

Versjonskontroll

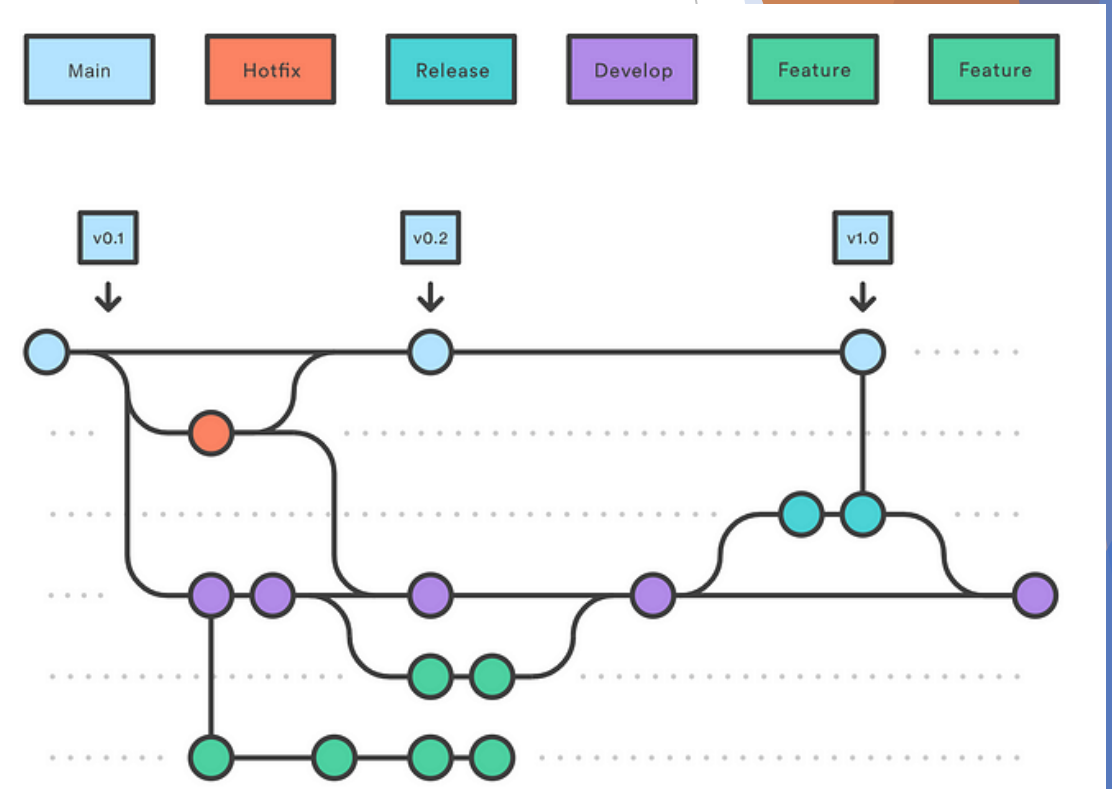
- git og github

Agenda

- ▶ Versjonskontroll: Hva og hvorfor
- ▶ Source Code Management og Git
- ▶ Github
- ▶ Git-funksjonalitet
- ▶ Anbefalt arbeidsflyt (team og hver enkelt utvikler)

