TDDI82

Projektintroduktion, Make och Git

Eric Ekström

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1 Projektintroduktion
2 Projektplanering
3 Objektorienterad analys
4 Make
5 Git

LINKÖPING UNIVERSITY

Projektinformation

- Objektorienterat projekt i C++
- Oftast ett spel, men kan vara något annat
- 4 personer per grupp. Vi har slumpat fram grupperna.
- Fokus på att programmera. Vi har inga stora krav på dokumentation.

Projektinformation - kodkrav

- Objektorienterad kod i C++

Kompletta krav på kurshemsidan!

Projektinformation - kodkrav

- Objektorienterad kod i C++
- Minst tre polymorfa arvshierarkier

Kompletta krav på kurshemsidan!

Projektinformation - kodkrav

- Objektorienterad kod i C++
- Minst tre polymorfa arvshierarkier
- Grafik genom biblioteket SFML

Kompletta krav på kurshemsidan!

Projektinformation - kodkrav

- Objektorienterad kod i C++
- Minst tre polymorfa arvshierarkier
- Grafik genom biblioteket SFML
- Programmet måste bygga upp sitt tillstånd från en extern resurs. T.ex.
  - fil med positioner för väggar
  - slumpade hastigheter och positioner för fiender

Kompletta krav på kurshemsidan!

Projektinformation - övriga krav

- Presentera er projektidé för handledaren

Kompletta krav på kurshemsidan!

Projektinformation - övriga krav

- Presentera er projektidé för handledaren
- Minst ett möte med handledare

Kompletta krav på kurshemsidan!

Projektinformation - övriga krav

- Presentera er projektidé för handledaren
- Minst ett möte med handledare
- Git ska användas aktivt under projektet
  - Ni måste använda ett givet repository i `gitlab.liu.se`
  - Examineras individuellt!
  - Regelbundna “gitkontroller”

Kompletta krav på kurshemsidan!

Projektinformation - övriga krav

- Presentera er projektidé för handledaren
- Minst ett möte med handledare
- Git ska användas aktivt under projektet
  - Ni måste använda ett givet repository i `gitlab.liu.se`
  - Examineras individuellt!
  - Regelbundna “gitkontroller”
- Det ska gå att kompilera i SU-salarna

Kompletta krav på kurshemsidan!

Projektinformation - övriga krav

- Presentera er projektidé för handledaren
- Minst ett möte med handledare
- Git ska användas aktivt under projektet
  - Ni måste använda ett givet repository i `gitlab.liu.se`
  - Examineras individuellt!
  - Regelbundna “gitkontroller”
- Det ska gå att kompilera i SU-salarna
- Det ska finnas en Make-fil och kommandot `make run`

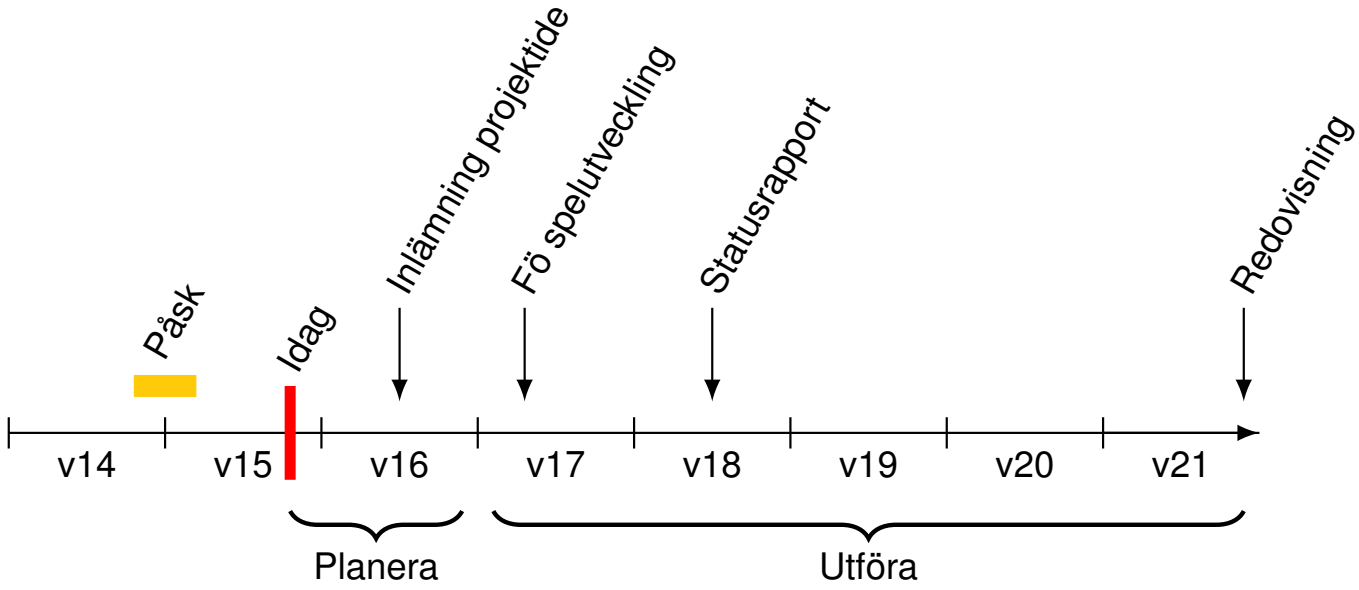
Kompletta krav på kurshemsidan!

