Mentimeter

Datorer och Datorsystem

Föreläsning 3 - Git

11

Uppföljning

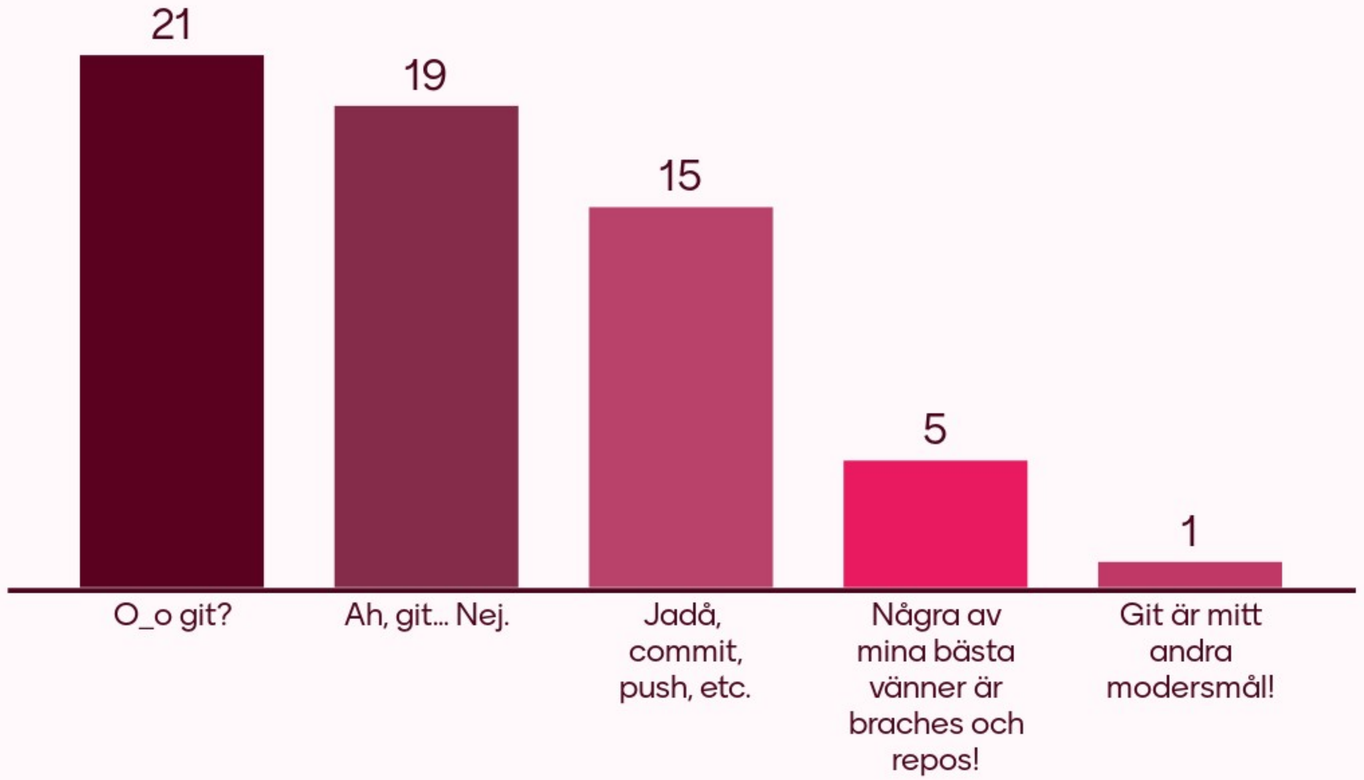
→ Labbar

→ Uppsamlingslaboration

→ CSN, behöver poäng kvickt? Mejla mig.

→ Andra frågor?

Tidigare erfarenhet av git?



Vad är versionshantering?

Hjälper oss att hantera komplexiteten det innebär att hålla reda på alla filer som hör till ett projekt – källkod, dokumentation, etc.

Låter oss gå tillbaka till äldre versioner av filerna om nödvändigt.

Gör det möjligt för oss att arbeta parallellt med filerna i projektet utan att komma i vägen för varandra.

