Hvordan løse (og skape) problemer i Git

Imre Kerr, Fredrikstad 2022

Hvordan bli en git-guru: Inngående kunnskap om hvordan ting faktisk henger sammen
Hva *er* en commit, branch, ff vs 3way merge, etc
Men å lære abstrakte ting er vanskelig uten knagger å henge på. Derfor holder vi det praktisk for nå.

Commit ofte

Så fort noe er commitet, er det så å si umulig å miste det.

Bedre å lage en rotete historikk og så rydde opp i etterkant.

Så fort noe er commitet, er det lagret i git, og er nesten umulig å miste helt.
Uansett hvor rotete ting blir, kan noen hjelpe deg hvis du har fulgt denne ene regelen.

NB! Garbage collection!

Reachability

En commit er *reachable*, dersom den

a) pekes på av en *ref* (branch eller tag), eller
b) er forelderen til en annen reachable commit

Regel: Reachable commits blir aldri slettet automatisk!

Tilgjengelig? Nåbar?

Lag en midlertidig branch før du gjør noe skummelt

```
git branch save-point-1
```

Hvis branchen din blir ødelagt:

```
git reset --hard save-point-1
```

og alt er som før.

Skummelt: Omskriving av historikk, store merges

Brancher er bare pekere til commits!

Omskriving av historikk

Rette på siste commit:

```
git commit --amend
```

Flytte aktiv branch til en annen commit:

```
git reset [--soft|--mixed|--hard] <commit>
```

Kopiere commit til aktiv branch (ikke omskriving, teknisk sett):

```
git cherry-pick <commit>
```

Ultimat multiverktøy:

```
git rebase --interactive [--onto <newbase>] <upstream> [<branch>]
```

Og dokumentasjonen er faktisk bra nå!

Lett å skape problemer her…

Reset uten args skriver ikke om historikk.

git rebase --interactive kan:
Slette enkeltcommits
Slå sammen commits
Splitte opp commits (bruk “edit”!)
Endre rekkefølge på commits
Rette på commits lengre bak i historikken

Ikke skriv om historikk på delte brancher

I praksis: Ikke force-push main/master, dev/stage/prod, versjonsbrancher etc.

Kan håndheves med branch protection på GitHub.

Ryddig commit-historikk

Letter code review, bisect, lettere å finne når og hvorfor endringer ble gjort…

Flytt rundt på commits, slå sammen, splitt opp

Splitte ucommitede endringer:

```
git add --patch
```

Bisect – finn når en bug ble introdusert

```
git bisect start
```

```
git bisect bad
```

```
git bisect good <kjent fungerende commit>
```

Deretter, til git sier at du er ferdig:

```
git bisect [good|bad]
```

Eller automatisk testing med `git bisect run <script> [<args>]`

Best om alle commits bygger og tester grønt (annet enn den aktuelle bugen)

Alternativt:

```
git bisect skip
```

Finne igjen unreachable commits

Historikk over commits du har vært innom:

`git reflog` / `git reflog <branch>`

Finn hashen til commiten din, og lag en branch som peker på den:

`git branch lost-commit a1b2c3`

Vanskelig å lese output fra git reflog?

`git log --reflog --graph --oneline`

```
* d0f8d82 (HEAD -> master) Third commit
* b5ae8d9 Second commit
* 87fda72 Initial commit
(END)
```

```
* d0f8d82 (HEAD -> master) Third commit
| * 77e1756 Oops, unreachable commit
|/
* b5ae8d9 Second commit
* 87fda72 Initial commit
(END)
```

Når du har gjort dette kan du gjøre merge, cherry-pick, restore, rebase eller hva du måtte trenge for å få ting på stell igjen.

Siste kommandoen lager en commit-graf som ligner på den man kjenner fra GUI-verktøy.
Kan GUI-verktøyet ditt gjøre det kanskje? (Seriøst, kan det? For jeg vet ikke.)

Finne igjen unreachable commits… på GitHub

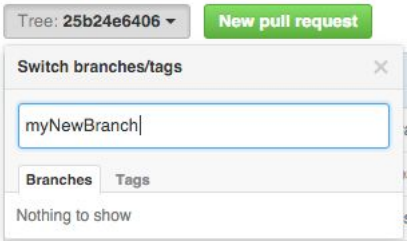
"Repository events" – GitHubs "reflog"

https://api.github.com/repos/{owner}/{repo}/events

Finn event av type PushEvent, sjekk before-feltet.

https://github.com/{owner}/{repo}/tree/{hash}

Og lag ny branch fra dropdown.



Events-APIet returnerer all mulig drit. Her må du sikkert grave litt.

Bedre å unngå det helt, med branch protection

Pull requests viser force-pushes…

Alternativt kan du spore opp en utvikler som har commiten på sin maskin og be dem om hjelp.

Finne commit i historikken

`git log` har en søkefunksjon:

```
git log -S <string>
```

Kan såklart kombineres med `--reflog`

Fjerne ting fra git-historikken

Hemmeligheter, GDPR-brudd, store filer…

På feature-branch: `git commit --amend` / `git rebase --interactive`

Lengre tilbake: BFG Repo-Cleaner (https://rtyley.github.io/bfg-repo-cleaner/)

⚠️ For å slette ting permanent etterpå:

```
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

❌ Ikke bruk: `git filter-branch`

Ukrypterte passord skal ikke sjekkes inn… Finnes verktøy for å finne dem i historikken: https://thecyberpunker.com/blog/git-exposed-pentesting-git-tools/

Testen på hvilken du skal bruke: Er commiten(e) bare reachable fra én ref? I så fall kan du bruke de innebygde verktøyene.
NBNB! Hva ligger på utviklermaskinene?

Filter-branch er så crap at selv dokumentasjonen til kommandoen anbefaler at du bruker noe annet.

BFG Repo-Cleaner, steg for steg

Alt må renses. Derfor:

Alle på teamet:

1. Commit og push (feature/midlertidig branch er fint)

En person:

2. `git fetch`, og så `git checkout {branch}` for hver remote-branch
3. Kjør BFG
4. `git push --force --all`
5. `git push --force --tags`

Alle:

6. Slett repo-mappa og clone på nytt.

Videre lesning

Mitt 💡-øyeblikk:

http://think-like-a-git.net/

Dypdykk:

https://github.com/pluralsight/git-internals-pdf

https://git-scm.com/book/en/v2

Tid for oppgaver!

Last ned oppgaver og slides: https://bit.ly/3wCkAh4

Jobb gjerne i par/små grupper

Flere av oppgavene kan løses på flere måter. Diskuter pros/cons.