Projektinformation - dokument

- Gruppkontrakt
  - För att främja gott samarbete
  - Underlätta konflikthantering
- Projektbeskrivning
  - kortfattad beskrivning av ert spel. Gör det tydligt vad som kommer ingå.
- Klassdiagram (UML-diagram)
  - En skiss som visar att ni tänkt igenom ert projekt.
- Individuell statusrapport

Mallar och krav finns på kurshemsidan.

Projektinformation - Tidslinje



1 Projektintroduktion
2 Projektplanering
3 Objektorienterad analys
4 Make
5 Git

LINKÖPING UNIVERSITY

Projektplanering

Innan ni börjar koda

Planera och designa!

1 Projektintroduktion
2 Projektplanering
3 Objektorienterad analys
4 Make
5 Git

LINKÖPING UNIVERSITY

Objektorienterad analys (OOA)

1. Finn objekten
   (leta substantiv)
2. Klassificera objekten
   CRC (namn, ansvar, samarbeten)
3. Beskriv relationer
   (arv eller association)
4. Identifiera aktörer och användningsfall

Objektorienterad analys (OOA)

Steg 1: Finn objekten

Ett förslag till hur man ska finna objekt är att läsa kraspecifikationen och notera förekommande substantiv.

Objektorienterad analys (OOA)

Steg 2: Klassificera objekten

Varje objekt från steg 1 ska representeras som en klass. Skriv ned klassens:

- namn - Engelska namn, substantiv, singular.
- ansvar - operationer som kan utföras på eller av objeketet.
- samarbetspartners - Vilka andra klasser som klassen behöver samarbeta med för att utföra sina åtaganden.

CRC-kort

Class-**R**esponsibility-**C**ollaborators

CRC-kort - Exempel

Bil	
Svänga Accelerera	Ratt, Hjul Pedal, Motor Person

Objektorienterad analys (OOA)

Steg 3: Beskriver relationer

Bestäm de relationer som finns mellan klasserna:

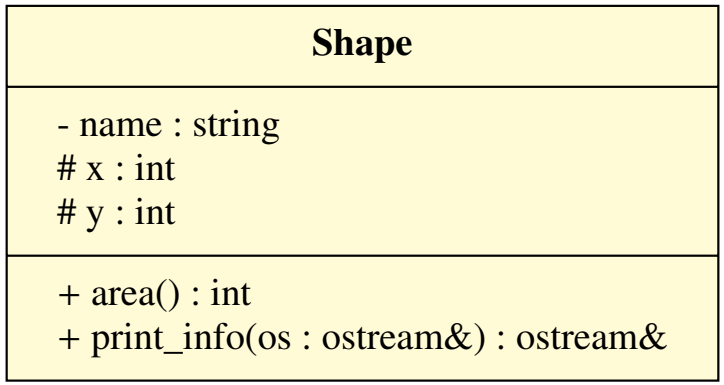
- arv - x är en specialisering av y
- komposition - x består av y (x finns inte utan y)
- aggregation - x består av y
- association - x känner till y

