Make, CMake och Git

Eric Ekström

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1 Make
2 CMake
3 Git
4 Git i större projekt

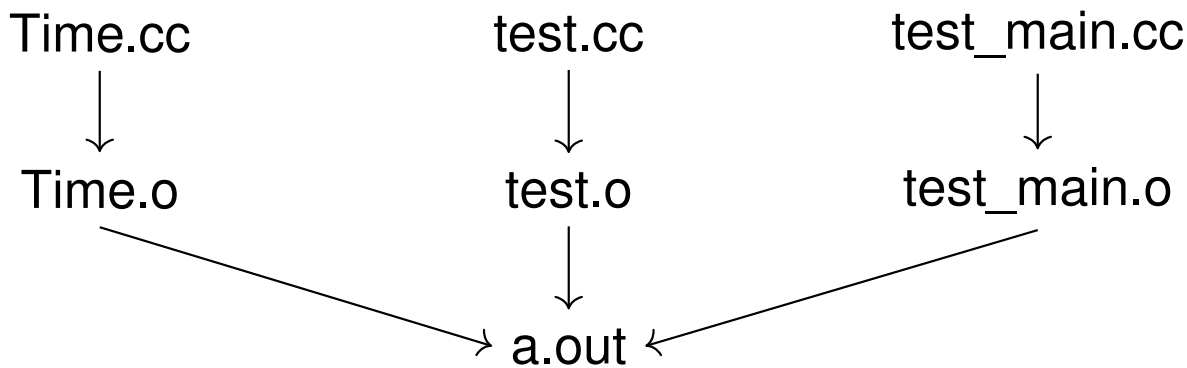
LINKÖPING UNIVERSITY

Kompilera stora projekt

- Mycket att hålla koll på
  - Vilka flaggor ska användas?
  - Vilka filer finns?
  - Hur ska varje fil kompileras?
  - Finns det olika exekverbara filer?
- 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 specifiera 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 Make
2 CMake
3 Git
4 Git i större projekt

LINKÖPING UNIVERSITY

CMake

Ett verktyg för att skapa Makefiler. Skivs i en fil som heter `CMakeLists.txt`.

```
cmake_minimum_required(VERSION 3.15)

project(MyProject)

add_executable(myexample simple.cpp)
```

Cmake används sen för att bygga en Makefile som sedan kompilerar programmet.

```
$ cd build
$ cmake ..
$ make
$ ./a.out
```

CMake

```
cmake_minimum_required(VERSION 3.10)

project(game
     VERSION 0.1
     DESCRIPTION "A game. Its fun."
     LANGUAGES CXX)

set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR})

add_executable(game
     src/Player.cc
     src/Game.cc
     src/main.cc
     src/Enemy.cc)
target_include_directories(game PRIVATE
     src)
target_link_libraries(game
     sfml-graphics
     sfml-window
     sfml-system
     ${SFML_DEPENDENCIES})
```

1 Make
2 CMake
3 Git
4 Git i större projekt

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

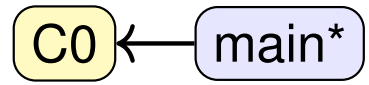
Remote repos

- 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

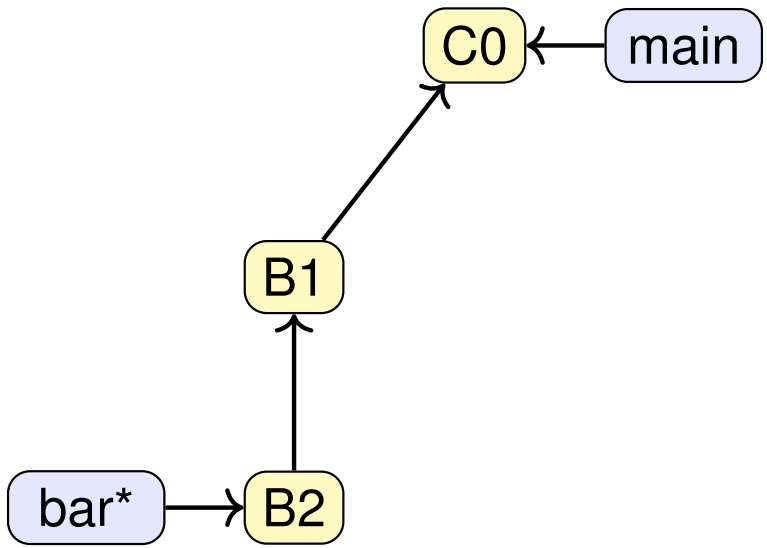
Branches

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

Git

Branches

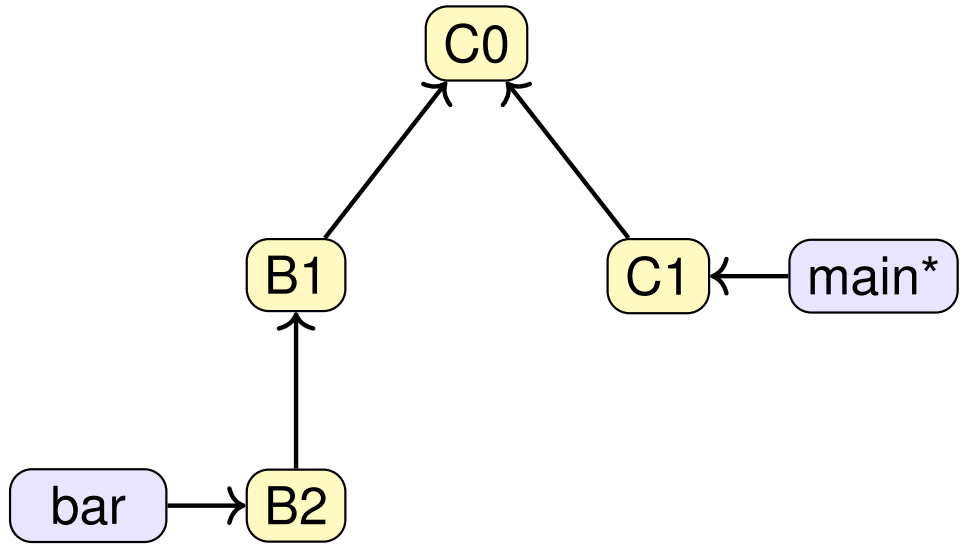
```
git commit
git branch bar
git checkout bar
git commit; git commit
```