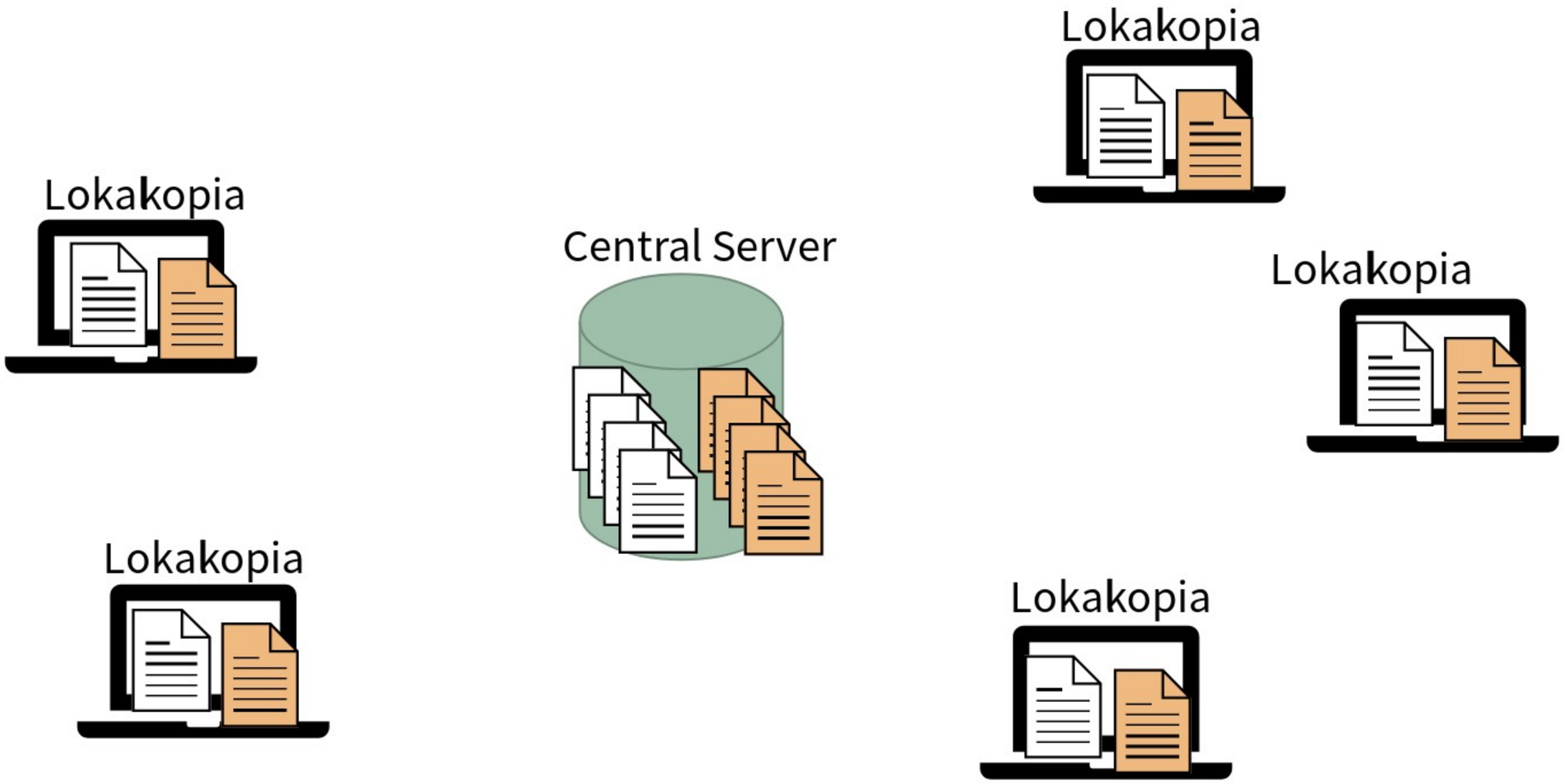
Versionshanteringssystem

Programvara som lagrar alla versioner som har existerat under projektets gång i någon form av databas.