UML-diagram

- Visuell representation av projektets design
- Oberoende av programmeringsspråk
- Standardiserad representation
- Innehåller:
  - Klasser med alla medlemmar
  - Relationer mellan klasser

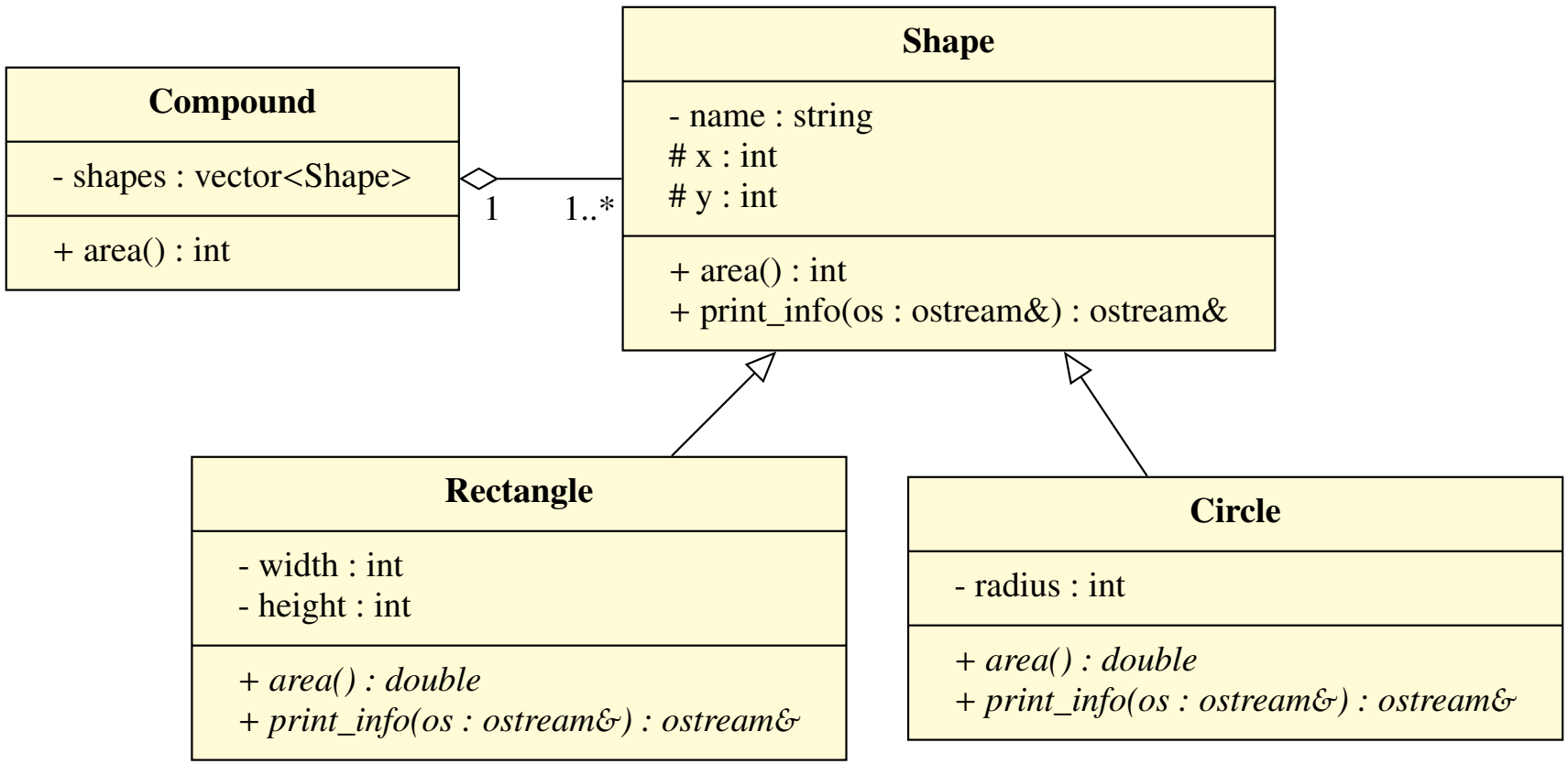
UML-diagram

Hur representeras en klass?

