Utvecklingsmiljön

Mikael Svahnberg*

2023-11-21

1 Introduktion

- Thomas & Hunt; Kapitel 3: The Basic Tools
- Grunderna i Konfigurationshantering
- Installera och bli vän med din utvecklingsmiljö

$\sum$ Allt är text, allt skall konfigurationshanteras.

2 Vikten av Rätt Verktyg

- Det här är din karriär. Ditt liv. Gör det bekvämt för dig.
- Grundläggande ergonomi
  - Sitter du rätt?
  - Har du rätt avstånd till skärmen?
  - Tar du pauser tillräckligt ofta? Tillräckligt långa?
- På samma sätt: Vilka datorprogram använder du?
  - Manualer?
  - Informationssökning?
  - Samarbetsytor?
  - Kommunikation?
  - **Programmering**?
- Går det att använda dessa program effektivt?
  - Hjälper de dig, eller är de i vägen?
  - Är de tangenbordsstyrda eller behöver du ett pekdon?
  - Hur stor andel av skärmen tillägnas till
    - ditt arbete
    - dator-administrativ bråte?

*Mikael.Svahnberg@bth.se

3 Att Välja en Utvecklingsmiljö

Klassiskt *Editor War*: Vi or Emacs

- Båda har fokus på en sak: *Redigera text*
- Allting annat fick man förr lösa utanför
  - e.g. kompilator, byggsystem, debugger, mm.
  - Idag finns det stöd i utvecklingsmiljön för att använda de externa verktygen.
  - Men det är ofta bekvämt att använda en terminal

Moderna Alternativ

- IntelliJ
- Eclipse
- Atom
- VS Code / VS Codium

Det viktiga är inte vilken du väljer

- Skall kunna använda flera olika miljöer
- Skall kunna se förbi utvecklingsmiljön till det du faktiskt utvecklar

Med VSCode vill jag skriva ett program

- eller -

Med den här filen vill jag göra XX

4 Datorprogram är Text

- Den pragmatiska ansatsen: *Datorprogram är text*
- Alltså fungerar vilken texteditor som helst. Inklusive Notepad++
- Text förklarar för dig själv och andra utvecklare vad du vill att datorn skall göra.
- Text kommer alltid vara tillgängligt och läsbart
- Text kan vara strukturerat
  - HTML / XML
  - JSON
  - CSV
  - YAML
  - C++ / Java
  - Lisp

- Text kan vara mer eller mindre enkelt att filtrera och arbeta med på datorn.
  - *find all files that mention X*
  - *find all rows where the seventh column contains the following datum*
  - *find all entries where the social security number is XXX*
  - ...

5 Bli vän med ditt shell

- En terminal är ett program som kör ett shell.
  - Kan vara ett GUI-fönster
  - Kan vara inbäddad i din utvecklingsmiljö
  - Kan också vara textbaserat, t.ex. `tmux` , `screen` , eller `ssh`

Exempel på shell:

- `/bin/sh` , `/usr/bin/bash`
- `cmd.exe`
- powershell
- git bash

6 Shell för att arbeta med datorn

- Utforska filsystemet
- Leta efter filer
- Öppna, flytta, ta bort filer.
- Enkel filredigering
- Kompilera och länka din programkod
- Skapa och köra små program (*shell scripts*) för att upprepa arbetsflöden
- Koppla upp dig mot andra datorer
- Testa så du t.ex. får rätt svar från en webserver
- Inspektera filer så att de innehåller rätt information
- Installera program
- Hantera vilka processer som körs
- *Länka ihop många små program för att lösa en större uppgift*
  - unix *pipes*: `producer | consumer`

– Exempel: `ps -ef | grep emacs | grep -v grep | tr -s ' ' | cut -d ' ' -f 2`
  * `ps -efc` lista alla processer
  * `grep emacs` visa bara rader där det står `emacs`
  * `grep -v grep` ta bort alla rader där det står `grep`
  * `tr -s ' '` byt ut alla blanksteg till ett enda
  * `cut -d ' ' -f 2` använd blanksteg som kolumnseparator, spara fält #2

7 Lite blandade shell-kommandon

Aktivitet	Unix	Ms-DOS
Hjälp om ett kommando	man <command> info	help
lista filer	ls	dir
lista filer med mer information	ls -l	
byt katalog	cd	cd
byt katalog, lägg nuvarande på stacken	pushd	
byt tillbaka till förra katalogen	popd	
skapa en ny katalog	mkdir	mkdir
ta bort en (tom) katalog	rmdir	rmdir
visa namnet på nuvarande katalog	pwd	
visa en fil	cat/more/less	type/?/?
visa början/slutet på en fil	head/tail	
kopiera en fil	cp	copy
flytta en fil	mv	move
ta bort en fil	rm	del
skapa en tom fil (eller uppdatera datum)	touch	
byt rättigheter på en fil	chmod	
Sök efter en viss fil	find	find
Sök efter ett visst innehåll	grep	
visa alla miljövariabler	env	
skriv ut text	echo	echo

8 Att arbeta med Text

Saker som din utvecklingsmiljö redan kan, men vet du hur?

- Hoppa till nästa ord, rad, stycke.
- Hoppa till nästa syntaktisk struktur (sexp, block, ...)
- Indentera raden automatiskt
- Kommentera/avkommentera ett kodblock med ett enda kommando

- Ångra kommandon, ångra ångringen
- Dela fönstret i flera delar och hoppa mellan dem.
- Gå till en viss rad.
- Sortera rader
- Söka efter strängar och reguljära uttryck
  - Upprepa senaste sökningen / ersättningen
- Skapa flera markörer baserat på vald text och redigera på flera ställen samtidigt
- Visa kompileringsfel
- Köra tester

Välj din utvecklingsmiljö så att

- det här blir enkelt för dig.
- du själv kan lägga till och justera hur den fungerar.
- du kan göra det här utan pekdon!

