Introduksjon til

Agenda

- Historie
- Git virkemåte
- Konsepter og Kommandoer
- Praktiske øvelser

Historie

Historie

- Source Code Control System (SCCS)
- The Revision Control System (RCS)
- Concurrent Versions System (CVS)
- Microsoft Visual SourceSafe
- Subversion
- BitKeeper
- Git

Sentralisert versjonskontroll

- All historikk ligger på sentral server
- De fleste operasjoner foregår sentralt
- Ofte låsing av filer en jobber på
- Mange operasjoner er tunge og tar lang tid
- Dårlig støtte for å jobbe offline

Distribuert versjonskontroll

- Alle har en lokal kopi av hele repoet
- De fleste operasjoner utføres lokalt
- Ikke behov for kontakt med server for å jobbe
- Alle har en backup, om noe skulle gå galt

Historie Git

- Utviklet av Linus Torvalds som kildekontrol for Linux-kjernen
- 1.0 release desember 2005
- Ville lage distribuert, BitKeeper-aktig arbeidsflyt
- CVS som eksempel på hva en ikke skulle gjøre
- Implementere en sikring mot feil, enten som uhell eller ondsinnet

Git Nøkkelfunksjonalitet

- Distribuert System
- Branching and Merging
- Hurtig og Effektivt
- Historie

Populære Plattformer for Remote Git Repositories

- GitHub
- GitLab
- Bitbucket
- Azure DevOps

Git nøkkelkommandoer

- `git init` - Initialisere et nytt Git repository.
- `git clone` - Opprette en lokal kopi av et remote repository.
- `git add` - Stage endringer før en commit.
- `git commit` - lagre staged endringer med en besskrivende melding.
- `git push` - Laste opp lokale endringer til et remote repository.
- `git pull` - laste ned og merge endringer fra et remote repository.
- `git status` - sjekke status på lokale endringer i den aktive branch.

Ordliste

- remote - en referanse til en server
- clone - lage en lokal kopi av et repo på en server
- init - initialisere git i en lokal folder
- branch - en representasjon av en utviklingslinje
- main branch - standard branch. Den eneste som finnes i et nyopprettet repo

Ordliste

- .gitignore - en fil som inneholder filer og foldere git skal ignorere
- commit - et punkt i en git-historie. Hele historien er representert av en rekke med relaterte commits.
- stage changes - fortelle git hva som skal inngå i en commit
- fetch - oppdatere lokale kopier av branches fra server

Ordliste

- merge - inkludere innhold fra en anne branch inn i den gjeldene branch
  - merge commit - en commit som opprettes for å slå sammen to branch-er
  - fast-forward - en merge der det ikke opprettes en merge commit
- pull - oppdatere den aktive branch med endringer fra server
  - kombinasjon av en fetch og en merge

Ordliste

- push - flytte lokale endringer opp på server
- stash - midlertidig stue vekk endringer
- tag - merke en commit. For eksempel for å markere en release
- cherry-pick - inkludere utvalgte commits fra en branch inn i en annen

Git Virkemåte

Objekter og Relasjoner

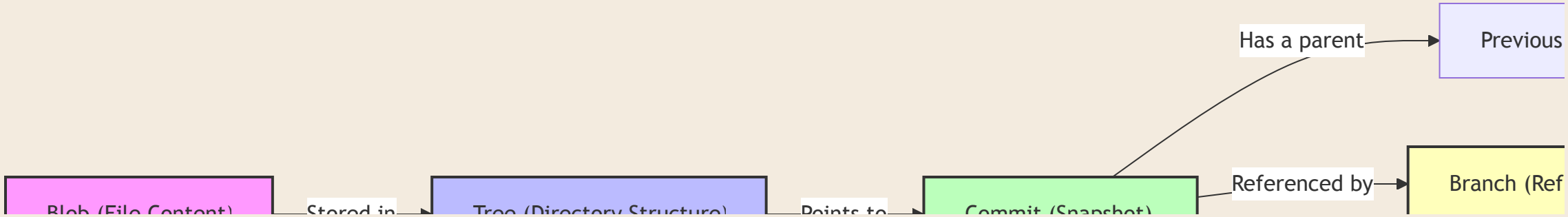
SHA1 hashing

- alle objekter i Git blir gitt en sjekksum generert med SHA-1
- kan referere til en hash med de første 7-8 karakterer

```
commit 58d1419168aee0eb02e09733a134470405ca3aec
Author: Kristian Berg <kristian.berg@ks.no>
Date:   Fri Dec 13 15:12:32 2024 +0100
```