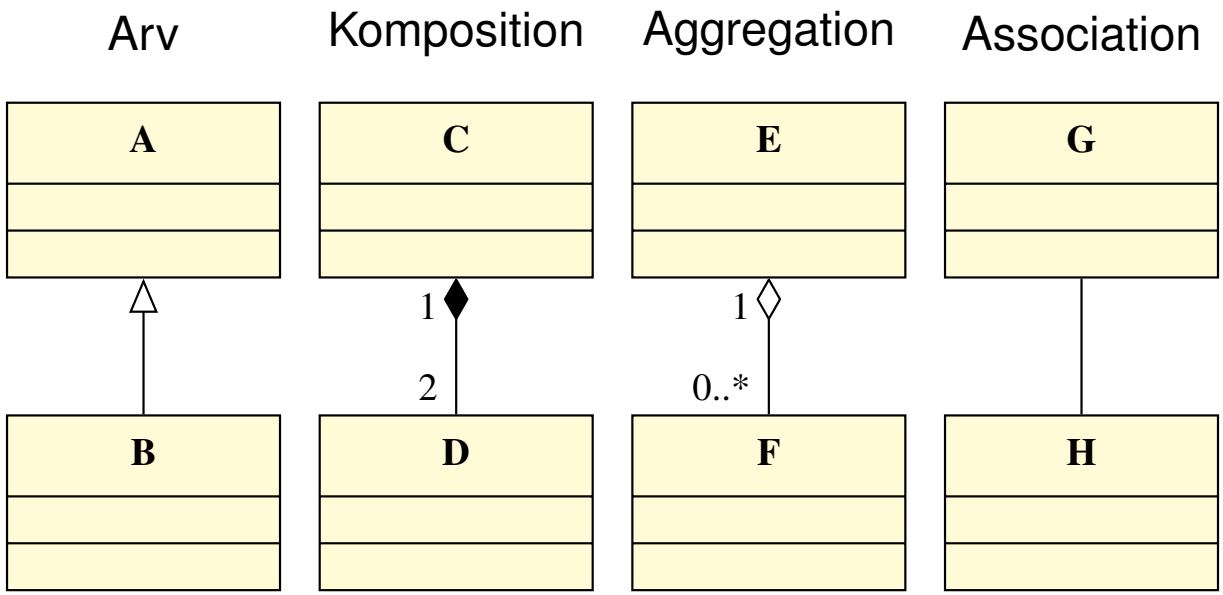


- + public
- - private
- # protected

UML-diagram


Shape
- name : string
x : int
y : int
+ area() : int
+ print_info(os : ostream&) : ostream&
Compound
- shapes : vector<Shape>
+ area() : int
1
1..*
Rectangle
- width : int
- height : int
+ area() : double
+ print_info(os : ostream&) : ostream&
Circle
- radius : int
+ area() : double
+ print_info(os : ostream&) : ostream&

UML-diagram



UML-diagram i kod

```
class Compound
{
public:
  Compound() = default;

private:
  vector<Shape*> shapes; // Aggregation eller komposition
  std::string name;      // Komposition
};

class Rectangle : public Shape // Arv
{
  //...
};
```

Objektorienterad analys (OOA)

Steg 4: Aktörer och användningsfall

Ett användningsfall är en interaktion som kan inträffa under exekvering.

- Identifiera en interaktion och beskriv vilka komponenter som är inblandade i att hantera den.
- Syftet är att få en djupare insikt i samarbetet mellan objekt.
- Kan resultera i nya ansvar, klasser eller samarbeten, eller att klasser som inte används tas bort.

Projektplanering

Mer om detta på lektionen!