Git

Branches

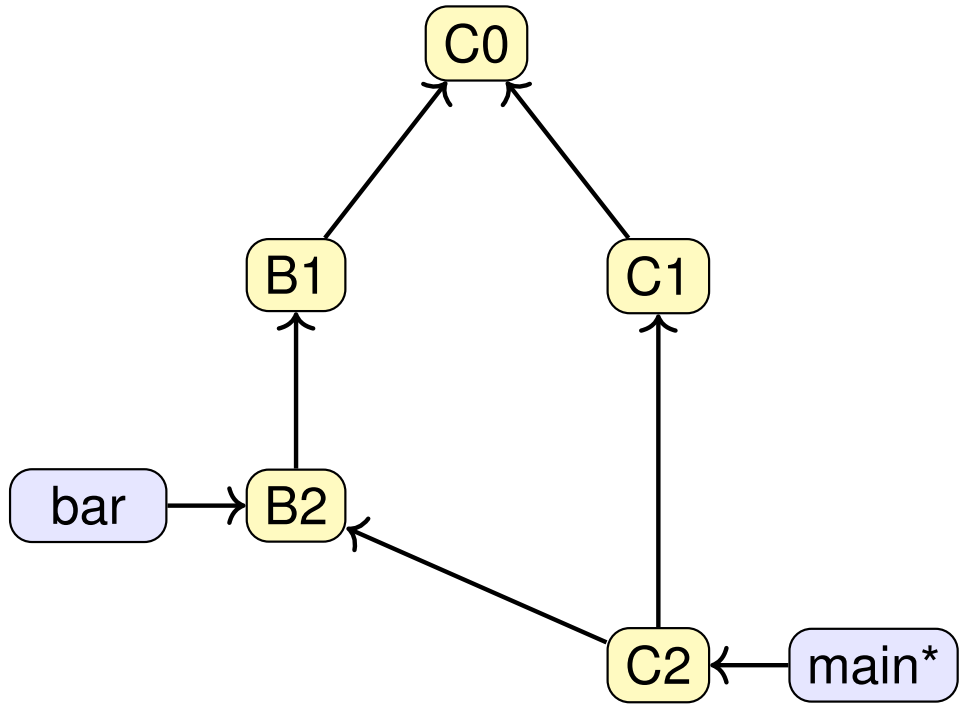
```
git commit
git branch bar
git checkout bar
git commit; git commit
git checkout main; git commit
```



Git

Branches

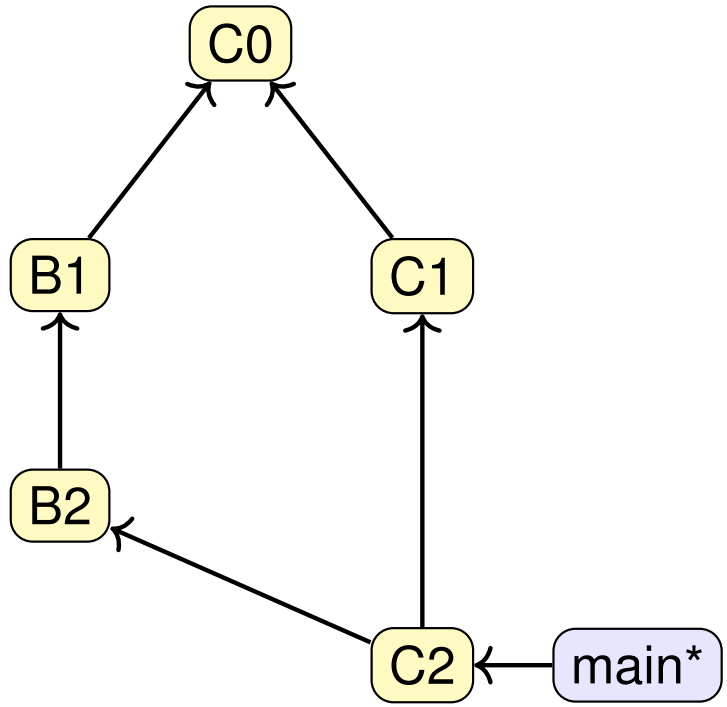
```
git commit
git branch bar
git checkout bar
git commit; git commit
git checkout main; git commit
git merge bar
```



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`.

1 Make
2 CMake
3 Git
4 Git i större projekt

LINKÖPING UNIVERSITY

GitLab – Issues

GitLab har inbyggt stöd för issues. Kan användas för att hålla koll på vad som ska göras härnäst, buggar, etc.

I ett commit-meddelande kan man skriva:

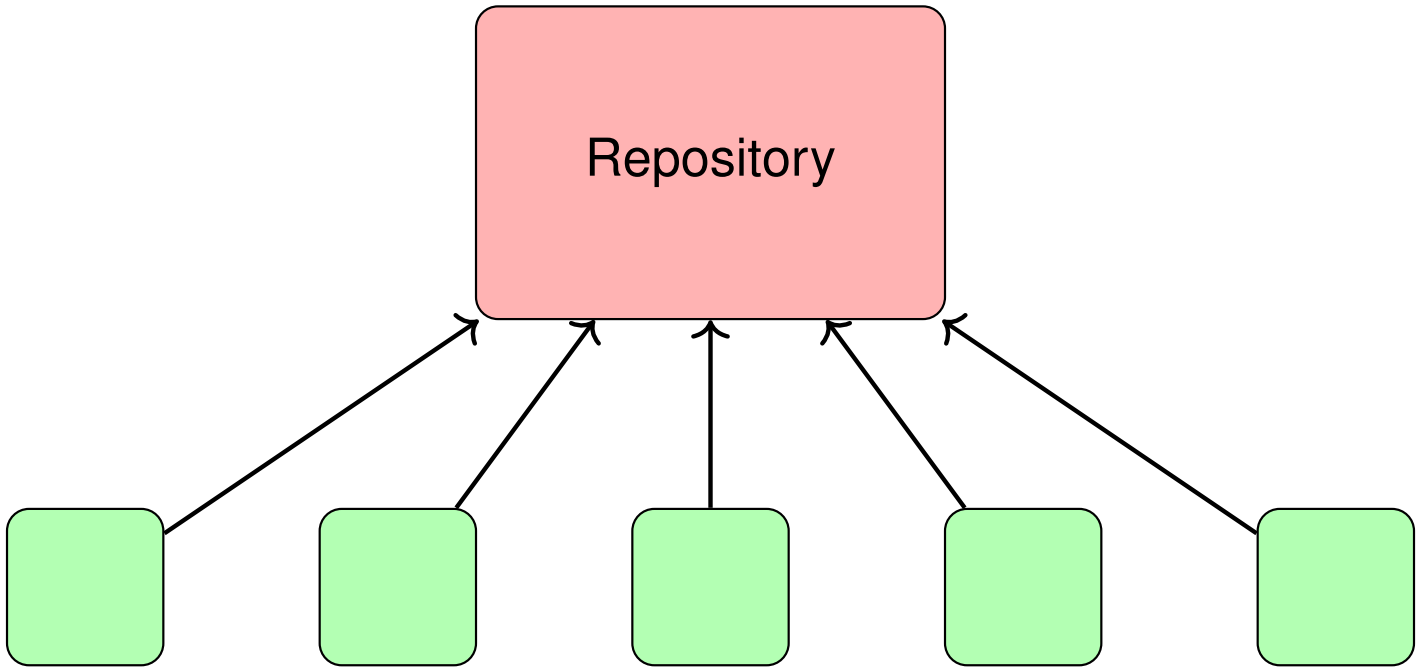
fixes #13

Då stängs automatiskt issue #13, och en referens till den commit som stängde den läggs till.

Git – Hur organiseras koden i större projekt?