Git-funksjonalitet - Branches

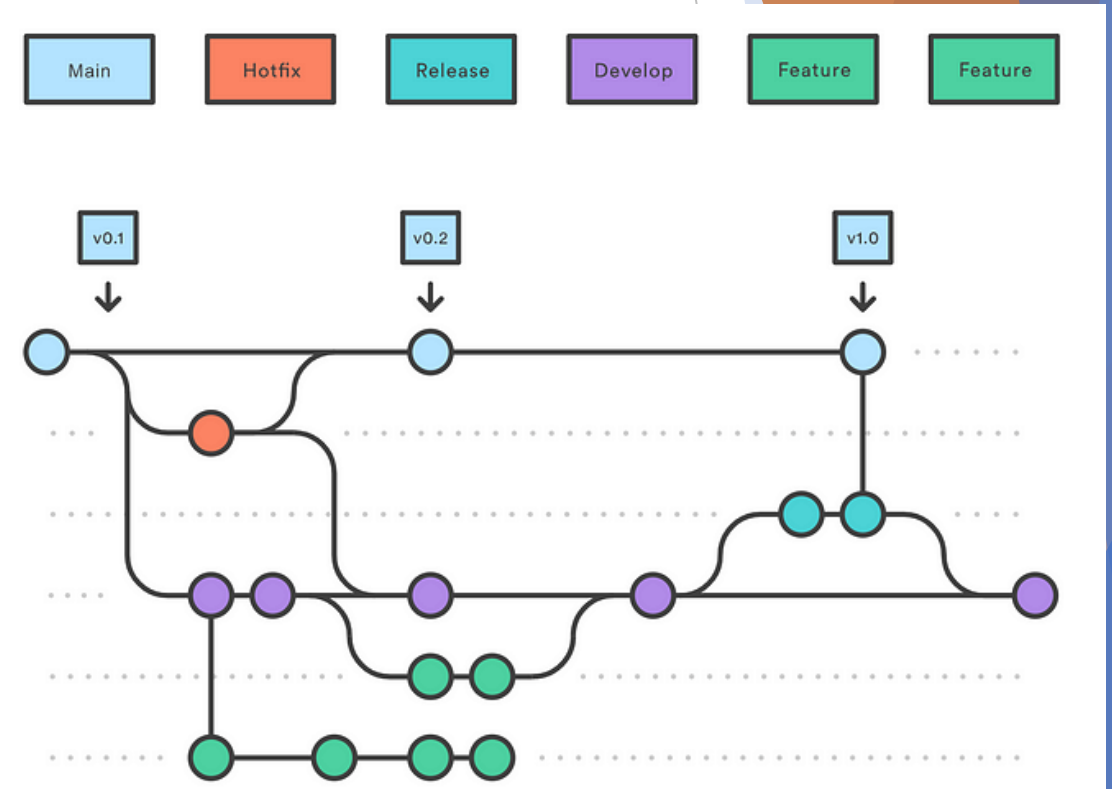
- ▶ En branch er en parallell versjon av repo-et
 - Kan være remote eller lokal (ofte begge deler)
 - Fordelaktig for at utviklere skal kunne jobbe samtidig og trygt
- ▶ master/main branch
 - Opprettes samtidig som repo-et og representerer "hovedversjonen"
 - Navn kan variere (kalles main i GitHub)
 - Når vi jobber i et team burde vi unngå å jobbe direkte i main
- ▶ Vi kan i tillegg lage andre branches for forskjellige formål
 - Develop, Feature, Hotfix, Release ...
 - Finnes flere strategier, men ingen fasist



<https://medium.com/@dmosyan/version-control-branching-strategies-e68e8d5ef1e0>

Git-funksjonalitet - Merging

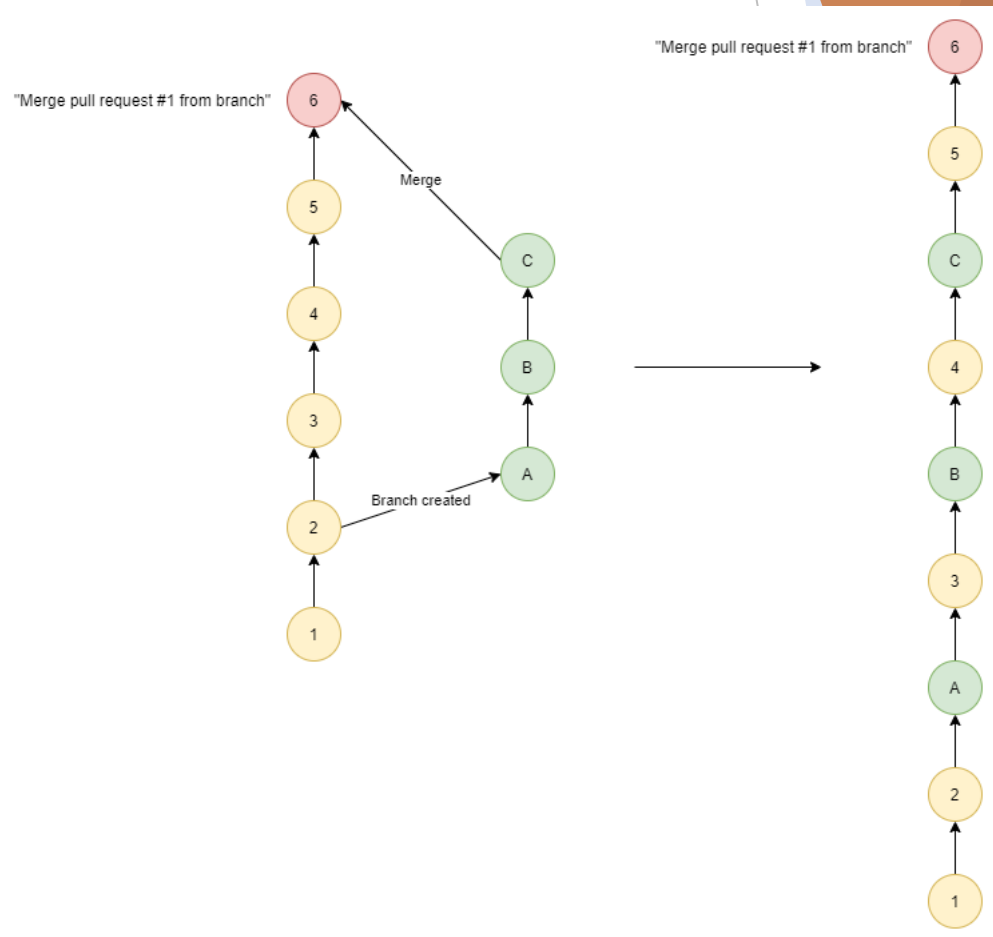
- ▶ Selv om branches er nyttige ønsker vi til slutt ett produkt (samlet på en branch)
- ▶ Vi må på et vis slå sammen funksjonalitet utviklet på forskjellige branches
 - Dette kalles merging
 - I praksis: En «base-branch» pull-er en annen branch inn i seg
- ▶ Det finnes flere typer merging
 - Merge
 - Fast forward merge
 - Squash and merge
 - Rebase and merge