1 Projektintroduktion
2 Projektplanering
3 Objektorienterad analys
4 Make
5 Git

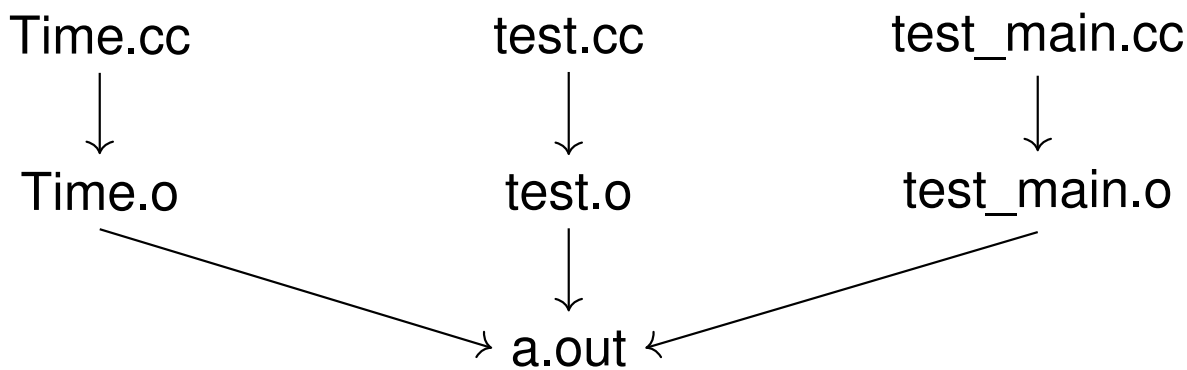
LINKÖPING UNIVERSITY

Kompilera stora projekt

- Mycket att hålla koll på
  - Vilka flaggor ska användas?
  - Vilka filer finns?
  - Vilka berodende har varje fil?
  - Ska flera exekverbara filer skapas?
- Lång kompileringstid om all filer ska kompileras om

Kompilera stora projekt

```
$ g++ -c Time.cc
$ g++ -c test.cc
$ g++ -c test_main.cc
$ g++ Time.o test.o test_main.o
```



Make

- Ett verktyg där vi specifierar hur kompilering ska ske
- Sparas i en fil som heter `Makefile`
  - Innehåller regler för hur varje fil ska kompileras
  - Körs genom kommandot `make` i terminalen
- Make håller koll på vilka filer som har uppdaterats och behöver kompileras om

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Mål: Den filen som ska skapas `a.out`

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Beroenden: “Om dessa filer uppdaterats behöver vi kompilera om.”
  `Time.cc Time.h`

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Kommando: Hur skapas målet?
  `g++ -c test.cc`

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Regel: Alla bitar tillsammans är en regel.

```
test_main.o: test_main.cc
    g++ -c test_main.cc
```

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Mål: Den filen som ska skapas
  `a.out`
- Beroenden: “Om dessa filer uppdaterats behöver vi kompilera om.”
  `Time.cc Time.h`
- Kommando: Hur skapas målet?
  `g++ -c test.cc`
- Regel: Alla bitar tillsammans är en regel.

```
test_main.o: test_main.cc
    g++ -c test_main.cc
```

Make - Stort exempel

```
FLAGS = -std=c++17 -Wall -Wextra
LIBS  = -lsfml-window -lsfml-graphics -lsfml-system
OBJS  = player.o enemy.o main.o

# $(var) ger värdet av variabeln
# $@ refererar till målet och $^ till beroenden
game: $(OBJS)
    g++ $(FLAGS) -o $@ $^ $(LIBS)

# % är en wildcard, $< refererar till första beroendet
%.o: %.cc %.h
    g++ $(FLAGS) -c $<

# detta är en regel som inte skapar en fil
.PHONY: clean
clean:
    rm *.o game
```

1 Projektintroduktion
2 Projektplanering
3 Objektorienterad analys
4 Make
5 Git

LINKÖPING UNIVERSITY

Git

- Ett så kallat versionshanteringssystem
  - Sparar vår kod i ögonblicksbilder
  - Vi kan spåra ändringar
  - Vi kan gå tillbaka i tiden
  - Vi kan återskapa arbete om olyckan är framme
- Git != Github/Gitlab
  - Git är i första hand ett sätt att versionshantera
  - Går att koppla till en server så att flera kan samarbeta, men ej nödvändigt
- https://learngitbranching.js.org/