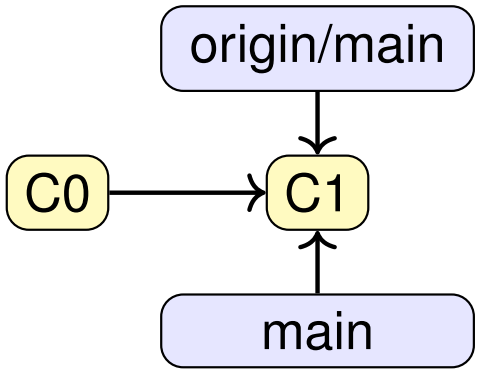
- Merge
- Rebase
- Feature Branch
- Gitflow

Git – Centralt repository



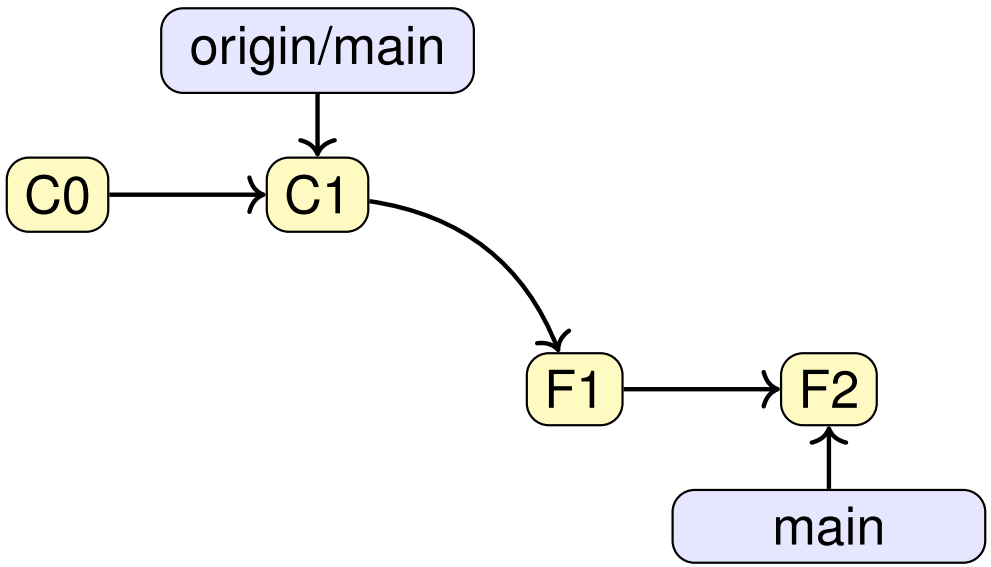
Git – Merge

```
git commit
git push
```



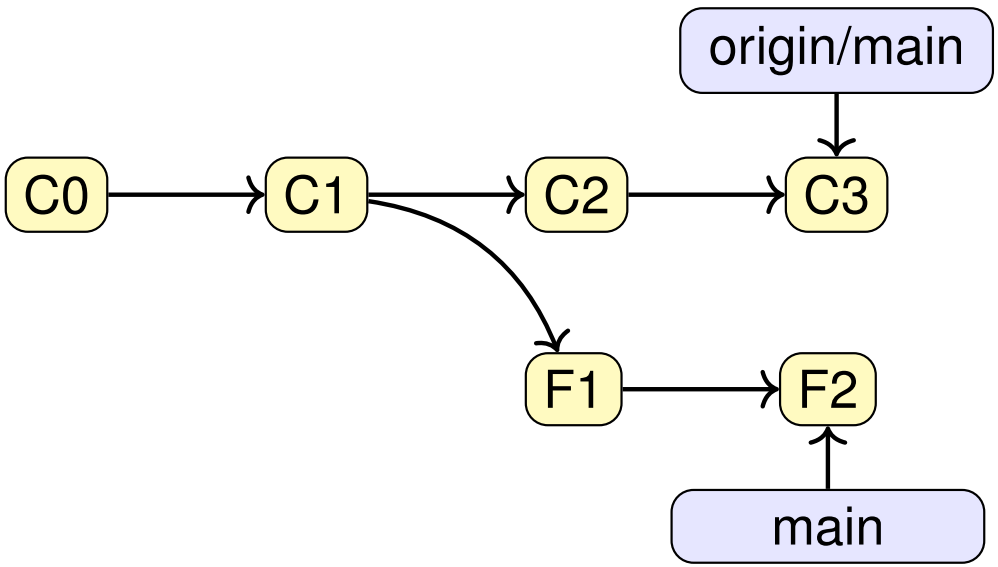
Git – Merge

```
git commit
git commit
```



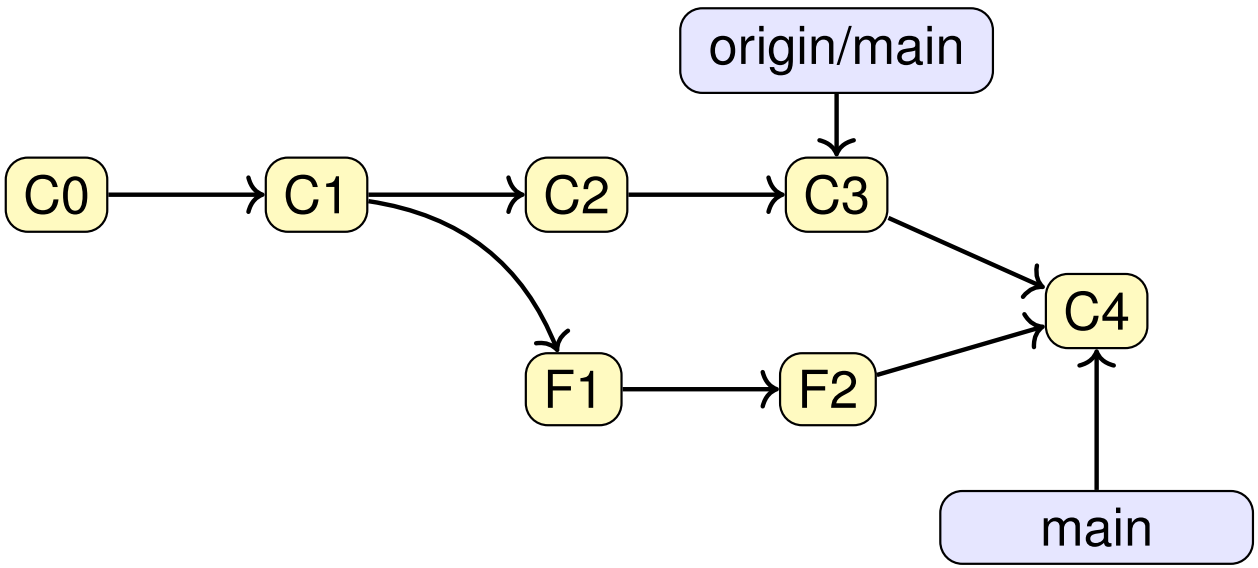
Git – Merge

```
<remote work...>
git fetch
```