<https://medium.com/@dmosyan/version-control-branching-strategies-e68e8d5ef1e0>

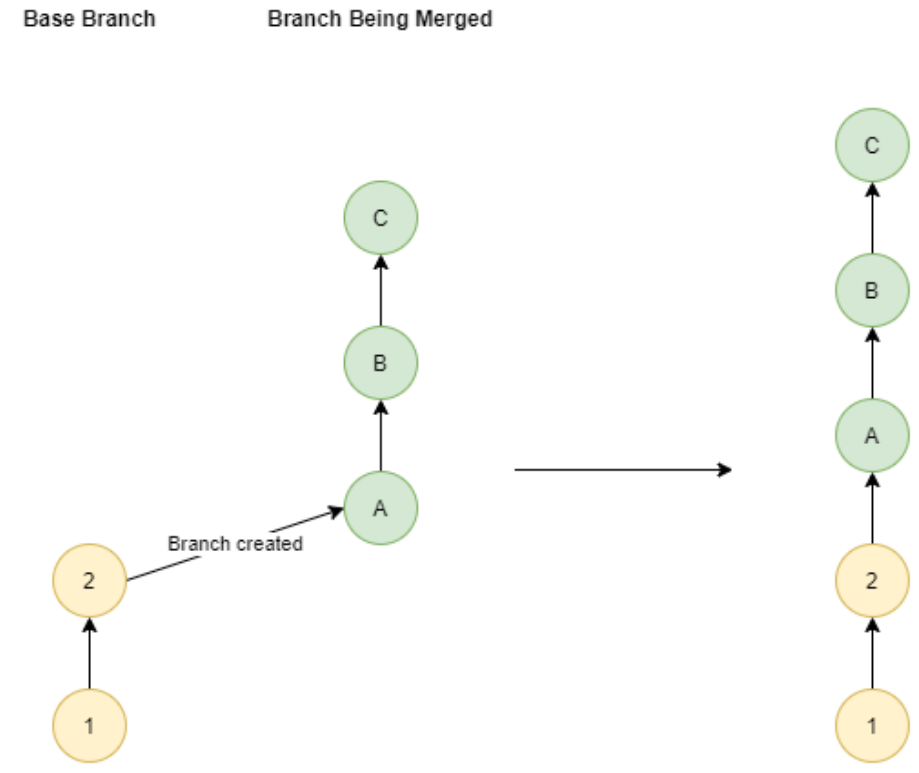
Git-funksjonalitet - Merge

- ▶ Merge - Setter sammen commits i en branch med en annen basert på tidspunkt
 - ▶ Endringen slås egentlig sammen samtidig i en merge-commit
- ▶ Fordeler
 - Beholder alle commits
 - Beholder tidspunkt av endringer
 - Beholder informasjon om branches
- ▶ Ulemper
 - Kan føre til rotete og uoversiktlig historikk