9 Förändra Text

Exempel på behov:

- Byt ut alla kommatecken mot semikolon i en csv-fil
- Byt ut alla förekomster av "Mikael Svahnberg" till "the author" i 500 filer.
- Räkna alla förekomster av ordet "kibo" i 500 filer. Summera värdet i kolumnen efter.
- Bryt upp alla filer i block om X rader.
  - Gör om varje block till md5-summor, och
  - lista bara de block som förekommer mer än en gång.

Man *kan* lösa mycket av detta i sin editor

- särskilt om den stödjer makros
- det är sällan smidigt eller skalbart

I stället: *Tillbaks till ditt shell*

- `find` , `grep` , `cut` , `sed` , `awk`

1. Exempel på sed

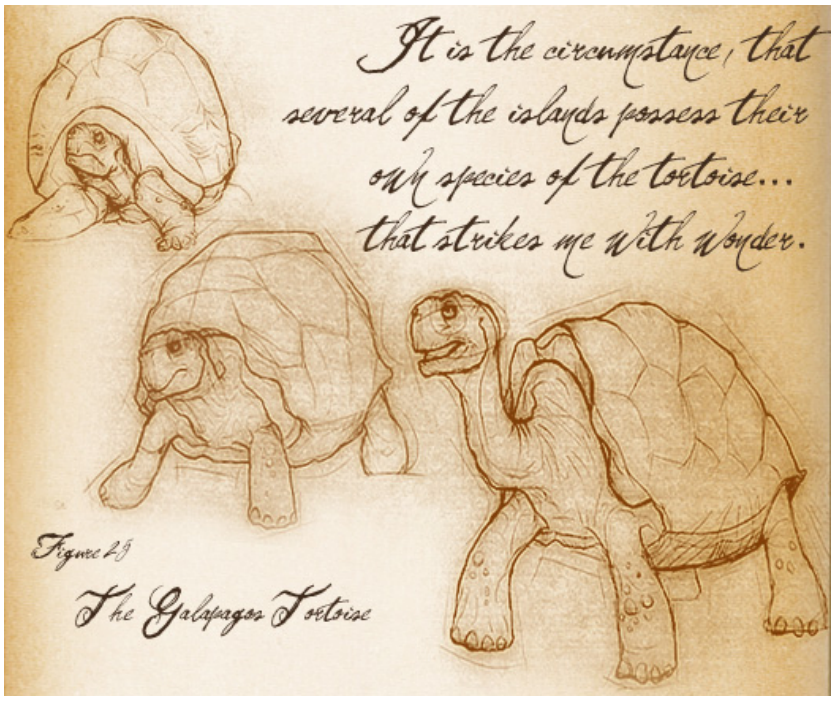
```
echo "---------- Bara för att visa att det finns en rad med ordet 'Introduktion'"
grep -HnC 3 Introduktion AA-Basic-Tools.org | head -6
echo "\n---------- Nu byter vi ut det mot något annat"
cat AA-Basic-Tools.org | sed "s/Introduktion/Översikt/" | head -20 | tail -10
```

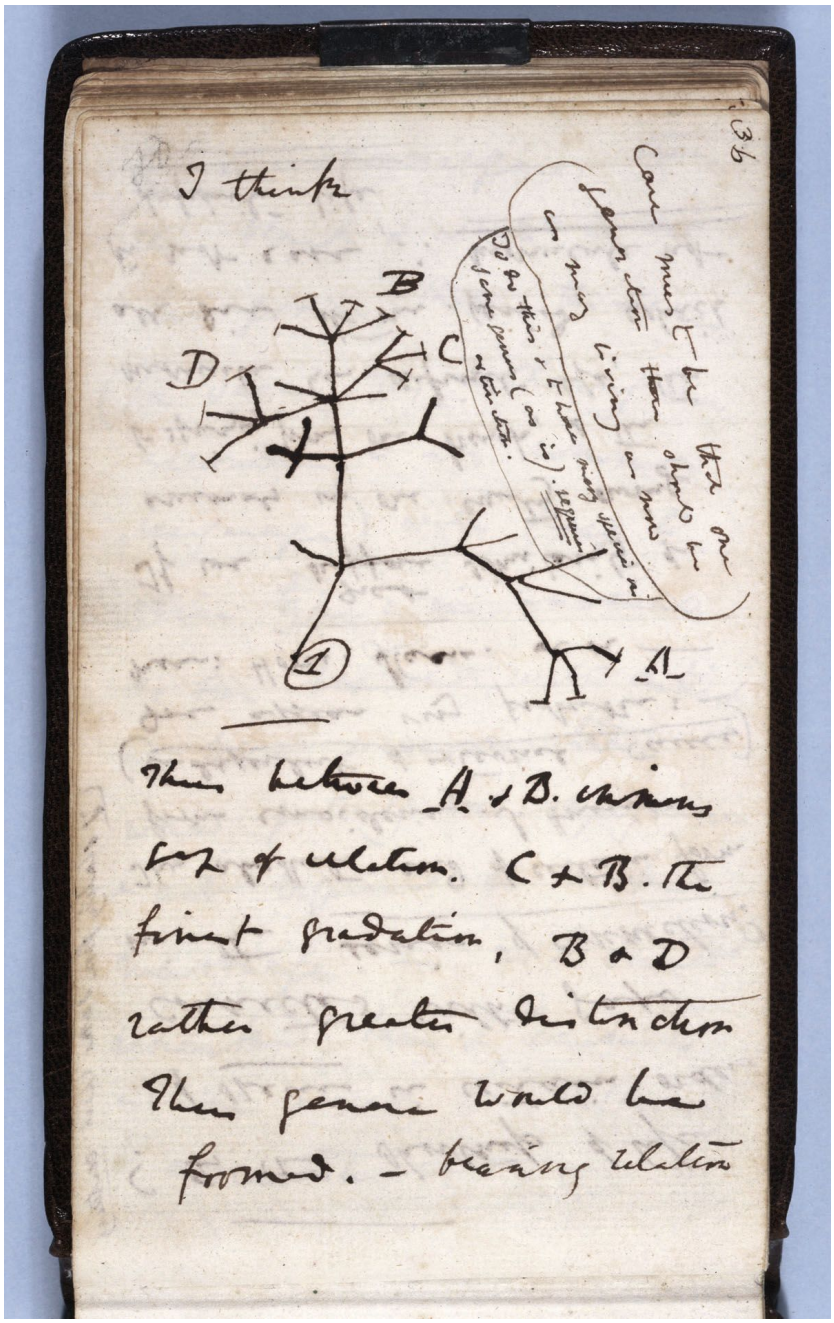
2. Exempel på awk

```
pwd
echo "Find all png files in the first lecture and print their sizes"
find ../01-Introduction -name "*.png" -printf "%s\n"

echo "... and use awk to sum this up"
find ../01-Introduction -name "*.png" -printf "%s\n" | awk 'BEGIN{print "Size is:"}{t
```