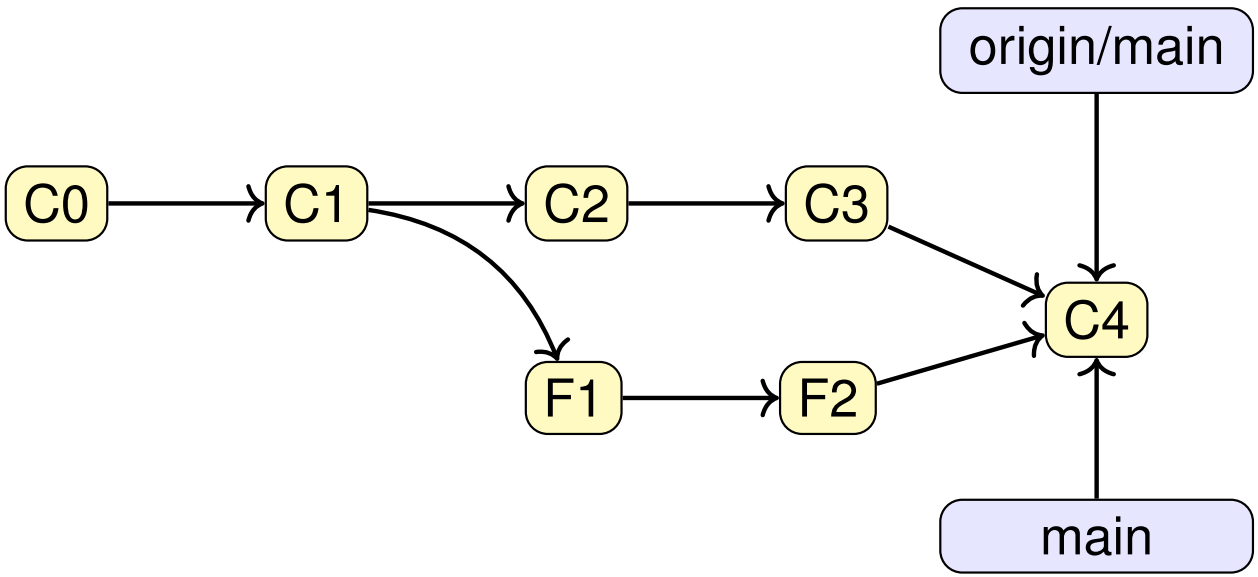
Git – Merge

```
git merge origin/main
```



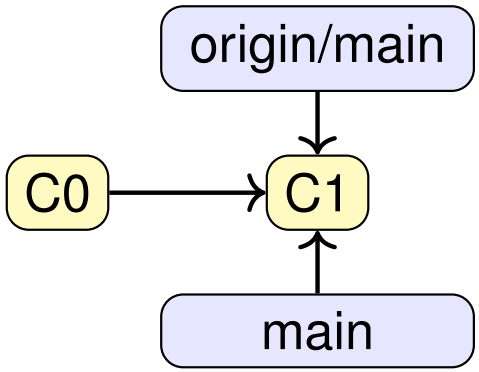
Git – Merge

```
git push
```



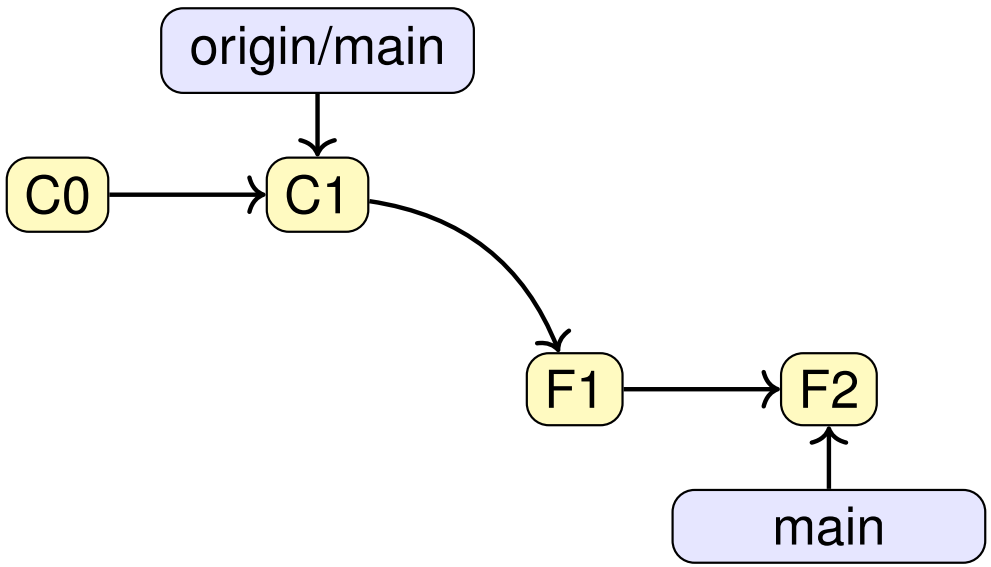
Git – Rebase

```
git commit
git push
```



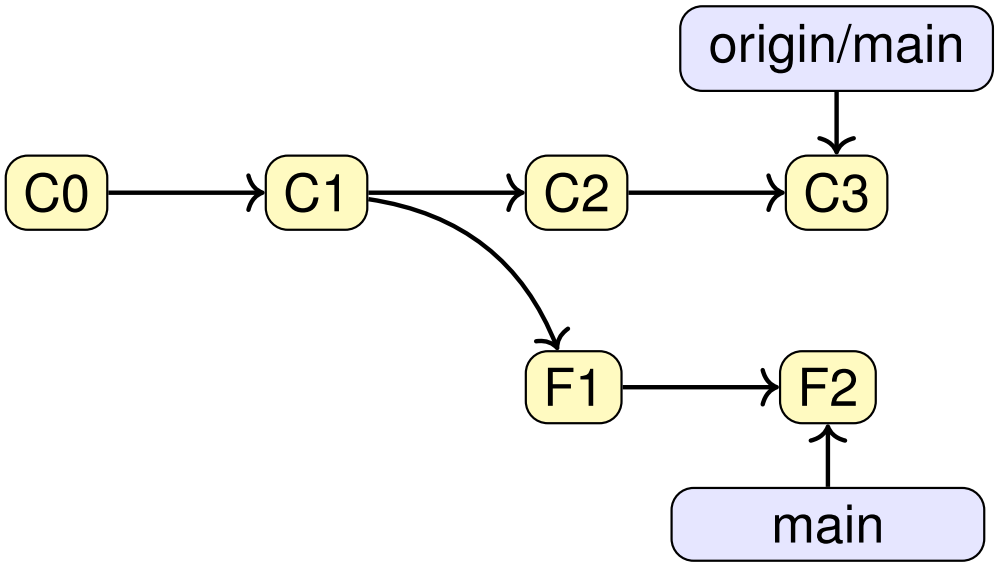
Git – Rebase

```
git commit
git commit
```