- commit-en ovenfor kan derfor refereres med `58d14191`

Relasjoner Mellom Git Objekter



BLOB

- filer lagres som en BLOB
- filnavnet er SHA-1 sjekksummen
- innholdet er komprimert med `zlib`
- BLOBs med samme innhold gjenbrukes
- kan også lagres som en endring fra en eksisterende BLOB

Tree

- et tre er en representasjon av en filstruktur
- filnavnet er SHA-1 sjekksummen
- inneholder referanser til underliggende foldere med navn og sjekksum
- inneholder referanser til filer med navn, sjekksum og tilganger

Tree

```
100644 blob caa32e6753a00606f47a9d4781a7db9767cee246    .gitignore
040000 tree deae07085b9991546b7de53010d48239d27bdfa1    images
100644 blob bd92ad6c518f82c4ce16a2703a056d82c670889a    marp.config.js
100644 blob d63c8fd144d1921bc54cb2e129fe15886def0f9c    slides.md
```

Commit objekter

- Et commit-objekt er en fil som inneholder
  - en referanse til parent-commit
  - en referanse til rot-tre
  - metadata om commit (author, committer, gpg signatur og commit-melding)

Commit objekter

```
tree d0b128700528a961dfcafc9100e7db1b087a07a8
parent 58d1419168aee0eb02e09733a134470405ca3aec
author Kristian Berg <kristian.berg@ks.no> 1739307410 +0100
committer Kristian Berg <kristian.berg@ks.no> 1739307410 +0100
gpgsig -----BEGIN PGP SIGNATURE-----
 ...
 -----END PGP SIGNATURE-----

Startet på slides
```

Branch objekter

- en fil med navnet til branch-en
- peker på et commit-objekt
- vil oppdateres ved nye commits
- eksempel: `.git/refs/heads/main`

```
3be30ce68527b6b9dacf8534cde0f0cb878581ef
```

Brancher og Hvor De Lever

Git har tre typer branches:

- Lokale branches – Dine egne branches som du aktivt jobber på.
- Remote-tracking branches – Speilbilder av branches på remote server (for eksempel origin/main).
- Remote branches – Branches som faktisk eksisterer på remote server (origin eller en annen remote).

Lokale Branches

- Disse lever kun på din maskin.
- Når du oppretter en ny branch:

```
git branch feature-branch
```

- Git lager en lokal branch basert på den nåværende committen.
- Denne branchen har ingen direkte kobling til en remote branch før du setter opp tracking.

Remote Branches (Fjernbranches)

Disse lever kun på remote server (GitHub, GitLab, Bitbucket osv.).

Remote-tracking Branches (Fjernspeil)

- Dette er lokale, readonly-kopier av fjernbranches.
- De oppdateres når du kjører git fetch eller git pull.
- Eksempler:
  - origin/main er en remote-tracking branch som speiler main på serveren.
  - origin/feature-branch speiler feature-branch på serveren.

Tag objekter

- brukes til å markere en commit med en spesiell mening. f.eks en release
- en fil med navnet til tag-en som peker på en git commit
- vil aldri kunne flyttes, med mindre den slettes og opprettes på nytt

`.git/refs/tags/0.0.1`

```
3be30ce68527b6b9dacf8534cde0f0cb878581ef
```

Konsepter og kommandoer

Opprette Nytt Repo lokalt

- Å kjøre `git init` vil initialisere git i den folder en står i
- Dette oppretter et tomt git repository
- Lager en `.git` folderstruktur
- Ingen filer er 'staget'