10 Ingenjörens Dagbok

- Vad sade ni under det senaste designmötet?
- Har du pratat med kunden? Vad sade ni?
- Har du löst ett knivigt problem?
- Har du en idé på hur du skall lösa ett knivigt problem?
- Påminn dig om något som måste göras
- Tvinga dig att reflektera över vad du faktiskt gör eller har gjort.
  - Använd helst papper och penna; det sänker farten och gör det möjligt att reflektera
  - ... och illustrera.
- Den varken behöver eller skall vara snygg och städad: Det är en förlängning av din hjärna.
- Jämför med lab-boken i Kemi/Fysik: *In case of fire: Save the Notebook!*
- Gamla tiders naturvetare, se utdrag ur Charles Darwin's anteckningsböcker nedan:


It is the circumstance, that
several of the islands possess their
own species of the tortoise...
that strikes me with wonder.
Figure 23
The Galapagos Tortoise



11 Konfigurationshantering

- *Versionshantering* kontra *Konfigurationhantering*
- Versionshantering:
  - En gigantisk undo-knapp för en hel fil.

  - Automatiskt skött, inte en massa idioti med att döpa om filerna efter vem som har ändrat dem.

- Konfifurationshantering sköter dessutom
  - Vilka versioner av filer fungerar ihop?
  - I vilka filer implementerar vi *feature X*
  - I vilka filer löser vi *issue Y* (Bug/feature request/...)?
  - Vilka versioner av alla filer är det som kör hos *Kund Z*?
  - Stöd för (semi-) automatisk *merge* om flera har ändrat filen

- Kända verktyg (idag)

  Subversion – Klassiskt server-orienterat: all CM sker på en och samma server

  Git – Används av de flesta idag
  – I teorin distribuerat; i praktiken också server-orienterat

  Mercurial – Liknar git, men anses av många vara lite enklare

12 Hela projektet skall alltid konfigurationshanteras

- Källkod
- Designdokumentation
- Dokumentation
- Script för att bygga applikationen
- Script för att ställa i ordning utvecklingsmiljön / testmiljön / produktionsmiljön
- Script för att starta applikationen

13 Använd Alltid Konfigurationshantering

- Även om det är ditt eget hobbyprojekt
- Inklusive konfigurationsfiler för att ställa i ordning din dator och utvecklingsmiljö
  - Svårt om allt ligger i windows registry
  - Lätt (och självklart) om allt finns i .rc-filer i din användarkatalog

If it's not comitted, it does not exist!

14 Det börjar med en Branch...

- Kärnan i all konfigurationshantering är att inse att du *alltid* arbetar på en gren
  - Du arbetar med filerna på en delad nätverkskatalog
  - Du kopierar projektet till din dator
  - Du klonar projektet
  - Du skapar en ny gren för att implementera en ny feature
- Utmaningen blir alltid att kombinera förgreningar med en *merge*
  - Om du är den enda som har redigerat en viss fil
  - Om du är den enda som har redigerat en viss del av en fil
  - Om ni är flera som har redigerat samma del av en viss fil
  - Om du har redigerat en fil som någon annan har tagit bort

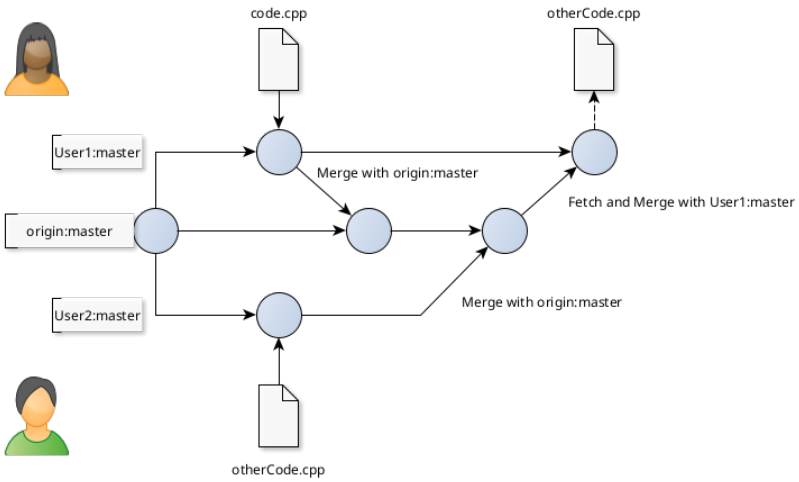
15 Commit

- Ofta behöver man redigera flera filer för att lösa en uppgift
- Paketera alla redigerade filer i en `commit`
- En commit har också ett meddelande som berättar vad man gjort.
  - Ofta länkat med krav-databasen eller issue-databasen
  - T.ex. Meddelandet `Fixes #222` kommer länka denna commit med issue #222.
- Den senaste committen kallas för *head*

Meddelandet skall vara kort men beskrivande

- inte bara "redigerade x.txt", utan *varför*, t.ex. "bakgrund om glasstilverkning.", eller "Implementerar krav #23"
- en commit == ett syfte. Flera syften, flera commits.