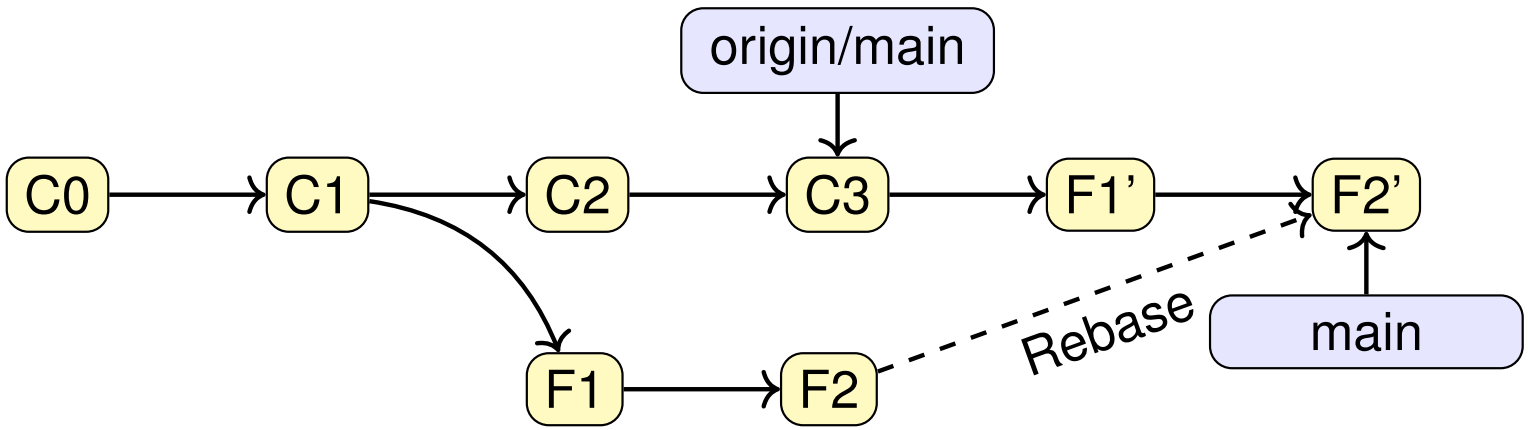
Git – Rebase

```
<remote work...>
git fetch
```



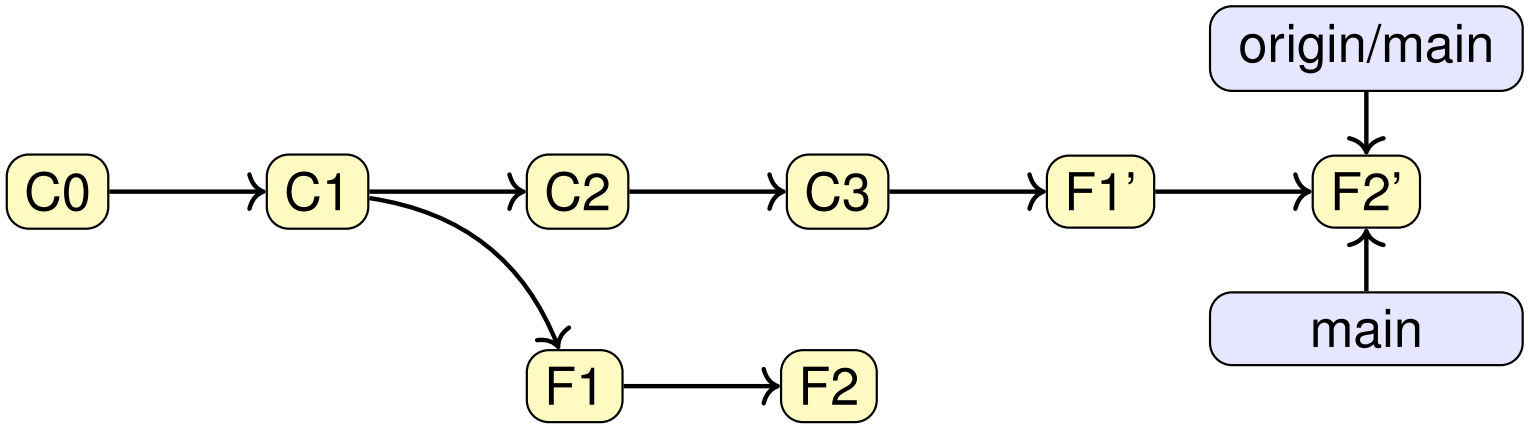
Git – Rebase

```
git rebase origin/main
```

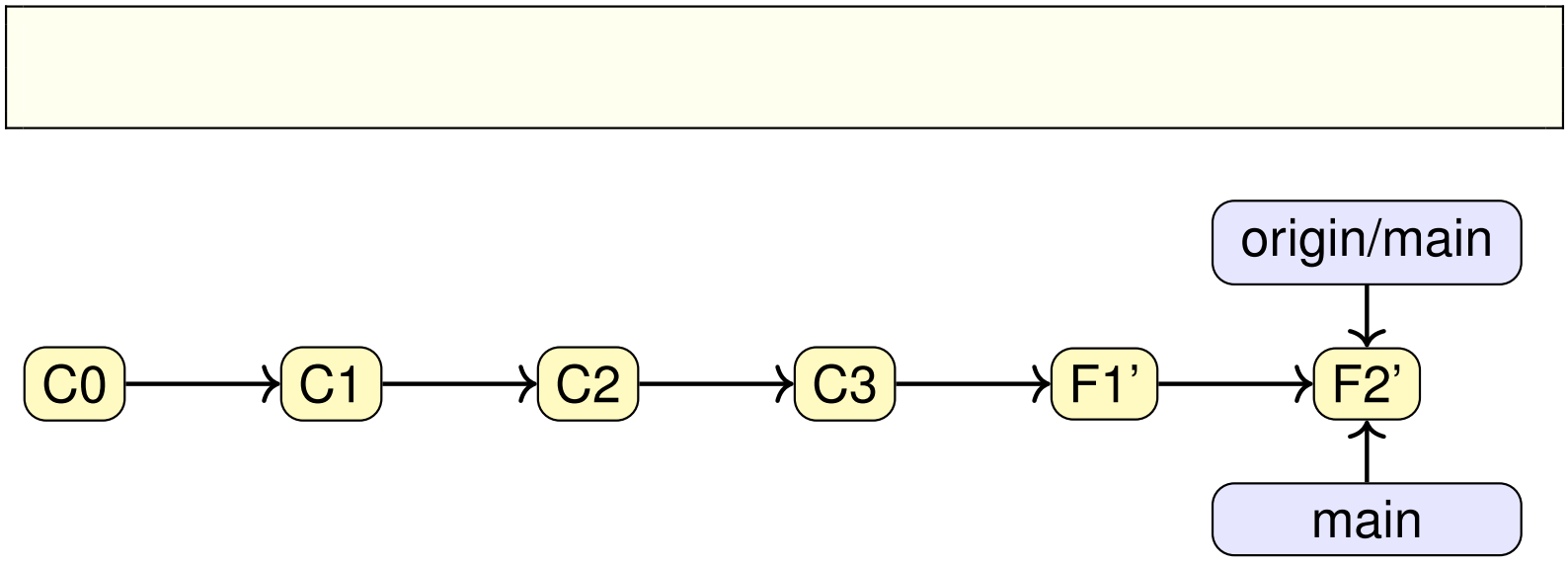


Git – Rebase

```
git push
```



Git – Rebase



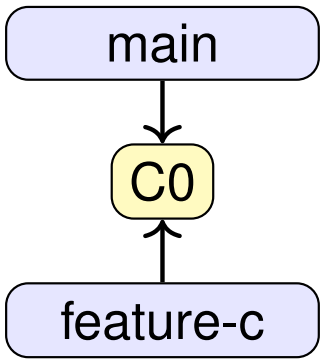
Git – Merge och Rebase

- Enklaste sättet att samarbeta
- Fungerar bra i mindre projekt

- Kan inte samarbeta på funktionalitet som ej är klar
- Risk att `main` inte fungerar, problem för andra
- Stor feature, stora merge-konflikter