Git funksjonalitet - Fast forward merge

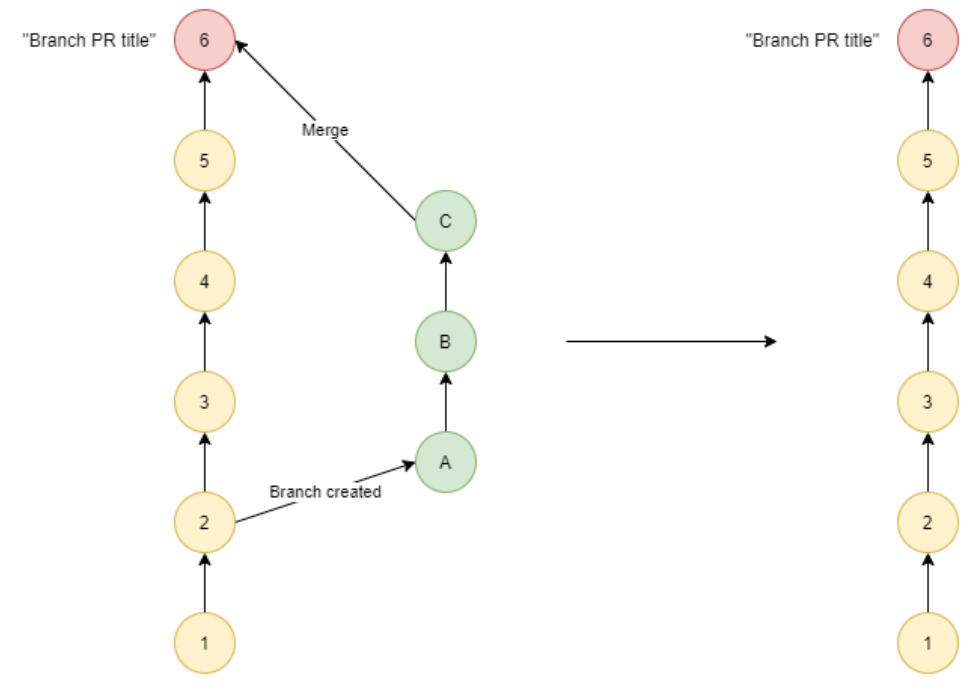
- ▶ Hvis base-branchen ikke har blitt oppdatert i mellomtiden kan branch-commitene legges på slutten av main uten en merge commit
 - Kalles Fast forward merge og gjøres ofte automatisk hvis mulig
- ▶ Fordeler
 - Beholder commits
 - Viser commits som om de har blitt gjort direkte i base-branchen
- ▶ Ulemper
 - Er bare relevant hvis det ikke har skjedd noen endringer i base-branchen
 - Mister informasjon om branches



<https://lukemerrett.com/different-merge-types-in-git/>

Git funksjonalitet - Squash and merge

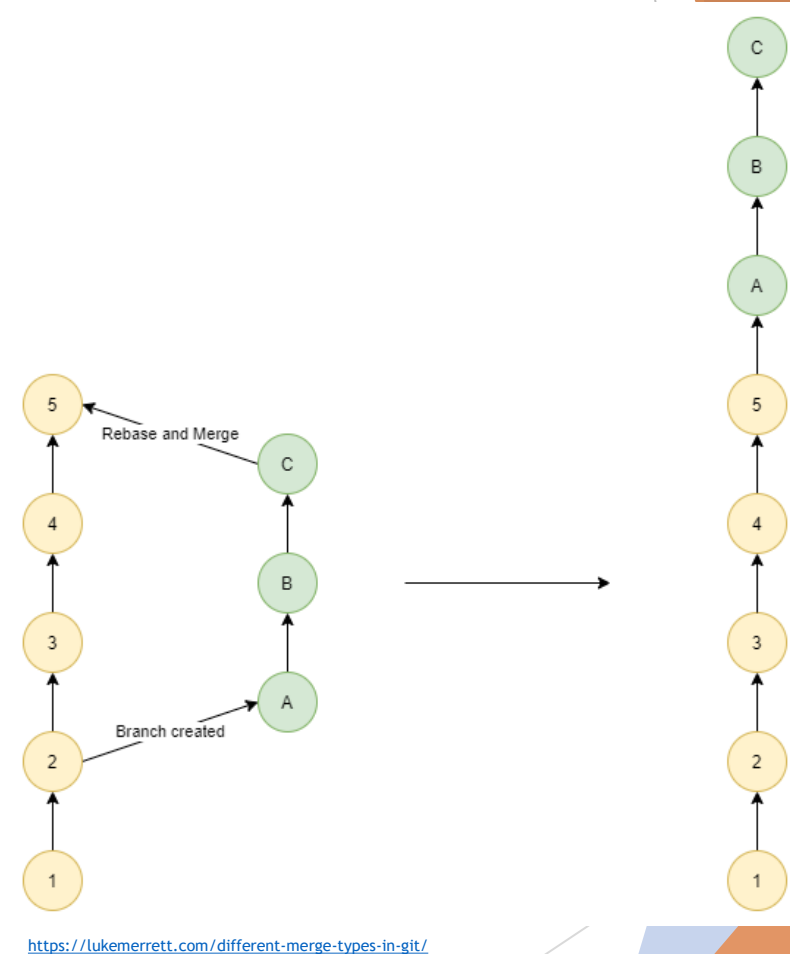
- ▶ Squash merge - Slår sammen commitene i en branch til én commit og merger denne commiten til base-branch
- ▶ Fordeler
 - Luker vekk små-commits og fører ofte til en mer oversiktlig historie
 - Samler mer overordnet arbeid i én commit
- ▶ Ulemper
 - Mister individuelle commits og dermed også muligheten til å se / revertere til mer spesifikke "tidspunkter"
- ▶ Kan også selektivt squashe enkelte konseptuelle små-commits, mens la andre stå før man merger på vanlig måte