16 Samarbete i ett Repository



17 Flera Förgreningar/Branches

Man kan ha hur många förgreningar man vill, och skapa dem av vilka anledningar som helst.

Ett strukturerat exempel på hur man kan arbeta är *Git Flow*-modellen:

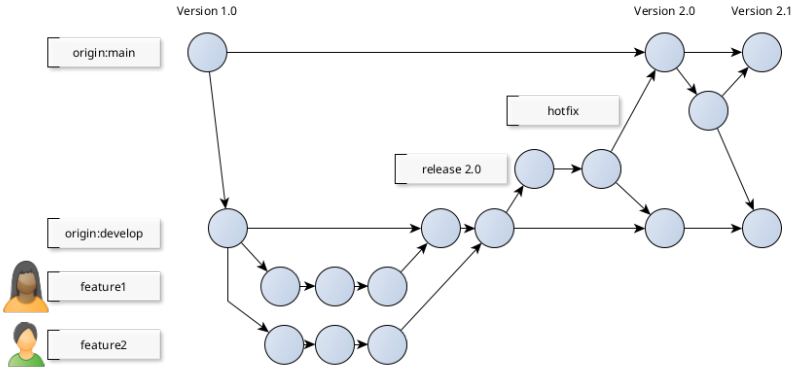
main Håller bara "stabila" releaser. Man arbetar inte på den här förgreningen.

develop Integration av features. Ingen nyutveckling här, utan bara mergearbete.

feature En per feature man arbetar med. Ny feature == ny förgrening.

release Fixa det sista innan man kan släppa nästa större release.

hotfix Om man absolut måste fixa något *just nu* i den senaste produkten.



18 Att Hantera Merge

- Att kombinera olika förgreningar är svårt.
- Moderna konfigurationshanteringsverktyg har slutat försöka.
  - De varnar användaren och vägrar fortsätta tills dess konflikten är fixad.

Strategi 1 Ha en egen förgrening, t.ex. för en viss feature, där färre utvecklare är inblandade.

Strategi 2 Ha en egen förgrening av feature-förgreningen (local branch).

Men till slut räcker inte detta, utan du behöver kombinera grenarna.

- Din lokala gren med gruppens
- Feature-grenen med utvecklingsgrenen
- Utvecklingsgrenen med main-grenen
- I distribuerade CM-verktyg som `git` kan du också hämta en förgrening från en annan utvecklare.

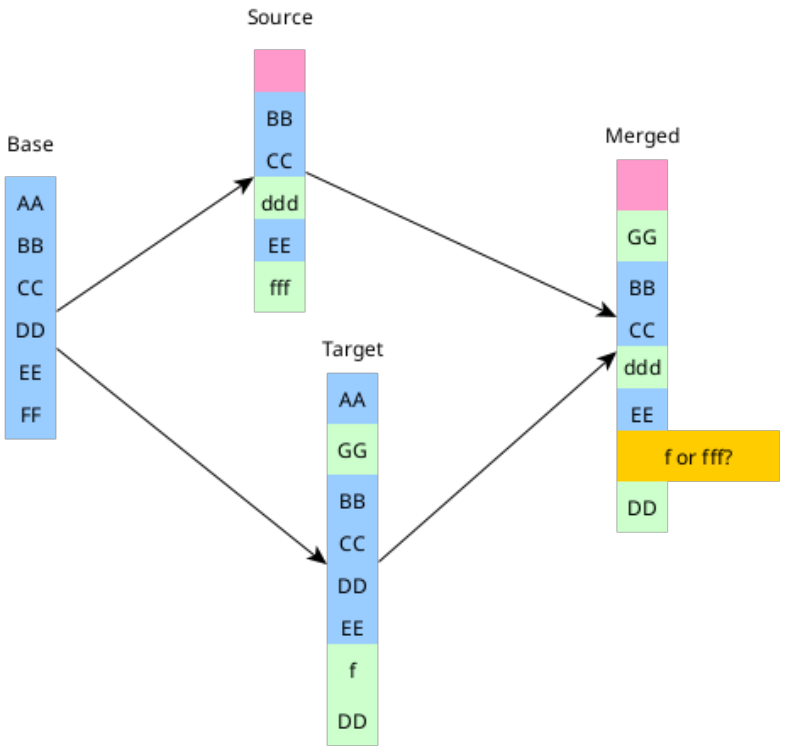
Förr eller senare kommer du då behöva hantera konflikter mellan grenarna.