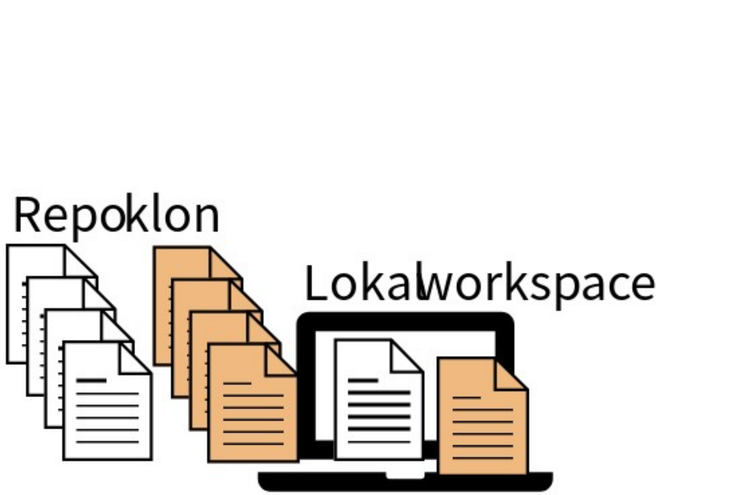
Centraliserade system: Subversion (SVN), CVS, etc. En central databas, ett repositorium (repo), håller reda på alla filer. Utvecklare *checkar ut* en lokal kopia av filerna och arbetar med dem. När arbetet är klart checkar utvecklaren in filerna igen (kopierar dem till det centrala repot) varvid en ny version av projektet skapas.

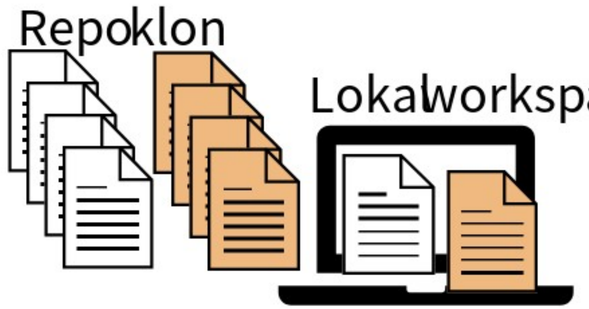
Distribuerade system: Git. Finns inget centralt repo. I stället klonar man ett repo (gör en komplett kopia av repot). Ändringar kan hämtas (pull), eller överföras till (push), andra repon.

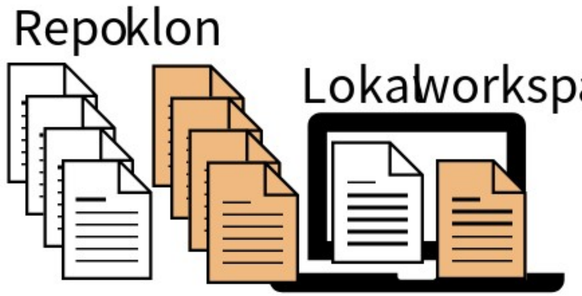
Centraliserad versionshantering

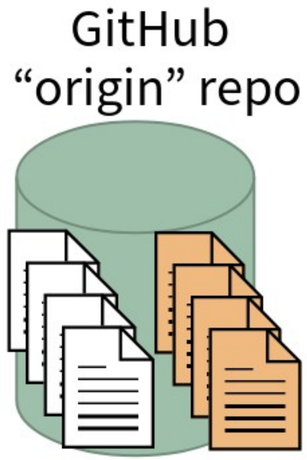


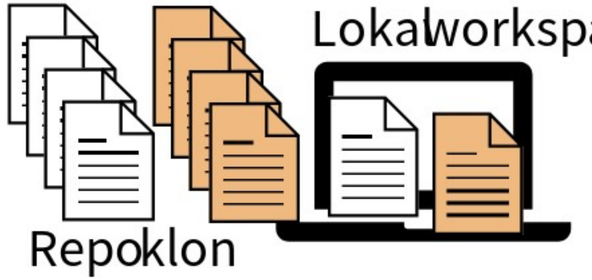
Distriburead versionshantering











Om Git

- Skapad av Linus Torvalds, Linux skapare, 2005, för att hantera utvecklingen av Linuxkärnan.
- Mål
  - Snabbhet
  - Stödja massivt parallell utveckling (tusentals samtidiga aktiviteter)
  - Helt distribuerat
  - Kunna hantera stora projekt effektivt