Git

Grundanvändning av git

```
$ git init
$ git status
On branch main

No commits yet

nothing to commit
```

Git

Grundanvändning av git

```
$ echo "En sträng" > fil.txt
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include what will be committed)
        fil.txt
```

Git

Grundanvändning av git

```
$ git add fil.txt
$ git status
On branch main

No commits yet

Changes to be committed:
        new file: fil.txt
```

Git

Grundanvändning av git

```
$ git add fil.txt
$ git status
On branch main

No commits yet

Changes to be committed:
        new file: fil.txt

$ git commit -m "Min första commit!"
$ git status
On branch main
nothing to commit, working tree clean
```

Git

Grundanvändning av git

```
$ echo "en ny sträng" > fil.txt
$ echo "till en annan fil" > annan.cc
$ git diff
fil.txt
@@ -1 +1 @@
-en sträng
+en ny sträng
```

Git

Grundanvändning av git

```
$ echo "en ny sträng" > fil.txt
$ echo "till en annan fil" > annan.cc
$ git diff
fil.txt
@@ -1 +1 @@
-en sträng
+en ny sträng

$ git add .
$ git status
On branch main

No commits yet

Changes to be committed:
        new file: annan.cc
        modified: fil.txt
$ git commit -m "Min andra commit"
```

Git

Grundanvändning av git

```
$ git log
commit 50e6a8 (HEAD -> main)
Author: Eric Ekström
    Min andra commit

commit bd07e4
Author: Eric Ekström
    Min första commit!
```

Git

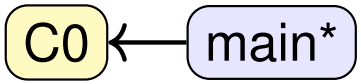
Remote repos

- Vi kan koppla vårt lokala repo till en remote, och på så sätt dela kod med varandra.
  `git remote add origin <address>`
- Git sköter (oftast) att slå ihop kod från flera användare så att vi kan jobba parallellt.
- `git push` för att ladda upp commits till en remote.
- `git pull` för att ladda ner commits från en remote.

Git

Branches

```
git commit
```



Git

Branches

```
git commit
git branch bar
```

Git

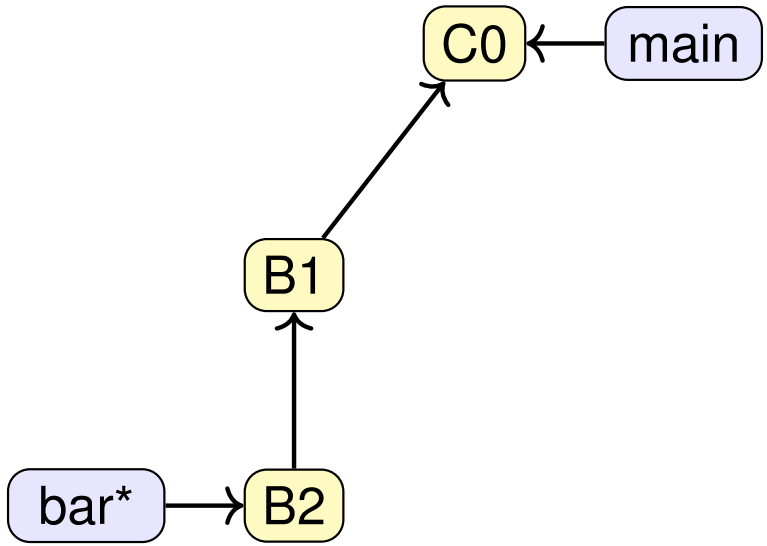
Branches

```
git commit
git branch bar
git checkout bar
```