19 diff

```
cd /tmp
echo "aaa" > first.txt ; echo "aaa" > second.txt
echo "bbb" >> first.txt; echo "BBB" >> second.txt
echo "ccc" >> first.txt; echo "ccc" >> second.txt
echo "ddd" >> second.txt

diff first.txt second.txt
```

20 Three-Way Merge

- Använder ett gemensamt ursprung för att avgöra om förändringar är nya eller inte.
- Skapar ibland lite nya problem
  - Vad händer t.ex. i figuren nedan om "GG" använder sig av "AA"?
  - t.ex. en `#include` , eller en referens i text *"som tidigare nämnt"*
- När båda filerna ändrats så måste användaren besluta (t.ex. `f or fff?`)



21 Merge med git

Antag: Du står i branchen `feature1`

- Ny fil: f1-klass.cc
- Ny fil: f1-header.hh
- Förändrad fil: context.cc

Först: Se till att allt är committat:

```
git add .
git commit -m "implements requirement #feature1"
```

Sedan: Genomför merge från `master` (Du "drar" alltid in de ändringar du vill ha)

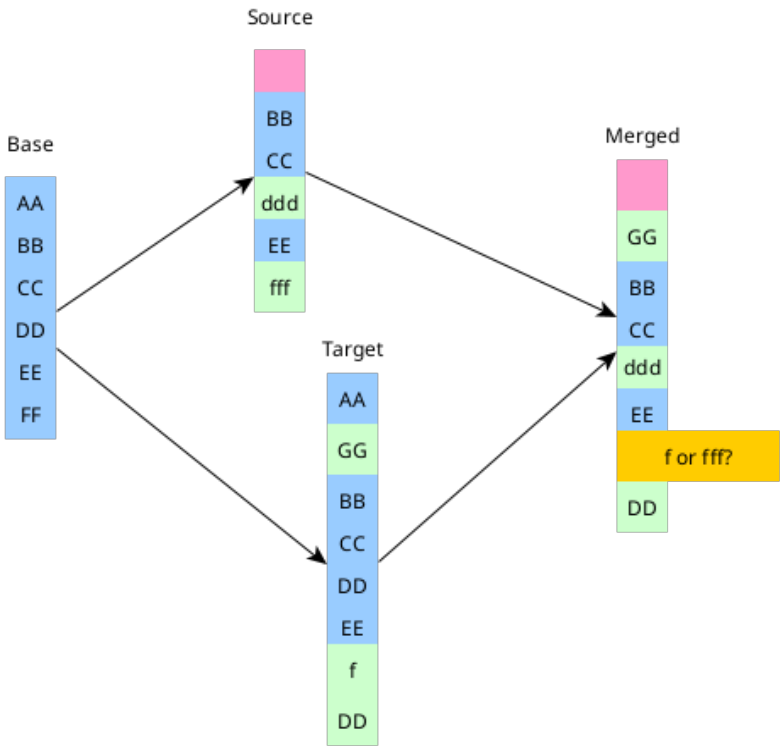
```
git checkout master
git merge feature1
```

22 Merge med konflikter

- Varning: Det här blir krångligt

- Merge-konflikter kommer hända
- Det är lika bra ni får ser vad man gör åt dem.

23 Samma exempel som innan



24 ... I olika grenar

`base` ligger i grenen `master`, `source` ligger i `src`, och `target` i `trg`:

```
$ git log --all --graph --abbrev-commit --decorate --oneline
 * efd01a3 (HEAD -> trg) trg
 | * 95b9019 (src) src
 |/
 * 107e415 (master) base
```

25 Försök med en merge

```
$ git status
On branch trg
...
```

```
$ git merge src
Auto-merging c.txt
CONFLICT (content): Merge conflict in c.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Vi kan alltid ge upp: `git merge --abort`, men hur kul är det?

26 Git markerar förändringarna

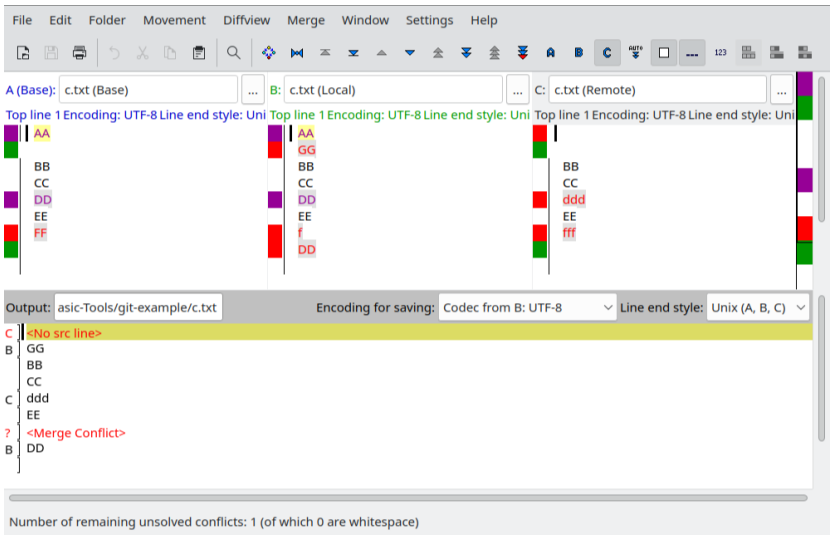
Låt oss se vad git har gett oss:

```
$ cat c.txt
<<<<<<< HEAD
AA
GG
=======
>>>>>>> src
BB
CC
ddd
EE
<<<<<<< HEAD
f
DD
=======
fff

>>>>>>> src
```

Blä! Vi behöver (önskar) ett verktyg för att kunna arbeta med detta:

```
git mergetool --tool=kdiff3
```

27 Lös konflikter



Slutresultatet är att jag har gjort min merge själv. `src` är nu jämkad med `trg` .

- Jag behöver alltså inte köra git merge igen
- `master` har inget som inte finns i `trg`, så den visas som samma gren. Just nu i alla fall.
  - Om jag gör en merge i `master` från `trg` så kommer bara `master` snabbspolas fram till `HEAD`.

```
$ git log --all --graph --abbrev-commit --decorate --oneline
 *   9b00fb9 (HEAD -> trg) merged
 |\
 | * 95b9019 (src) src
 * | efd01a3 trg
 |/
 * 107e415 (master) base
```

28 Grundläggande kommandon

29 git init – Starta ett tomt repository

Helt nytt och enbart lokalt:

```
mkdir git-example
cd git-example
git init
```

30 git clone – klona ett existerande repository

Hämta ett existerande projekt:

```
git clone git@github.com:mickesv/MeanStreets.git /tmp/ms
ls -l /tmp/ms
```

31 git status – Var är jag, vad behöver göras?