Git och GitHub

Git är en programvara med vars hjälp man kan skapa nya repon, klona ett repo från en annan dator (via nätet), arbeta med det på sin lokala dator samt hämta/överföra ändringar.

GitHub är en webbtjänst (bland flera andra liknande) som kan användas för att lagra git-repon samt utföra git-operationer på dem. Gratis att använda.

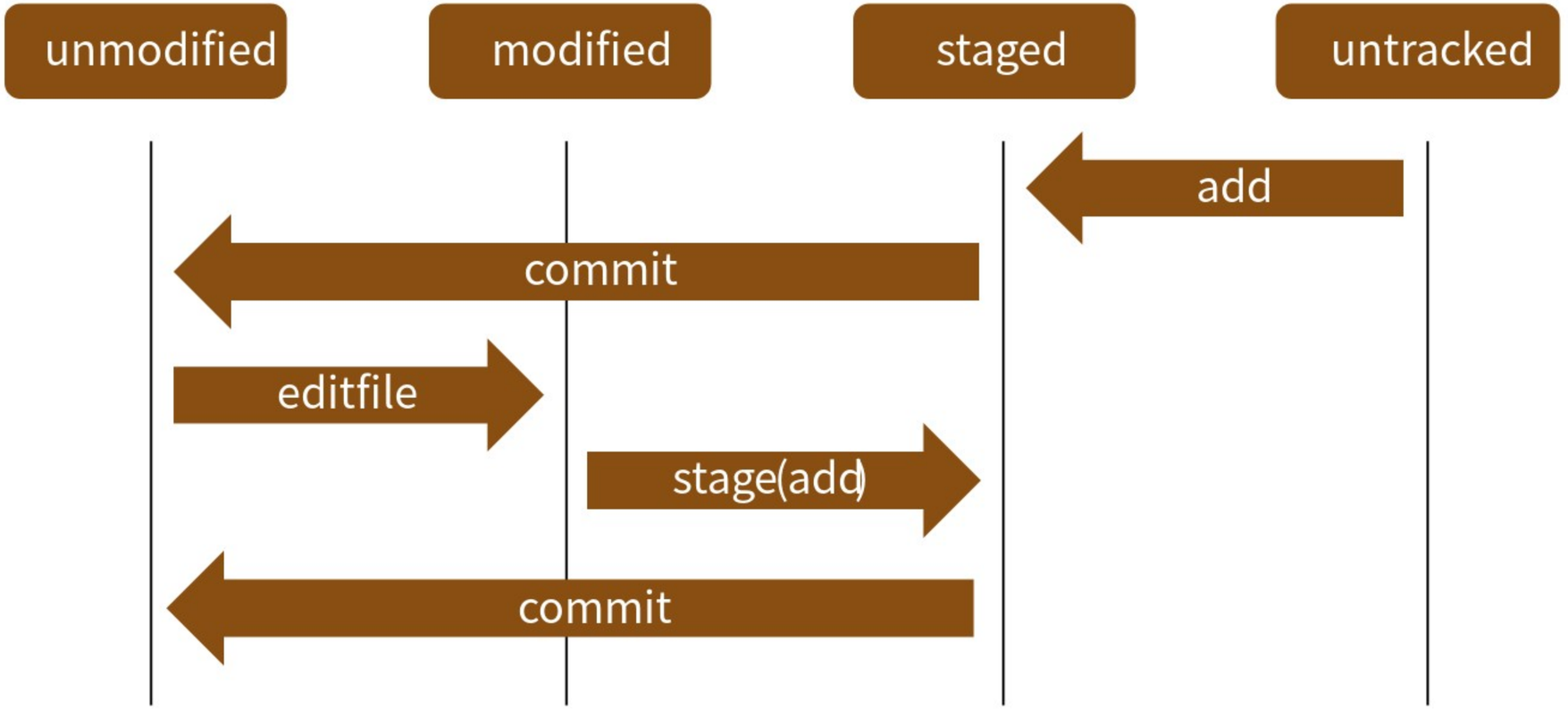
Beroende på ett projekts komplexitet så kan arbetsprocessen se olika ut. Laborationen visar hur Git och GitHub kan användas i ett mindre projekt (t.ex. för en inlämningsuppgift) med ett fåtal deltagare.