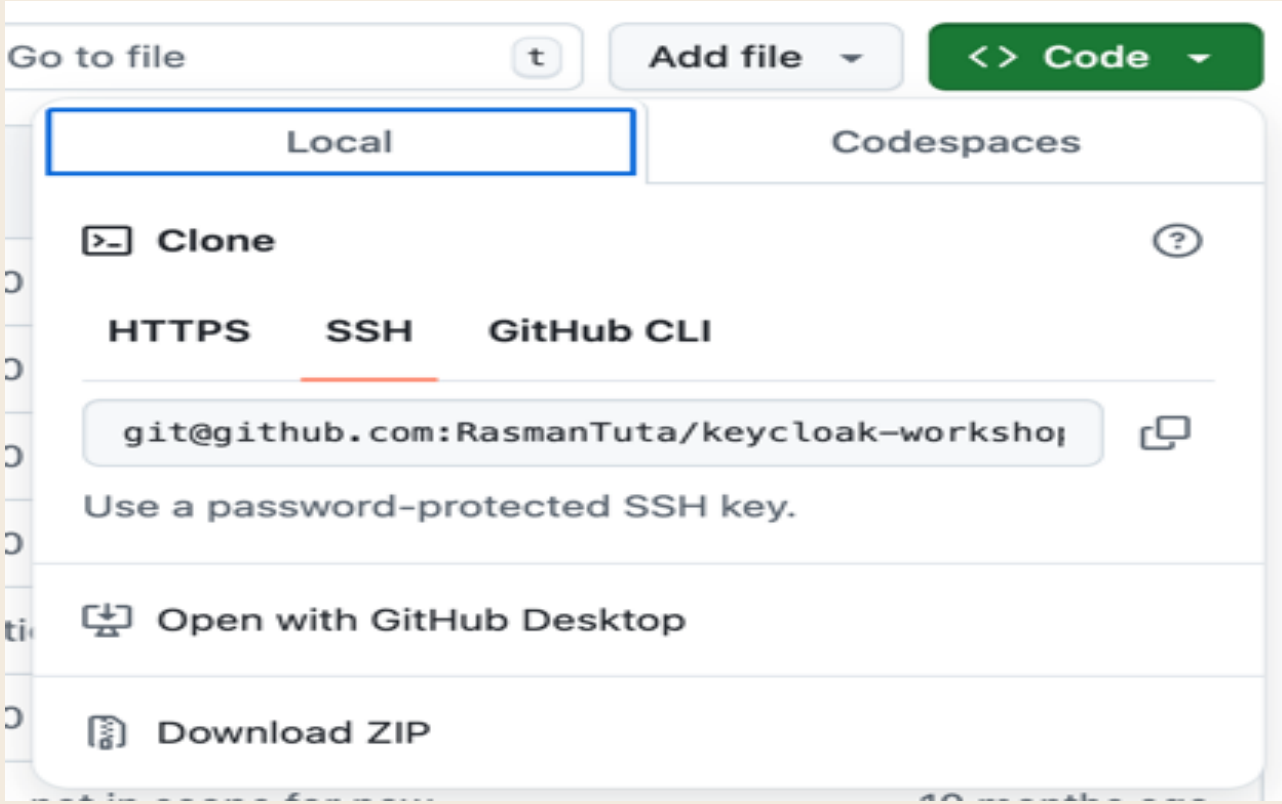
Opprette Nytt Repo remote

- Gå inn på den foretrukne git plattform
- Opprett nytt repository
- Følg instruksjoner for å
  - lage lokal klone
  - Laste opp et eksisterende prosjekt

Klone et eksisterende remote repo

- Gå på siden for det remote repoet
- Finn lenken til repoet. Enten SSH eller HTTPS
- kjør `git clone <lenke fra remote>`