```
echo "hello" > h.txt
echo "world" > w.txt
ls -l
cat *

git status # Kolla vad git anser om det vi har såhär långt
```

32 git add – Skapa en Commit, steg 1: The staging area

- Staging area är ett område där man kan "plocka ihop" allting som man vill skall hänga ihop i en commit
- Man kan göra ganska komplexa saker här (lägga till en del av en fil, t.ex.)
- oftast handlar det bara om att man skall välja ut filerna.

```
git add h.txt
git add w.txt
git rm --cached w.txt # Jag ångrade mig, den här skall inte med än.
# git add --all   # lägger till alla filer som inte är tillagda än.

git status
```

33 git commit – Skapa en Commit, steg 2

- Nu är vi redo att committa allt som vi har i vår staging area.

```
git commit -m "Finally"
git status
```

34 git log – Kolla Loggen

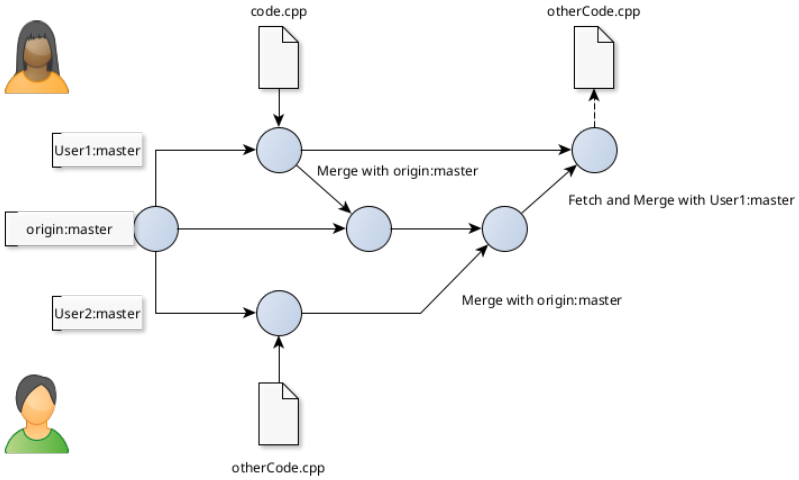
```
cd /tmp/ms
git log --all | cat -
# git log --all --graph --abbrev-commit --decorate --oneline | cat -
```

35 git stash – Lägg åt sidan när du blir avbruten

```
# Jag vill jobba med task 1
git branch task1
git checkout task1
echo "sss" > s.txt

# Kompisen kommer på besök och vi skall jobba på task 2
# Stoppa undan det jag höll på med:
git add s.txt
git stash -m "jobbar på task1"

# Skapa en ny branch för task2 och jobba på
git branch task2
git checkout task2
echo "eee" > e.txt
ls -l
git add e.txt
git commit -m "task2 klar"

# Nu är vi klara. Kompisen har gått, och jag kan fortsätta med task1
# Byt tillbaka till rätt branch, och plocka fram arbetet igen
git checkout task1
git stash pop

# Jobba op
ls -l
git status
echo "vvvv" > s.txt
git status

# Nu är jag klar, dags att committa.
git add s.txt
git commit -m "task1 klar"
```

36 Arbeta mot en Server

- Git är konstruerat för att vara distribuerat
- I praktiken arbetar man nästan alltid mot en server ändå
- *merge with origin:master* har sitt eget kommando: `git push`
  - I alla övriga fall så *drar* man in förändringarna till den branch man är i
  - I det här fallet så *trycker* man (push) upp til servern.



37 Arbetsflöde

```
# Download the project
git clone https://codeberg.org/mickesv/gitex.git

# Enter the directory and make some edits
cd gitex
git checkout -b testar
echo "bb" > b.txt
git add b.txt
git commit -m "added b"

# Merge back to main
git checkout main
git merge testar

# push to server
git push
```

38 När git push inte funkar

- t.ex. när någon annan gjort en push under tiden

```
$ git push
Username for 'https://codeberg.org': mickesv
Password for 'https://mickesv@codeberg.org':
To https://codeberg.org/mickesv/gitex.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://codeberg.org/mickesv/gitex.git'
```

39 git fetch – hämta vad som är nytt från `origin`

```
# Hämta allt nytt men gör ingenting annat
$ git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 277 bytes | 277.00 KiB/s, done.
From https://codeberg.org/mickesv/gitex
   07c6db7..497cb6d  main       -> origin/main