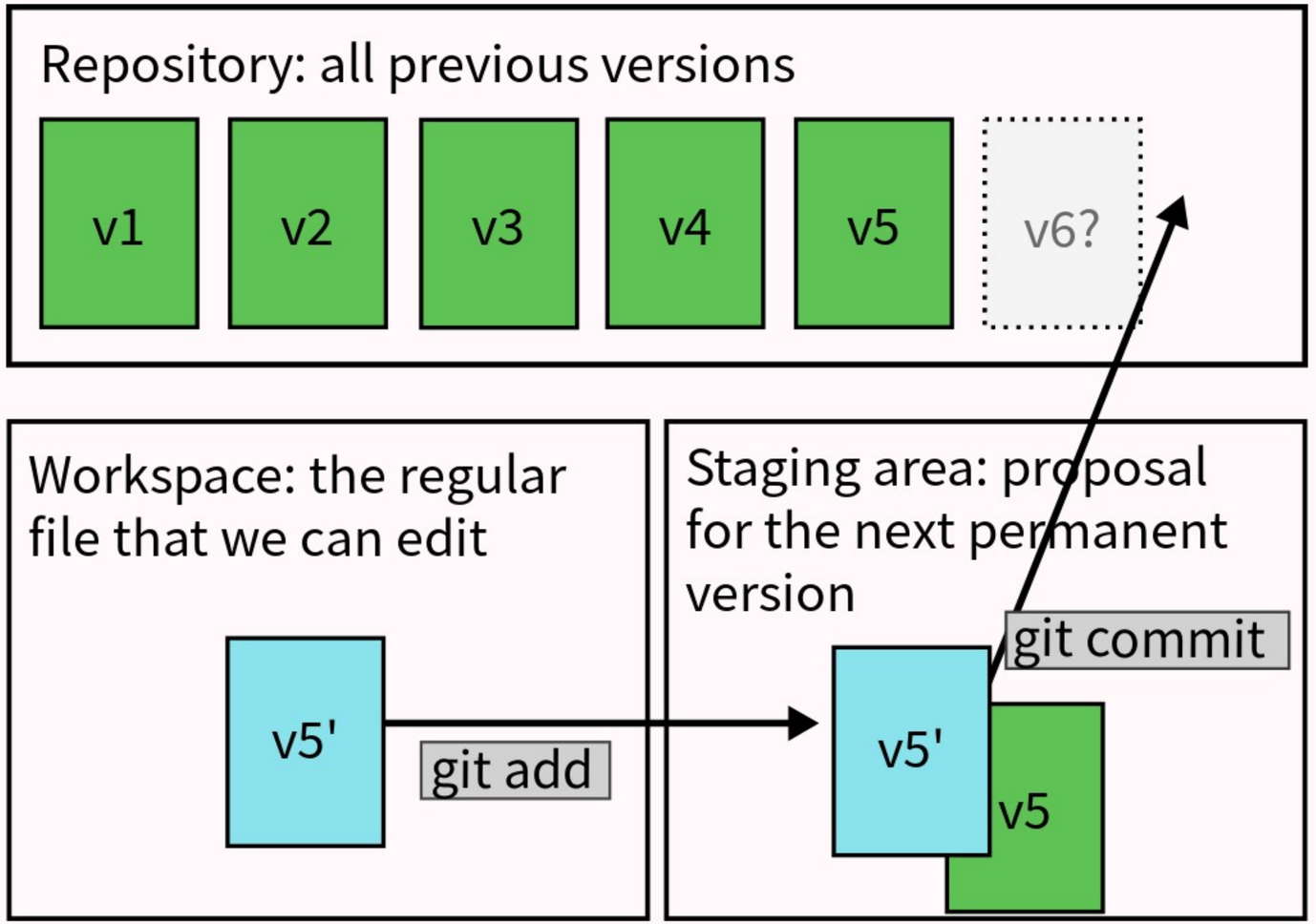
I kommande kurser, t.ex. EDAF45 Programvaruutveckling i grupp , kommer ni att stöta på mer avancerad användning av Git.