Eksempel Github



git status

- Viser status for lokale endringer: `git status`

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   images/gitclone.png

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   slides.md
```

Git Log

- `git log` - Viser commmit-historikk i pager. Space skroller en side, Q avslutter.
- En mengde forskjellige modus
- `--oneline` - En linje per commit med kort hash, melding og evt tags
- `--graph` - Viser en enkel sammenheng mellom commits
- `-p` - Lister opp commits med diff

'staging' av filer

- Kun filer som er 'staget' vil komme med i en commit
- Enkeltfiler 'stages' med `git add <path til fil 1> <path til fil 2>`
- Alle filer 'stages' med `git add .`

.gitignore

- En fil med navn `.gitignore` inneholder mønster på filer og foldere git ikke vil inkludere i en commit
- `/` - brukt som mønster for folder(-struktur)
- `*` - wildcard for en eller flere karakterer
- `build/` - ignorerer alle foldere som heter 'build'
- `build` - ignorerer filer som heter 'build'

fetch og pull

- `git fetch origin` - vil oppdatere alle remote-tracking branches fra server
- `git pull` - vil først gjøre en 'fetch', for så å gjøre en 'merge' av den aktive branch
- `git pull --rebase` - vil i stedet for å gjøre en 'merge', gjøre 'rebase'. Mere om merge og rebase senere

commit

- `git commit` - vil lage et snapshot av alle 'stagede' filer og åpne editor for commit-melding
- `git commit -a` - vil i tillegg 'stage' alle kjente, endrede filer før commit gjennomføres
- `git commit -m "Commit-melding her"` - lar deg legge på melding
- `git commit --amend` - legger endringer (om noen) til forrige commit, og eventuelt endre commit-melding.

push

- Push laster opp lokale endringer til remote server
- Vil feile om om lokal historikk ikke stemmer overens med 'upstream'
- `git push origin main` - vil 'pushe' main branchen til `origin`

stash

- 'stash' er et mellomlager der du kan stue bort endringer for å hente dem frem siden
- `git stash` - stuer bort endringer
- `git stash pop` - henter dem frem igjen

konflikter

- 'merge', 'rebase' og 'stash pop' er typiske operasjoner som kan lage en konflikt.
- Konflikter oppstår når flere har endret samme sted i en fil
- konflikter må løses før en kan fortsette
- Kan løses manuelt, men det er å foretrekke å bruke et dertil egnet verktøy

konflikter

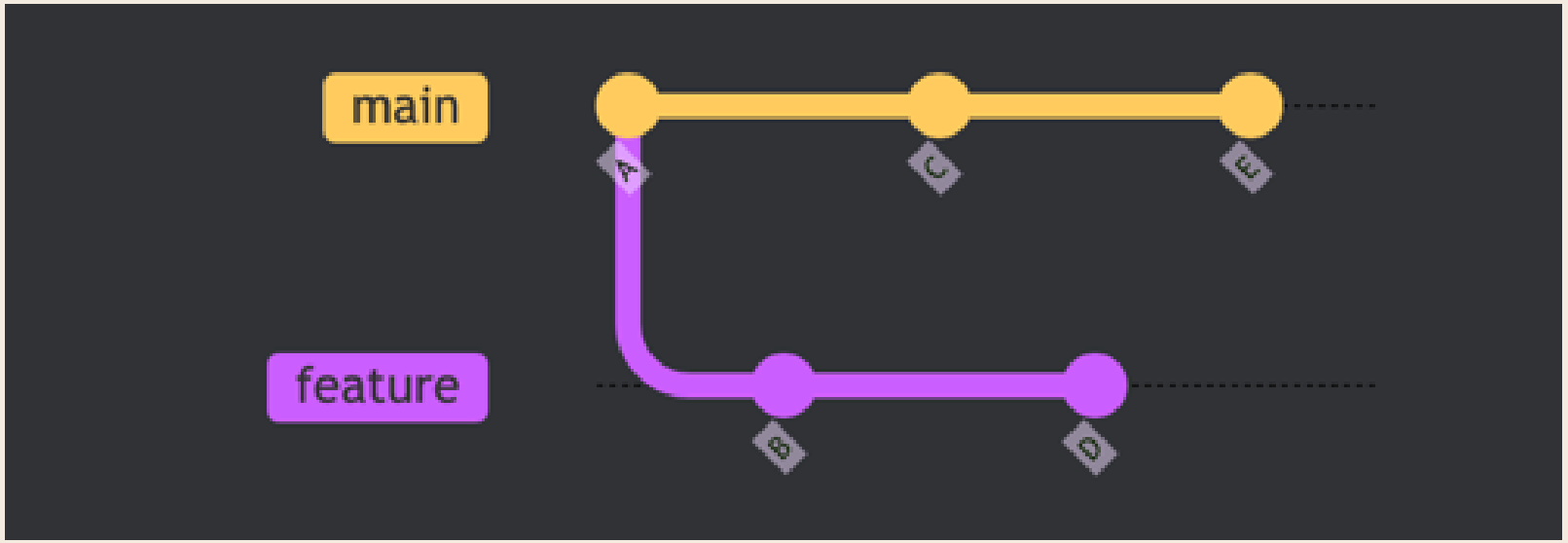
- I filen som har en konflikt, vil dette merkes på følgende måte:

```
<<<<<<< yours:sample.txt
Conflict resolution is hard;
let's go shopping.
=======
Git makes conflict resolution easy.
>>>>>>> theirs:sample.txt
```

Vanlig Arbeidsflyt

- Sjekke ut en 'feature branch'
- 'Committe' endringer etterhvert som utvikling skrider frem
- Om utviklingen pågår over et lengre tidsrom, bør en vedlikeholde historikken ved å enten gjøre merge eller rebase fra main-branch.
- 'Pushe' til `origin`
- Når utvikling er ferdig, opprette Pull Request og få utført 'review' av andre på teamet

Vanlig Arbeidsflyt

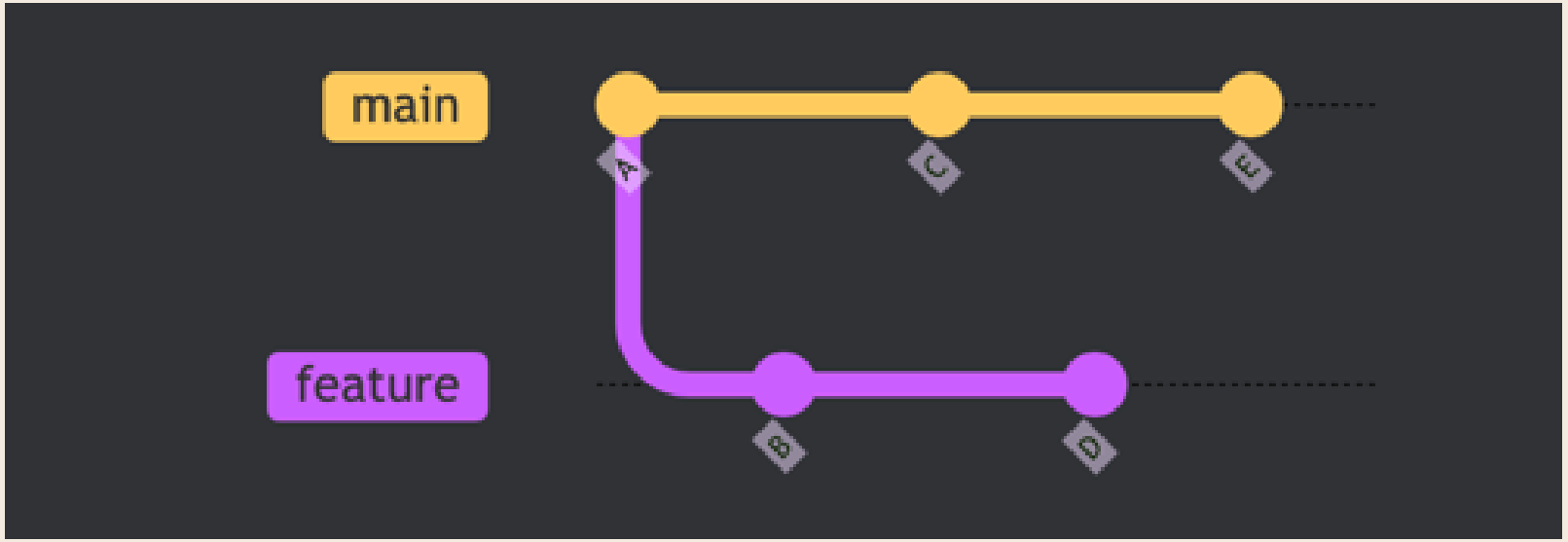


Vanlig Arbeidsflyt - merge eller rebase

- 'merge' gir en "tom" 'merge-commit'
- 'rebase' skriver om historien, men gir en mer lineær historikk
  - 'commits' fra feature-branch blir lagt til etter siste 'commit' på `main`
- Et tredje alternativ er 'squash'. Dette blir som en 'rebase', men alle 'commits' blir slått sammen til en.

Merge

- Med dette som utgangspunkt:

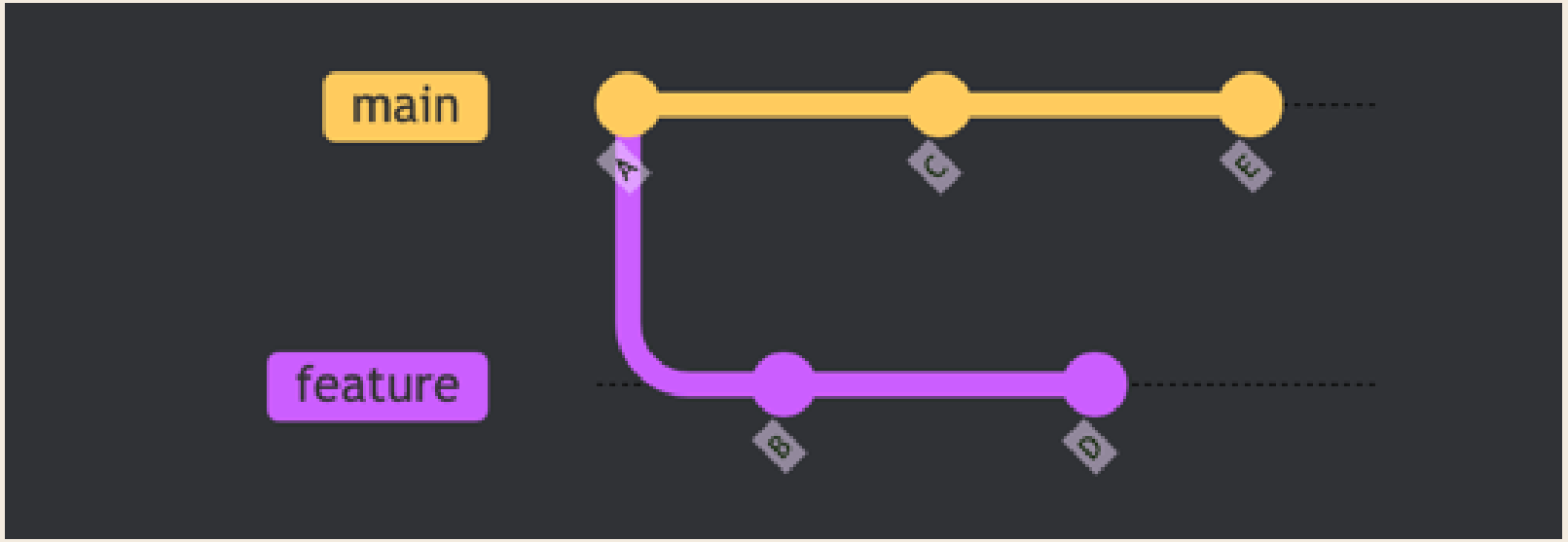


Merge

- ser en 'merge' slik ut:

Rebase

- Med dette som utgangspunkt:



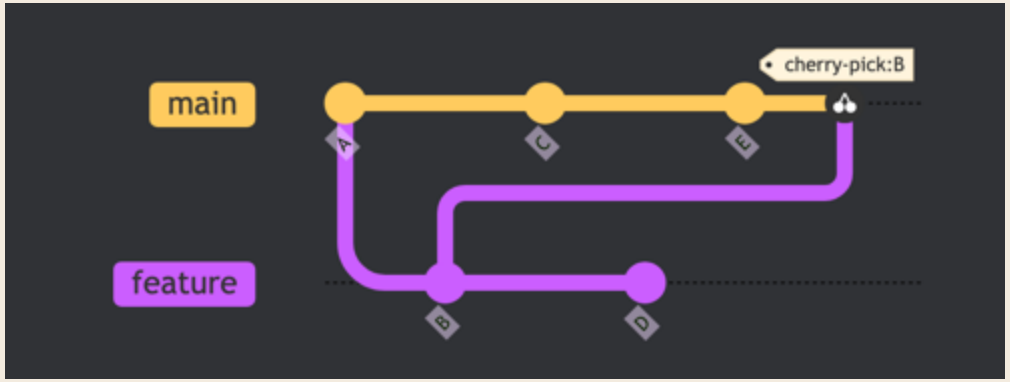
Rebase

- ser en 'rebase' slik ut:

- `B'` og `D'` vil ha fått ny hash da 'parent' er endret

Cherry-Pick

- 'cherry-pick' vil plukke en spesifikk 'commit' og legge den til gjeldende 'branch'
- `git cherry-pick 6b5b594ae6749fa1`



Interactive Rebase

- Gir mulighet for å endre på de 'commits' som inngår i en rebase
  - slå sammen 'commits' ('squash')
  - utelate 'commit'
  - endre melding
  - +++
- Eksempel: `git rebase -i HEAD~4`

Interactive Rebase

```
pick 34733a5 Oppdatert spec til versjon fra main branch
pick 4cb8ddd Fiks 1012 (#380)
pick b55983c FIKS-1008 Økt dybde på autorisasjons-query for å støtte brukere på underenhet
pick dac9767 FIKS-877 fjern bruker fra underenhet (#384)

# Rebase 381afda..dac9767 onto 381afda (4 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# ...
```

Force Push

- Operasjoner som `commit --amend` og `rebase` skriver om historien
- De vil derfor ikke alltid kunne 'pushes' på vanlig vis, men kan kreve 'forced push'
- `git push -f origin feature` vil kunne være OK
- 'Forced push' på `main` branch vil sjelden være nødvendig, og bør unngås

Workshop

https://github.com/RasmanTuta/git-intro-workshop