstränger sig för att skydda historiken.

Det innebär att vi (oftast) behöver anstränga oss för att slänga bort information. (T.ex. genom att använda `--force`.)

När allt går fel

```
git {merge|rebase} --abort
```

Avbryt och återgå till där du var innan du påbörjade sammanfogningen.

När allt går fel

```
git reset --hard origin/master
```

Släng bort det du har lokalt och börja om med det som ligger remote.

När allt går fel

```
git reset --hard origin/master
```

Släng bort det du har lokalt och börja om med det som ligger remote.

OBS

Var noga med att bara köra det här kommandot när det är okej att slänga bort lokala ändringar.

När allt går fel

`git reflog`

Git loggar varje gång `HEAD` ändras och `git reflog` visar loggen. Vi kan därefter använda `git checkout` för att gå till ett tidigare steg.

(`git checkout` ändrar också `HEAD`, så Git loggar det också.)

Tips & trix

Håll din gren uppdaterad

Se till att din gren innehåller senaste `master` innan du genomför en merge (eller skickar en pull request):

```
$ git checkout -b my-branch
# jobbajobbajobba
$ git checkout master && git pull
$ git checkout my-branch
$ git rebase master
# fixa eventuella konflikter
$ git checkout master && git merge --no-ff my-branch
```

Mina top 10 Git-kommandon:

```
$ history | awk '{ print $2, $3 }' | grep "^git" | \
    sort | uniq -c | sort -nr | head -n 10
855 git status
514 git diff
344 git log
214 git add
200 git checkout
179 git push
176 git commit
124 git fetch
 94 git stash
 79 git pull
```

Tips & trix

```
git diff --staged
```

För att kolla att du verkligen commit:ar det du tror.

Tips & trix

```
git tag --annotate
```

För att markera en release.

Tips & trix

```
git fetch --all
```

För att hämta hem nya taggar, nya grenar, osv.

Tips & trix

`git rebase --interactive`

För att skriva om historiken.

OBS

Det är viktigt att veta när det är okej att skriva om historik och när du definitivt inte ska göra det. Det samma gäller `git push --force`**.**

Tips & trix

```
git config --global rerere.enabled true
```

Spara lösningar på manuellt fixade konflikter och applicera dem automatiskt[1].

[1] https://git-scm.com/blog/2010/03/08/rerere.html

Tips & trix

`git cherry-pick`

För att plocka in individuella commits i din gren.

Tips & trix

```
git stash
```

För att (tillfälligt) spara undan ändringar du inte vill commit:a.

Tips & trix

`git blame`

För att ta reda på när en viss rad skrevs och av vem.

Bestäm ett arbetsflöde.

Se t.ex. `https://www.atlassian.com/git/tutorials/comparing-workflows`.

Frågor?

Resurser

Resurser

- **Pro Git**

  `https://git-scm.com/book/en/v2`
- **A Note About Git Commit Messages**

  `http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html`
- **Git for ages 4 and up**

  `https://www.youtube.com/watch?v=1ffBJ4sVUb4`
- **Git GUI Clients**

  `https://git-scm.com/downloads/guis`