$ git branch --all
 * main
   testar
   remotes/origin/main      # <- Här ligger allt som hämtades
```

- Det rekommenderas att använda `git fetch`
- Ger dig tid att studera vad som har hänt i `origin/main`
- Kan till exempel jämföra grenarna: `git diff main origin/main`
- När du är klar använder du `git merge origin/main` följt av `git push`

Genväg Det finns också `git pull`

- Ungefär en `fetch` följt av en `merge`
- Löser nästan alla problem – men inte alltid
- `git fetch` ger dig möjlighet att verifiera att allt kommer bli rätt.

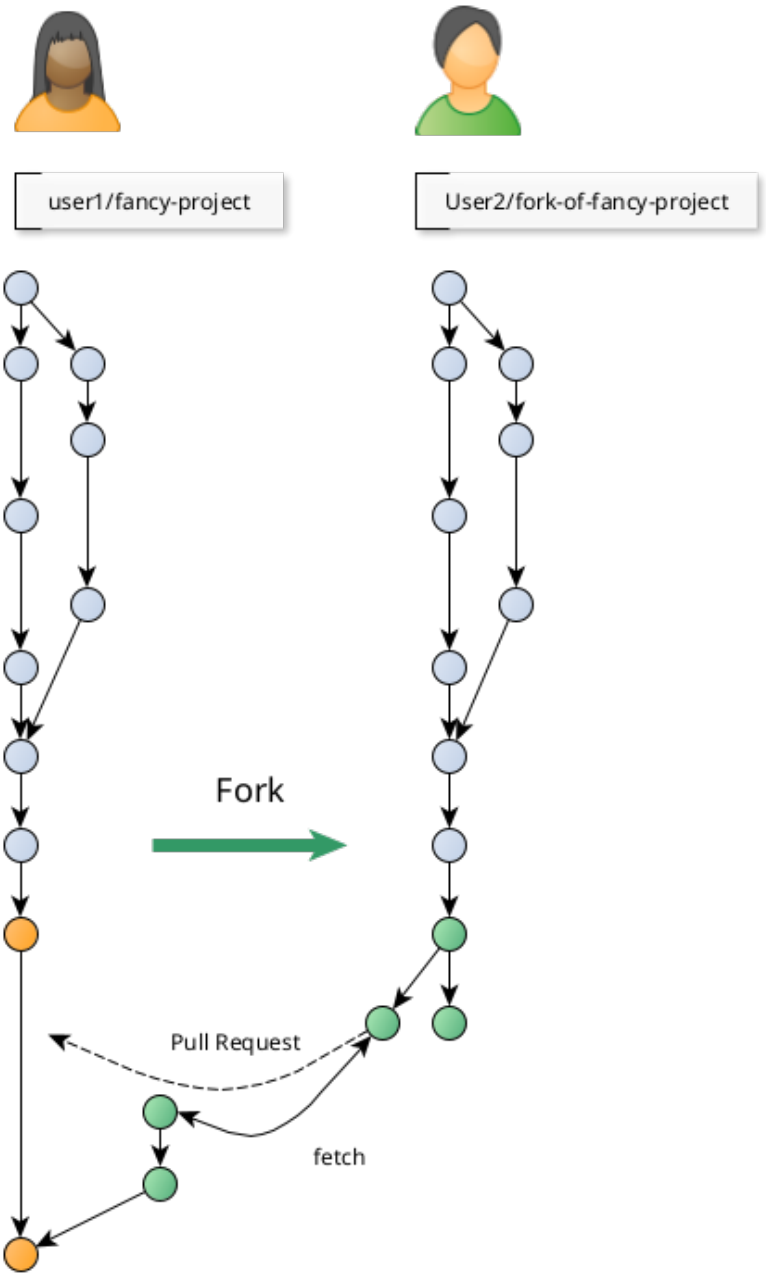
40 Resultat av git fetch/git merge

```
$ git log --all --graph --abbrev-commit --decorate --oneline
 *   e644bdf (HEAD -> main, origin/main) Merge remote-tracking branch 'origin/main'
 |\                                  # 2. merge, så att din main är samma som origin/main
 | * 497cb6d c.txt                   # 1. Det här skedde någon annanstans; fetch tar hem de
 * | 2a655d9 addded d
 |/
 * 07c6db7 (testar) added b
 * f28fad6 added a
```

- Du **tror** bara att du är i synk med `origin/main`
- Du *vet* inte förrän du faktiskt gör din `git push`
- Det här kan bli ett problem om branchen är väldigt aktiv.

41 Fork av ett projekt

- Specifikt för vissa github-servrar
- När man vill bidra till ett projekt så börjar man med att skapa sin egen kopia, en `fork`
- Man arbetar sedan som vanligt.
- När man är nöjd skapar man en `pull request` i originalprojektet
- Ägaren av originalprojektet kan då inspektera förändringarna och (om de godkänner dem) göra en `merge` med sitt projekt.
- Båda två kan fortsätta arbeta med sina respektive kopior av projektet. De behöver inte förena dem.



42 Sammanfattning

Allt är Text

- Att programmera handlar om att skriva text
- Att arbeta med en dator handlar om att ge den kommandon

- Att arbeta effektivt med en dator handlar om att kunna automatisera arbetsflöden
  - Alltså skriva små program
  - Alltså att skriva text
- Många gånger har man inte ens tillgång till mer än en textbaserad kommandoradstolk
- $\sum$ Lär dig hantera text!
- $\sum$ Välj en utvecklingsmiljö som gör det enkelt att hantera text.

Konfigurationshantering

- Att programmera är att samarbeta
- Man utvecklar inte linjärt. Man gör fel, man vill gå tillbaka, man vill prova olika vägar.
- Det krävs kraftfulla verktyg för att hänga med när alla i ett projekt utvecklar icke-linjärt.
  - Verktyg för konfigurationshantering
- Vanligast idag: `git`
- Största utmaningen:
  - Att ostört kunna arbeta i din egen förgrening
  - Att kunna jämka samman din kod med resten av projektet med hjälp av en `merge`.

Använd Alltid Konfigurationshantering. Använd Konfigurationshantering till Allt. Lär dig mantrat: *"If it's not committed, it does not exist!"*

43 Nästa Föreläsning: Testing and Debugging

- Thomas & Hunt: **Kapitel 7: While you are Coding**
- Thomas & Hunt: Kapitel 3; topic 20: Debugging
- Thomas & Hunt: Kapitel 9; topic 51: Pragmatic Starter Kit

44 Övning: Kom igång med din IDE Practice

- Starta `IntelliJ IDEA` eller den IDE du har valt (e.g. `VS Code` eller `VS Codium`) .
- Skapa ett Projekt `New Project`
  - Name: `TestProjekt`
  - Language: `Java`

- `Create`

Vad ser du nu? Vad ser du i kod-fönstret? Vad betyder symbolerna?
Hur kör du programmet? Vad händer?
Hur kör du programmet i debug-läge? Vad händer nu? Hur går du vidare?

1. Om du använder VSCode/VSCodium
   - "New Project" finns inte; bara new file
   - Skapa en fil `main.java`, lägg den i någon lämplig katalog
   - Klistra in:

```
public class Main {
    public static void main(String[] args) {
        System.out.printf("Hello and welcome!");

        for (int i = 1; i <= 5; i++) {
            System.out.println("i = " + i);
        }
    }
}
```

45 Redigera Main.java

- Lägg till en metod i klassen, `public static int addTen(int x)`
  - Skriv metoden ovanför main-funktionen
  - Den skall lägga till 10 till `x` och returnera detta
- Lägg till raden `System.out.println("i+10 = " + addTen(i));` inuti main-funktionen.
- Kör programmet nu. Vad händer?

IntelliJ försöker hela tiden ge dig ledtrådar på vad du kan skiva.

- Hur ser detta ut?
- Vad får du för hjälp?
- Hjälper det faktiskt, eller är det i vägen?

46 Hjälp

- IntelliJ
  - Ställ dig någonstans där det står `println` och tryck `ctrl-Q` (Quick Documentation)
    - Du kan också komma åt detta via `View` -menyn. Vad mer kan du hitta där?
- VSCodium: `Ctrl-Shift-P`

47 Lägg till en ny klass

- `File/New/Java Class`
- Name: `Nummer`
- Skriv in följande kod efter rad 1:

```
private int myNumber=0;

public Nummer(int x) {
  myNumber = x;
}

public boolean smallerThan(int x) {
  return (myNumber < x);
}

public int get() {
  return myNumber;
}

public String toString() {
  return "Number: " + myNumber;
}
```

48 Använd den nya klassen

- Byt tillbaka till `Main.java` och lägg till följande kod efter rad 12:

```
Nummer num = new Nummer(10);
System.out.println("Is "+ num + " smaller than 12: "
                   + num.smallerThan(12));
```

Vilken "hjälp" får du när du skriver detta? Hur kan du köra det här programmet utan att sträcka dig efter musen?

Första raden av utskriften blir: `Hello and welcome!Is Number:  10 smaller than 12:  true`

- Hur kan du fixa så att du får en ny rad efter "welcome!"?
- Vad kan du göra för att bara skriva ut siffran 10 och inte "Number: 10"?

49 Gör editorn din

- Måste allt synas samtidigt? Hur gömmer du undan saker?
- Gillar du inte färgschemat? Hur byter du?
- Det finns gott om plugins. `File/Settings/Plugins`
  - Vilka vill du installera? Vilka är redan installerade?

50 Hitta Terminalen

- Det finns en inbyggd kommandoradstolk. Var då?
- Vad kan du göra med den?

51 Övning: Kom igång med Git PRACTICE

52 Registrera ett Konto

- Måste tyvärr börja med att registrera ett konto på någon server.
  - `https://github.com/signup` eller `https://education.github.com/pack`
  - `https://gitlab.com/users/sign_up`
  - `https://www.atlassian.com/software/bitbucket/bundle`
  - `https://codeberg.org/`
- Github är fortfarande väldigt stort för open source-projekt
  - Lite i blåsväder för hur de använder koden som du laddar upp dit
- Många migrerade över till Gitlab när Microsoft köpte Github
- Atlassian och Bitbucket har bra integration med deras övriga produkter.
  - Brukade vara väldigt generösa för studenter och universitet (numera vet jag inte)
- Codeberg.org är specifikt fokuserat på open-source-projekt

53 Skapa och klona ett Repository

- Lättast att börja i webgränssnittet
- Döp projektet till något kreativt, t.ex. `gitexempel`
- När du är klar bör du kunna hitta en länk, t.ex. under `<> Code` som du kan använda för att klona projektet
  - till exempel: `git clone https://codeberg.org/mickesv/gitex.git`
  - Det här sätter up `remote/origin` mm. åt dig.

54 Skapa lite git-historia

1. Skapa några filer
2. Lägg till dem till stashen och committa
3. Ändra någon av filerna; lägg till och committa igen

4. Upprepa några gånger
5. Skapa en branch
6. Skapa några filer, lägg till och committa.
7. Redigera någon av dina första filer, committa.
8. Kolla loggen
9. Kolla status
10. Pusha till servern
11. Kolla status

55 Forka en kollegas repository

1. Leta rätt på en kollegas konto (på samma server)
2. Välj ett repository och forka det (Lämpligen exempel-kontot som ni nyss skapade)
3. Klona ner det till din dator och skapa lite mer git-historia
4. När du har pushat allt till din fork, skapa en `pull request` hos deras repository (via webben)

56 Hantera en pull request

När din kollega har skapat en pull request mot ditt repo, hantera den:

- Inspektera commit för commit vad som är ändrat
- Går den att merga automatiskt? Det borde stå någonstans.
- Skapa en merge commit.

Skapa några fler commits i era respektive forkar

- Skapa en ny pull request
- Den här gången skall ni *neka* pull requesten.

57 Fler deltagare i samma projekt

- Dela in er i grupper om ca 5 personer
- Välj en kollegas repository
- Gå till `Settings/Collaborators` och lägg till fler av er på samma projekt.
- Klona repot

Nu får ni bara arbeta i en viss fil `charlie-foxtrot.txt`

- Ni får skriva ny text
- Ni får redigera texten som finns där
- Ni får stoppa in text: mellan två rader, och mitt i en rad.
- Ni får ta bort text

Committa regelbundet (max 2-3 ändringar per commit) Pusha efter varje commit

- Ni kan behöva göra en `fetch/merge` för att få göra en `push`

hantera merge-konflikterna
Diskutera i små grupper: Hur skall ni göra för att få färre konflikter?