<https://lukemerrett.com/different-merge-types-in-git/>

Git funksjonalitet - Rebase and Merge

- ▶ Rebase and merge - Tar commitene i en branch og legger dem på slutten av base-branch uavhengig av tidsperspektiv
- ▶ Fordeler
 - Historikken blir ofte mer intuitiv, hvor feature-commits blir ryddig lagt til etter hverandre
 - Beholder alle commits
- ▶ Ulemper
 - Føles mer komplisert ut enn andre merge-typer
 - Vær bevisst på å ikke overskrive endringer som har skjedd i base-branch i mellomtiden
 - Mister kontekst av branches



Git funksjonalitet - Merge conflicts

- ▶ Når vi merger kan det oppstå konflikter
 - Typisk overlappende endringer - F.eks. at begge har endret på en metode
- ▶ I slike tilfeller må vi manuelt bestemme hvordan konflikten skal løses:
 - Velge én av endringene som skal beholdes
 - Beholde begge endringene
 - Manuelt gå inn og skrive om til en ny og bedre løsning
- ▶ Løses typisk lokalt

Eksempel - Branches, merging og merge conflicts


Git-funksjonalitet - .gitignore

- ▶ .gitignore er en fil som spesifiserer en liste med filer i lokale repos som aldri skal commites eller pushes
 - Git ignorerer disse fullstendig
- ▶ Nyttig for slikt som
 - Filer generert fra kode
 - Temp-filer under kjøring
 - Personlige IDE konfigurasjoner
 - [Typiske filer som ignoreres for forskjellige språk](#)
- ▶ Ekstra viktig for **HEMMELIGHETER** (passord, kryptografiske nøkler, osv.)
 - Egentlig best å håndtere disse utenfor prosjektet
 - Settes inn fra miljøet rundt applikasjonen
 - En egen config-fil som ignoreres
 - Et eget privat repo som er reservert for hemmeligheter

Dev put AWS keys on Github. Then BAD THINGS happened

Fertile fields for Bitcoin yields - with a nasty financial sting

By [Darren Pauli](#) 6 Jan 2015 at 13:02

25  SHARE ▼

Bots are crawling all over GitHub seeking secret keys, a developer served with a \$2,375 Bitcoin mining bill found.
https://www.theregister.com/2015/01/06/dev_blunder_shows_github_crawling_with_keyslurping_bots/

Forking

- ▶ En fork er en komplett, men selvstendig versjon av et eksisterende prosjekt
 - ▶ I praksis at vi kopierer et eksisterende public prosjekt til vårt eget repo
 - ▶ «Forken» beholder fullstendig historikk
- ▶ Vanlig for at utviklere som ikke har direkte tilgang skal kunne jobbe på prosjektet (privat eller komme med forslag)
- ▶ Skjer også ofte med open source-prosjekter
 - ▶ Uenigheter: Noen utviklere forker og forsetter i en annen retning
 - ▶ F.eks. MySQL → MariaDB og Percona
 - ▶ Andre bruker open source-prosjektet som et fundament for et eget prosjekt