Arbeta med ett lokalt repo

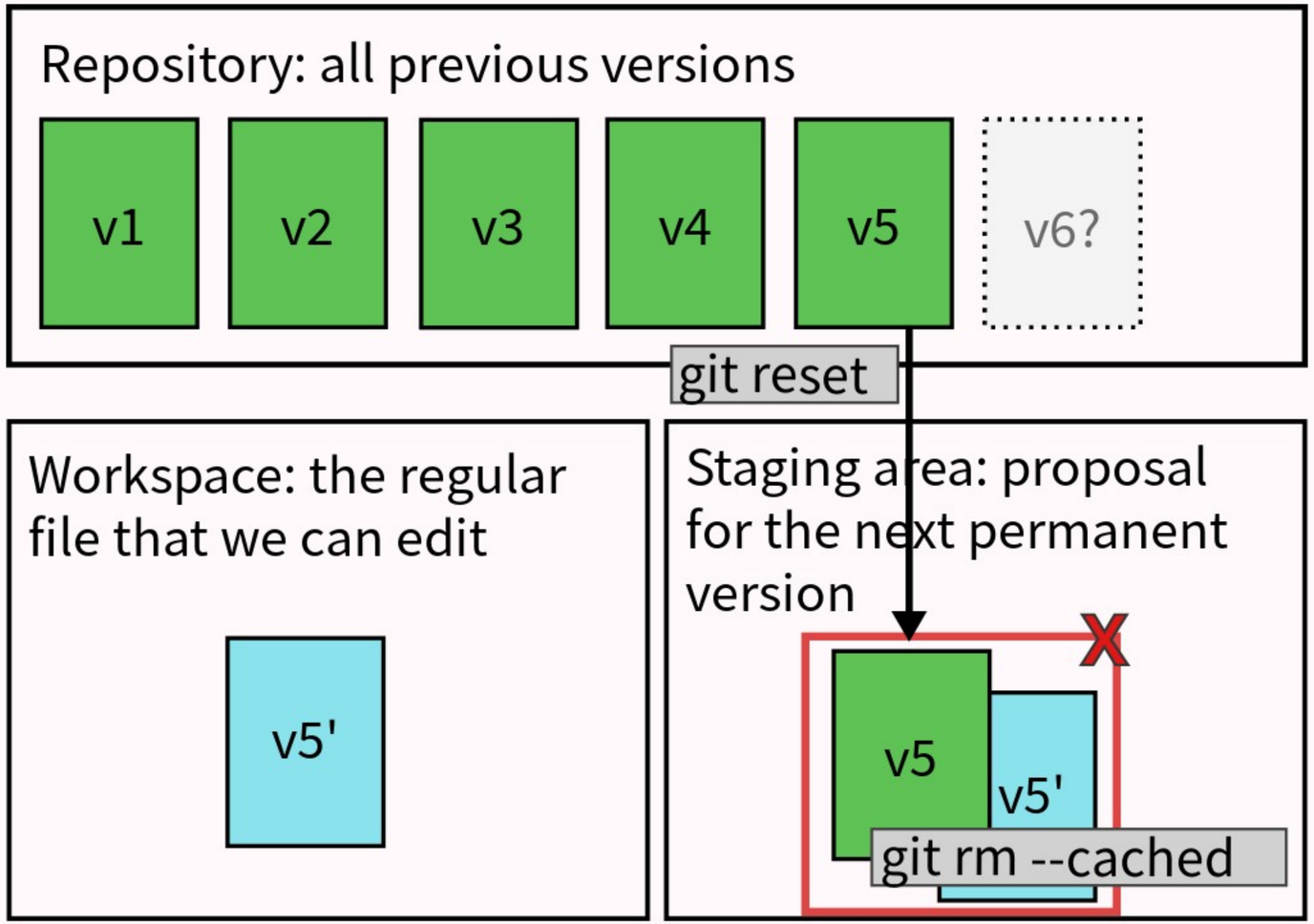
1. Skapa ett repo på din egen dator (eller klona ett existerande)
2. Lägg till eller modifiera filer. Filerna är nu modifierade.
3. När du har något som fungerar kan du skapa en ny version av projektet.
    a. Välj ut vilka (oftast alla) av de modifierade filerna som ska ingå i den nya versionen av projektet. De är nu utvalda (eng. staged).
    b. Skapa den nya versionen genom att göra en s.k. commit (svensk term?). Nu lagras en ny uppdaterad version av alla filerna i repots databas.
4. Upprepa från punkt 2.