Git

Branches

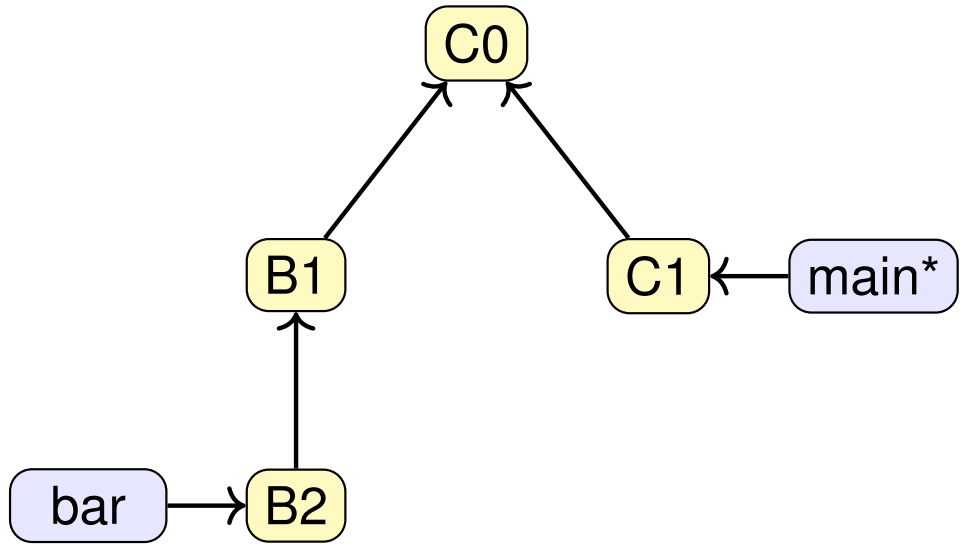
```
git commit
git branch bar
git checkout bar
git commit; git commit
```



Git

Branches

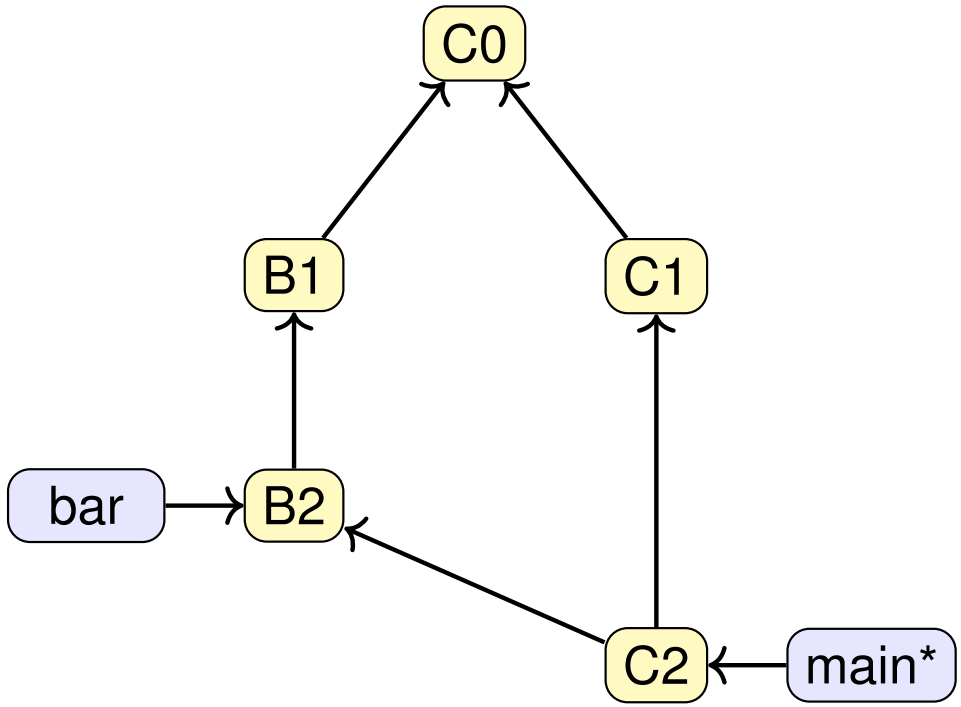
```
git commit
git branch bar
git checkout bar
git commit; git commit
git checkout main; git commit
```