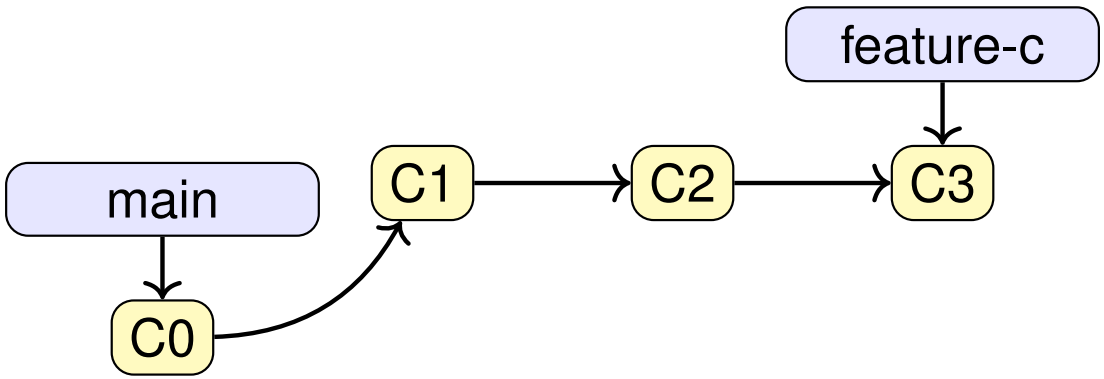
Git – Feature Branch

```
git branch feature-c
git checkout feature-c
```



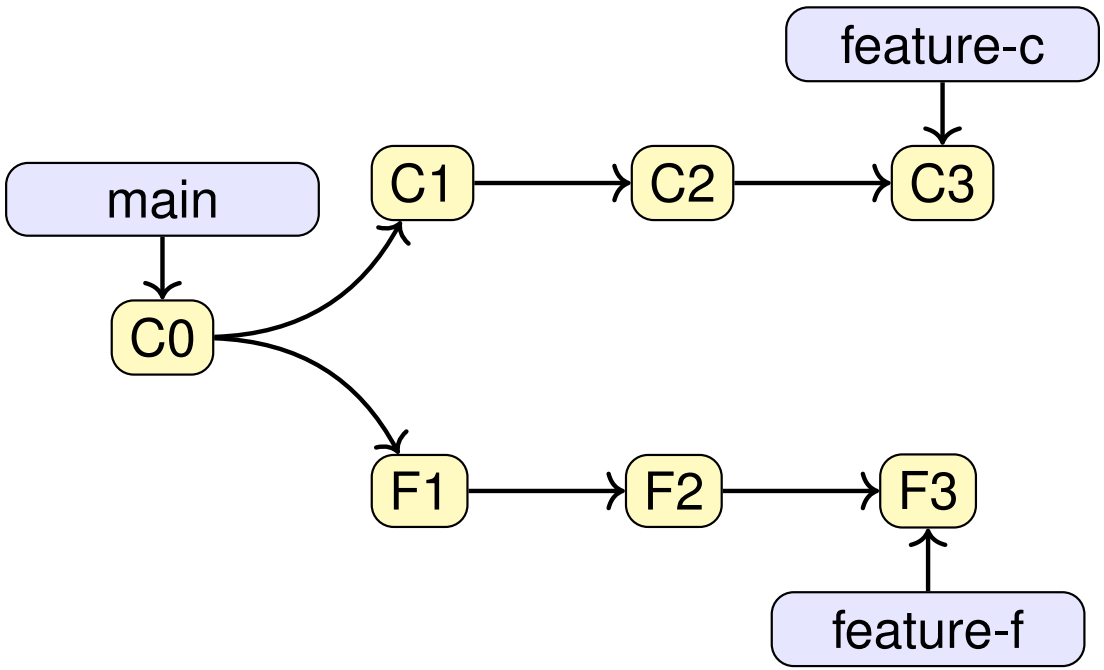
Git – Feature Branch

```
git commit x3
```



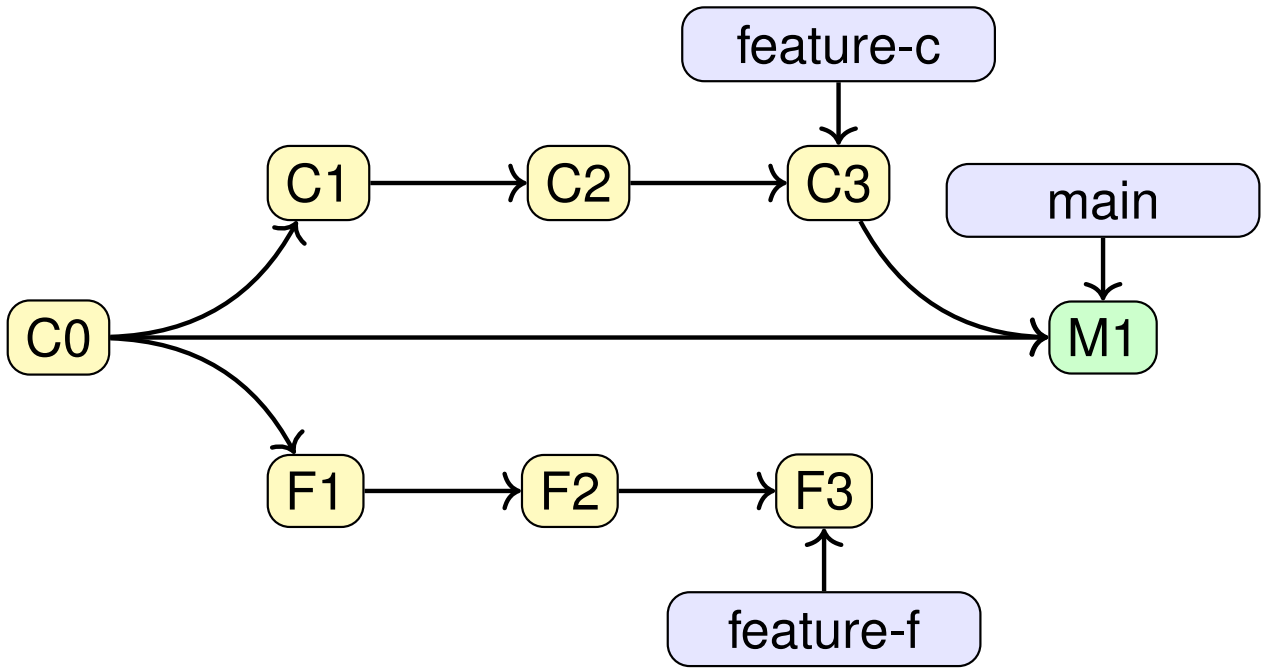
Git – Feature Branch

```
git checkout -b feature-f main
git commit x3
```