Filers livscykel





add / commit



remove / reset

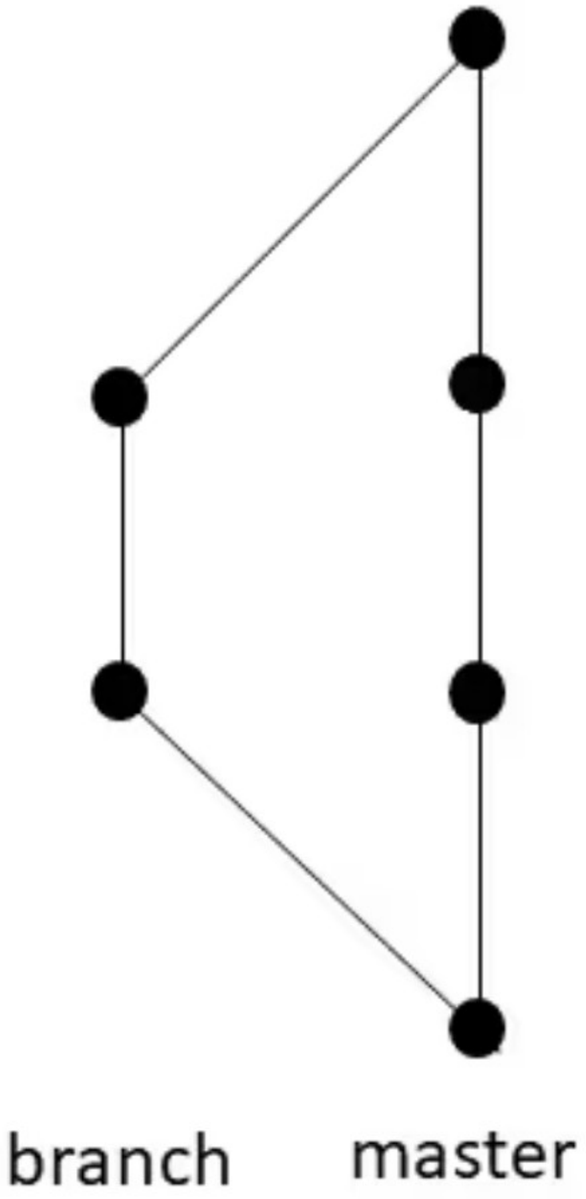
När jag nu hör git, tänker jag på...

47 responses

Commit

Varje commit skapar en ny ögonblicksbild (snapshot) av hela projektet som vi kan återvända till om nödvändigt.

Varje commit refererar till sin föregångare, vilket ger upphov av en "länkad lista" av versionshistorik.



Grenar (branches)

Man kan skapa parallella utvecklingslinjer som sedan sammanfogas igen (merge).

Varför vill man göra det?

Vilka problem kan uppstå?

Merge

→ När två grenar slås ihop kan konflikter uppstå om ändringarna i de två grenarna är motstridiga.