Git-funksjonalitet - Pull requests

- ▶ En pull request er et forslag til endringer
 - ▶ Endringene blir gjort på en egen lokal branch (i prosjektet eller i en fork)
 - ▶ Endringene forsøkes å pulles (merges) inn i «hovedbranchen» men må godkjennes av prosjekteier(e)
 - ▶ Altså begrepet «pull request»
- ▶ Fungerer som en måte å starte diskusjoner rundt endringene
 - ▶ Er de fornuftige?
- ▶ Dette er hvordan open source-prosjekter blir utviklet
 - ▶ Alle kan bidra, men endringer må godkjennes
- ▶ Kan også være nyttig i interne prosjekter

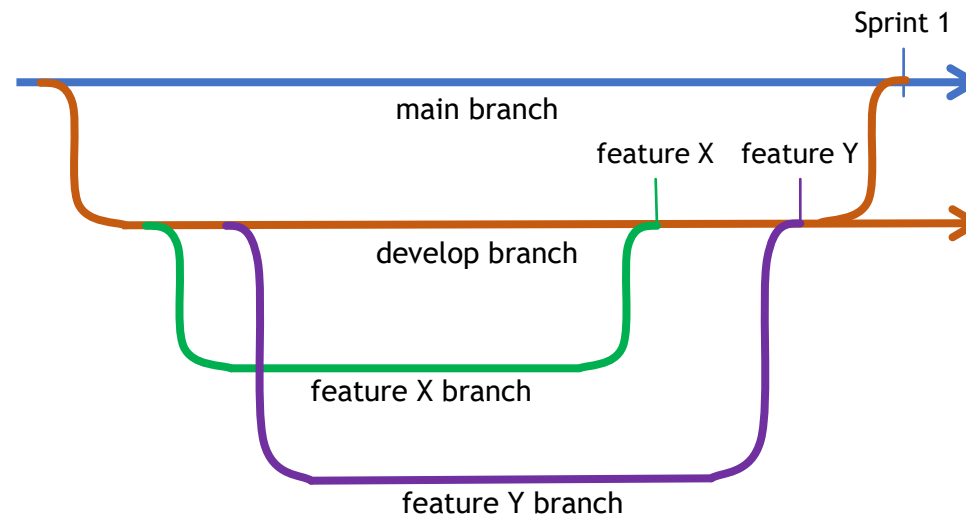
Anbefalt arbeidsflyt (Team)

► Del opp i branches

- main
 - Oppdateres hver sprint / innlevering
- develop
 - Inneholder alle nyeste features
 - Merges til main mot slutten av hver sprint
- feature
 - Lag en unik branch for hver feature (Brukes av én person)
 - Brukes til commits mot en spesifikk feature

► Hver sprint

- Oppdater backlog
- Uviklere velger / fordeler features og lager sine egne branches fra develop
- Merge feature-branches inn i develop når de er ferdige og slett dem
- Når sprint nærmer seg slutten merge develop inn i main (men ikke slett den)



Anbefalt Arbeidsflyt (Hver enkelt utvikler)

- ▶ Velg en feature/oppgave fra backlog-en
- ▶ Lag en ny branch og oppkall den etter feature-en
- ▶ Jobb på feature-branchen
 - Commit ofte - Kan være en ide å navngi etter featuren
 - Hvis develop oppdaterer seg betydelig i mellomtiden - MERGE develop inn i feature-branchen (IKKE rebase!)
 - Ideen: Løse merge konflikter ofte for unngå store senere
- ▶ Merge feature-brachen inn i develop når featuren er ferdig (Evt. pull request)
 - Løs eventuelle merge conflicts
 - Husk at du skal BYGGE på develop. Altså oftest naturlig at du justerer ditt eget arbeid
 - Ta eventuelt kontakt med utvikleren(e) av det som det krasjer med
- ▶ Slett feature-branchen

Workshop

- ▶ Opprett GitHub-konto (hvis ikke du allerede har en)
- ▶ Opprett et public repo og inviter andre gruppemedlemmer som collaborators
- ▶ Opprett en develop-branch
- ▶ Last ned en Git GUI-applikasjon (for eksempel GitKraken)
- ▶ Lag hver en lokal branch (fra develop) gjennom applikasjonen for å teste
 - ▶ Velg navn selv
 - ▶ Opprett noen filer med vilkårlig innhold
- ▶ Push branchen og endringene til remote
- ▶ Merge branches inn i develop, og så develop inn i main
- ▶ Prøv gjerne å lage overlappende innhold i samme fil for å bli kjent med merge conflicts