Git

Branches

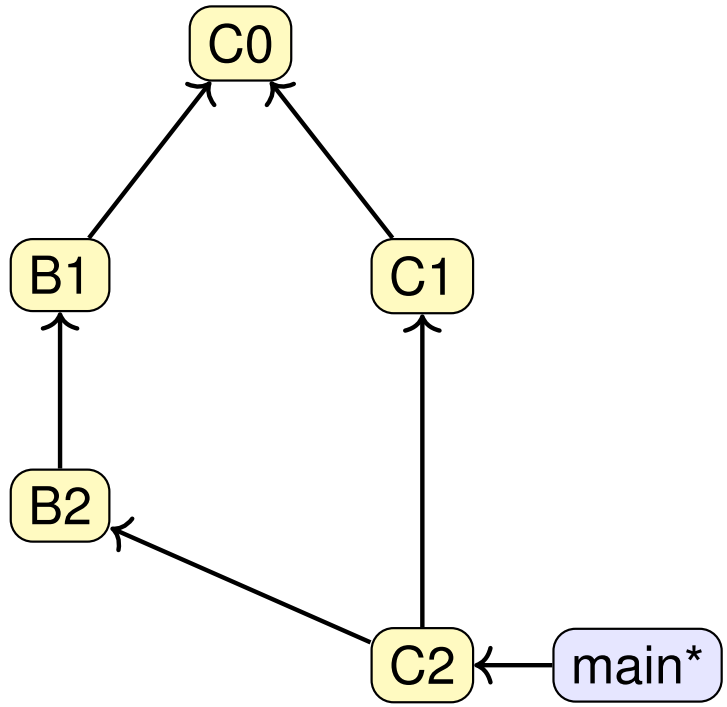
```
git commit
git branch bar
git checkout bar
git commit; git commit
git checkout main; git commit
git merge bar
```



Git

Branches

```
git commit
git branch bar
git checkout bar
git commit; git commit
git checkout main; git commit
git merge bar
git branch -d bar
```



Git

Viktiga commandon

Skapa ett git-repo:

- `init` - skapa ett nytt tomt repo i en befintlig katalog
- `clone <address>` - kopiera ett repo från en remote

Hålla koll på sitt git-repo:

- `status` - visar relevant information
  (Tips: Kör alltid status mellan varje operation!)
- `diff` - visar nuvarande ändringar jämfört med senast commit
- `log` - visar listan med tidigare commits

Git

Viktiga commandon

Göra nya commits:

- `add <file>` - registrerar ändringar till nästa commit.
- `commit -m "message"` - skapa en ny commit

Prata med en remote:

- `push` - synkronisera server med de commit du har lokalt
- `pull` - syknronisera ditt lokala repo med det som finns på servern

Git

Viktiga commandon

Hantera branches:

- `branch <namn>` - skapa en branch
- `branch -d <namn>` - ta bort en branch
- `checkout <namn>` - byt branch
- `merge <namn>` - tillämpa en branch till nuvarande

Git

Tips: Lyssna på vad git säger!

```
$ git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a
remote repository using

    git remote add <name> <url>

and then push using the remote name

    git push <name>
```

Git

Tips: Git config

Vi kan anpassa det mesta i git.

```
# Sätt användarnamn och mailadress
git config --global user.name eriek23
git config --global user.email eric.ekstrom@liu.se

# Välj vilken editor som ska användas
git config --global core.editor emacs

# Skapa ett alias 'git ci' för 'git commit'
git config --global alias.ci commit
```

Se till att sätta upp namn och mailadress!
Annars kan assistenten inte se vem det är som skrivit kod.

Git

Merge Conflict

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

```
int main()
{
<<<<<<<<< HEAD
  cout << "Bar" << endl;
=========
  cout << "Foo" << endl;
>>>>>>>>> bar
}
```

Git

Merge Conflict

Vi fixar konflikten

```
int main()
{
  cout << "Bar" << endl;
}
```

och kör sedan

```
$ git add Time.cc
$ git commit -m "Fixed merge conflict"
```

Git

Övrigt om git

- `.gitignore` - för filer som inte ska versionshanteras
- `git stash` - Vad gör man om man har arbete som inte är redo att bli en commit?
- ssh-nycklar - för att slippa skriva in lösenord vid varje `push` och `pull`.

www.ida.liu.se/~TDDI82/