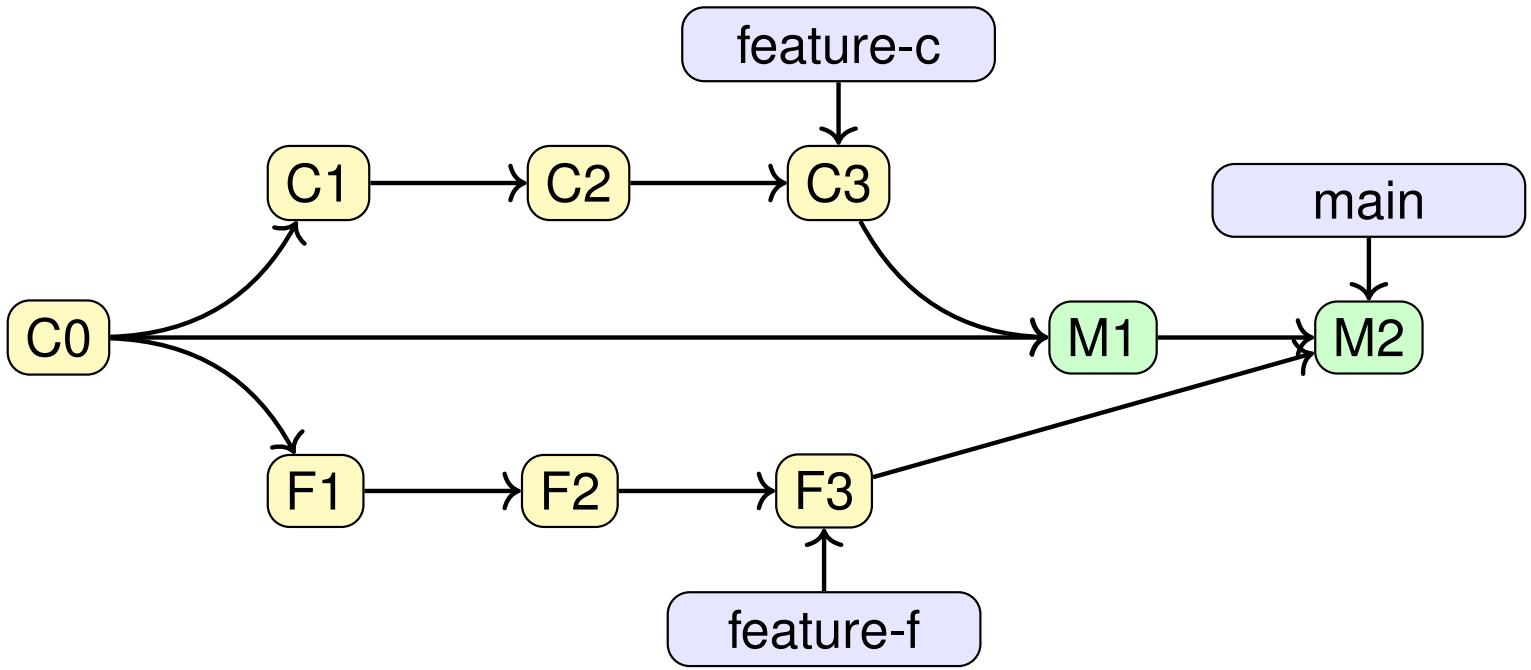
Git – Feature Branch

```
git checkout main
git merge feature-c
```



Git – Feature Branch

```
git merge feature-f
```

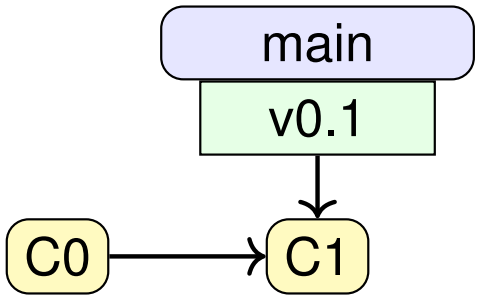


Git – Feature Branch

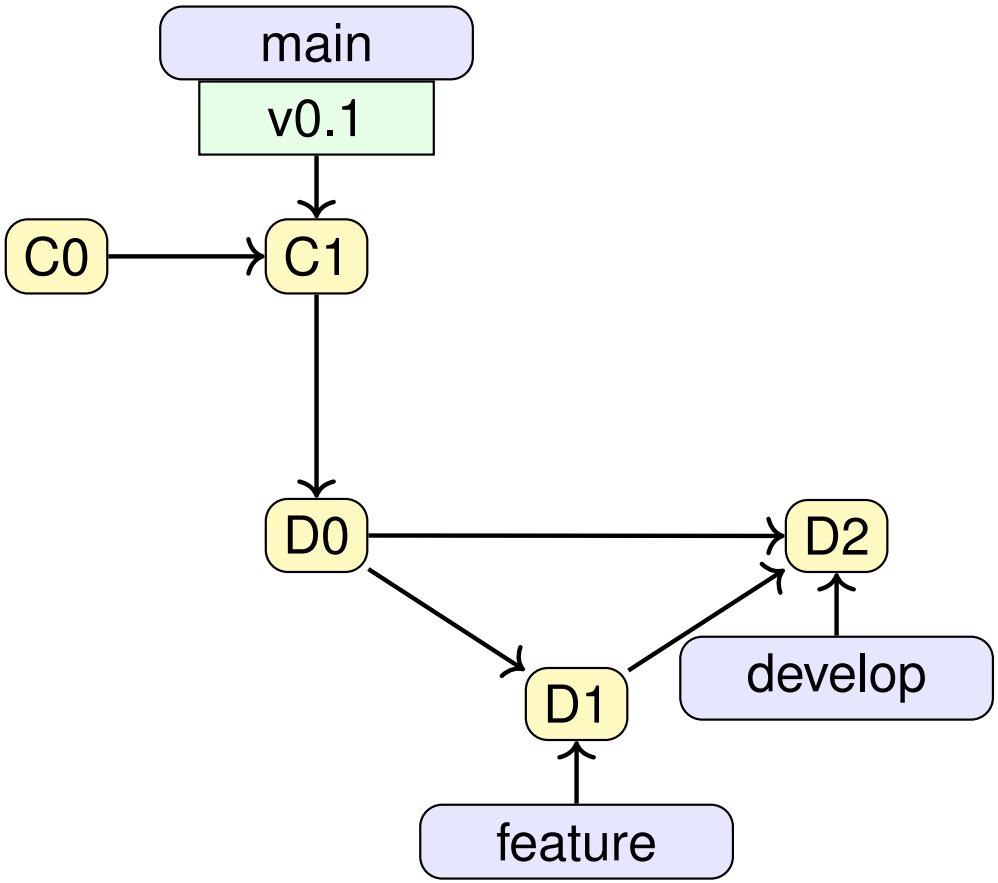
- Samarbete/backup på ej klara funktionalitet
- Tydligt var kodgranskning görs
- `main` fungerar alltid $\Rightarrow$ Continuous Integration