→ Ofta klarar Git av att själv göra en merge, men ibland kan manuellt ingripande vara nödvändigt.

→ Efter en merge, testa att allt fungerar som det ska och gör commit.

Merge-exempel 1

```
object MyApp {
  def main(args: Array[String]): Unit = {
    println("Hello World!")
  }

  def myFunction(): Unit = {
    println("This is the original fuction.")
  }
}
```

```
object MyApp {
  def main(args: Array[String]): Unit = {
    println("Hello World!")
  }

  <<<<<<< HEAD
  def featureFunction(): Unit = {
    println("This is the main function.")
  }
  =======
  def featureFunction(): Unit = {
    println("This is branched off function.")
  }
  >>>>>>> feature-branch
}
```

Merge-exempel 2

```
def myFunction(): Unit = {
  println("Some initial code.")
  println("Create data structures.")

  println("TODO: Run algorithm on data.")

  println("Clean up.")
  println("Log and return result.")
}
```

```
def myFunction(): Unit = {
  println("Some initial code.")
  println("Create data structures.")

  <<<<<<< HEAD
  println("A simple but ineffective way...")
  println("to calculate the needed data...")
  println("that is long and difficult to read.")
  =======
  println("A simpler way to do the...")
  println("the same thing.")
  >>>>>>> branch-a

  println("Clean up.")
  println("Log and return result.")
}
```

Arbeta distribuerat

- Ett repo som ligger på nätet (t.ex. på GitHub) kan klonas till din lokala dator. Du får en komplett kopia av repot.
- Om repot som du klonade senare uppdateras, kan du hämta ned de nya ändringarna (**pull**).
- Om du har skrivrättigheter i det klonade repot kan du ladda upp dina lokala ändringar (**push**).
- Om inte, måste du istället skapa en **pull request** på GitHub. Repots ägare kan granska och acceptera din request.
  - Kallas ibland **merge request**, t.ex. på GitLab.

GitHub

- Gratis
- Lagring av personliga repon
- Kan utföra git-operationer via webbgränssnitt: Skapa, klona, uppdatera repon.
- Publika repon – lämpliga för open-source-programvara och sådant ni vill visa för andra, t.ex. blivande arbetsgivare.
- Privata repon – lämpliga för laborationer/inlämningsuppgifter och liknande. Undvik att andra stjäl era lösningar och lämnar in dem som sina!

Enkel arbetsprocess för litet projekt

1. Skapa ett repo på GitHub som gemensam lagringsplats som alla projektdeltagare har skrivrättigheter till.
2. Varje deltagare klonar repot till sin egen lokala dator.
3. Deltagarna arbetar med det lokala repot tills det innehåller något som de vill dela med sig av till övriga deltagare.
    a. Då gör de en push som överför ändringarna till repot på GitHub.
4. De andra deltagarna, när de vill ta del av ändringarna, gör en pull.

Merge vid push/pull

Om någon annan redan gjort en push när vi försöker ladda upp våra ändringar måste vi först integrera våra ändringar med de ändringar som gjorts i det centrala repot. Git upptäcker detta åt oss.

Det gör vi genom att göra en pull som hämtar ner den senaste versionen av projektet och sammanfogar det med våra ändringar (merge). Därefter kan vi göra push på nytt.

Vad händer nu?

- → Läs avsnittet i scalakompendiet (Björn Regnell) om Git.
- → Se avsnitt 1.1, 1.2, 1.3, 1.6 samt 1.7 i youtubeserien "Git and GitHub for Poets".
- → Läs kapitel 1, 2 och 3.1-3.2 i Pro Git. (Chacon/Straub).
- → Länkar till materialet finns på kurshemsidan. https://cs.lth.se/edaa60/datorlaborationer/

Laborationen

- På laborationen får ni arbeta er igenom ett scenario som visar hur ni kan använda Git och GitHub för ett litet projekt.
- Tänkt att kunna göras individuellt, men OK att sammarbeta, speciellt i den sista delen.
- Installera git på din egen dator.
  - Kort demo

Ask me anything

13 questions
21 upvotes