- Lite mer arbete än tidigare
- Risk för att funktionalitet ”tappas bort”
- Risk för stora merges $\Rightarrow$ lite funktionalitet i taget

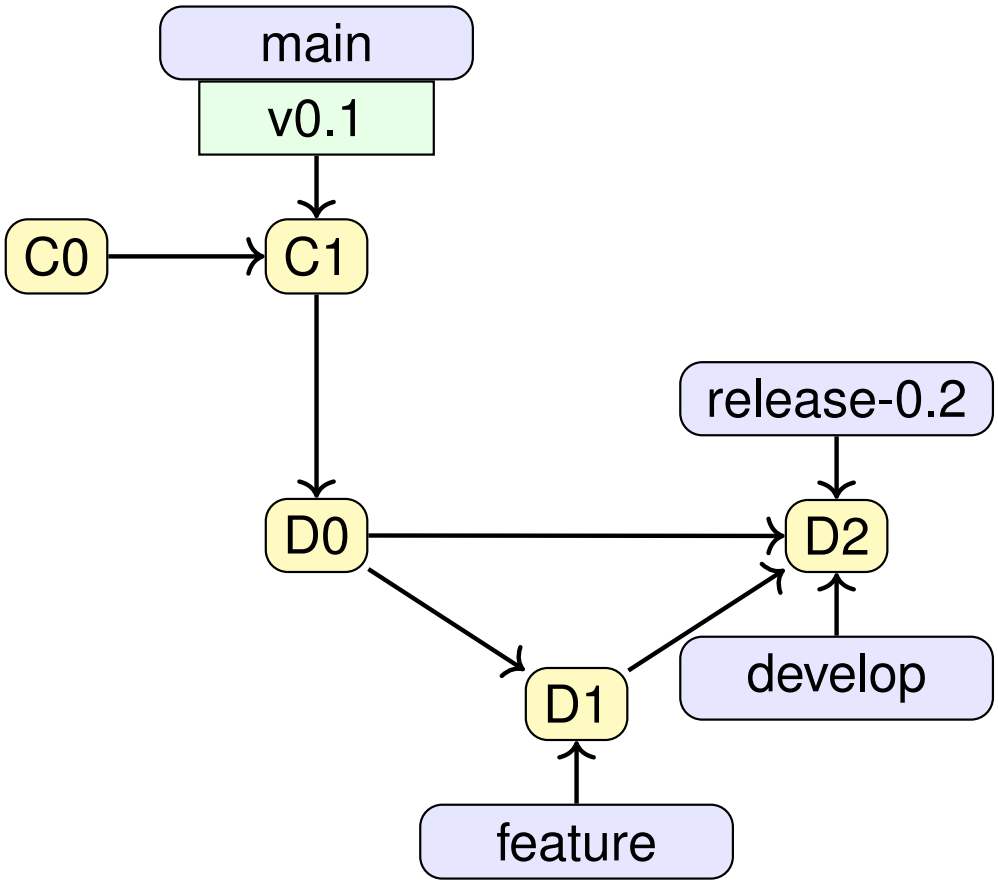
Git – Gitflow



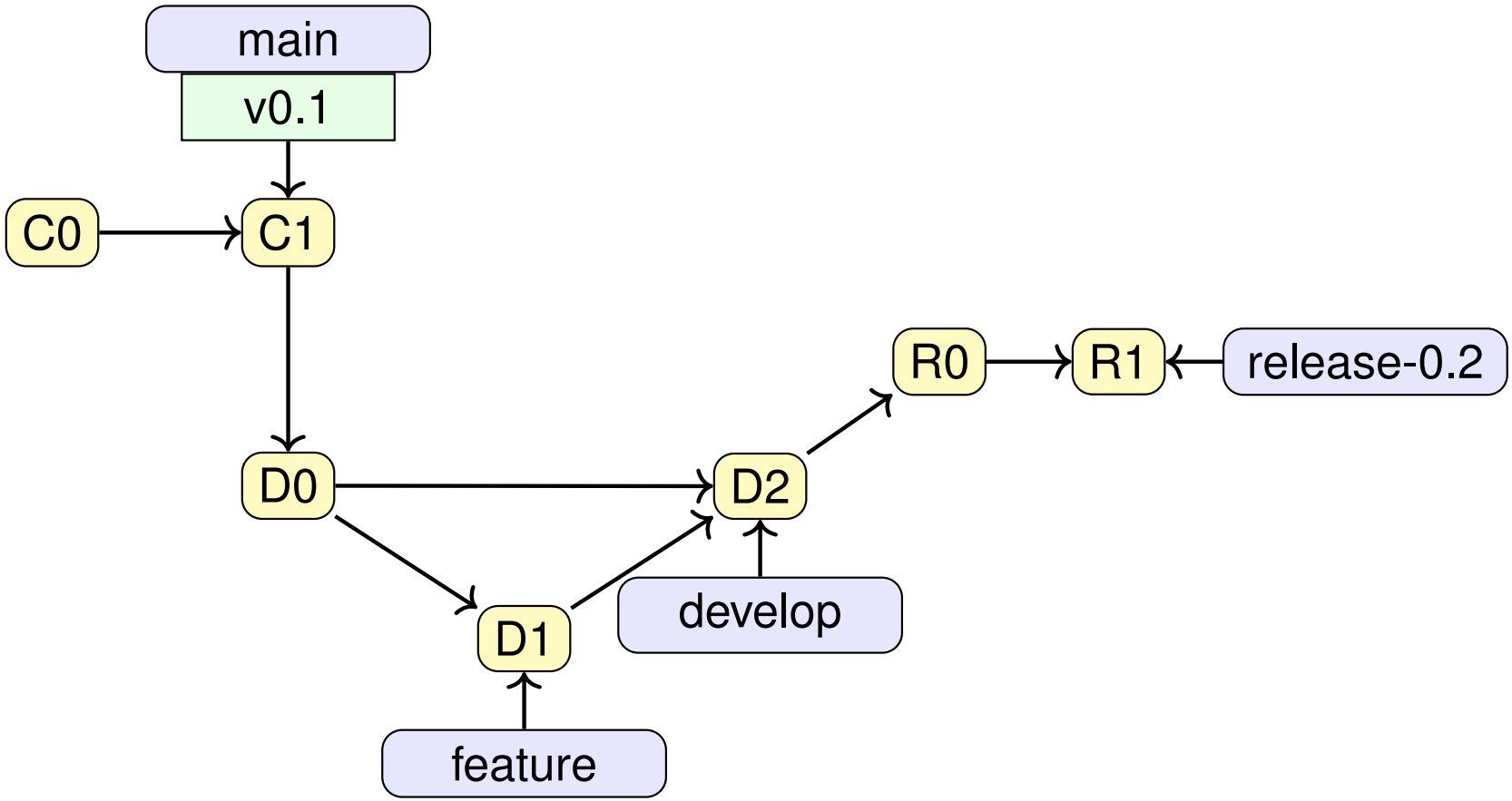
Git – Gitflow



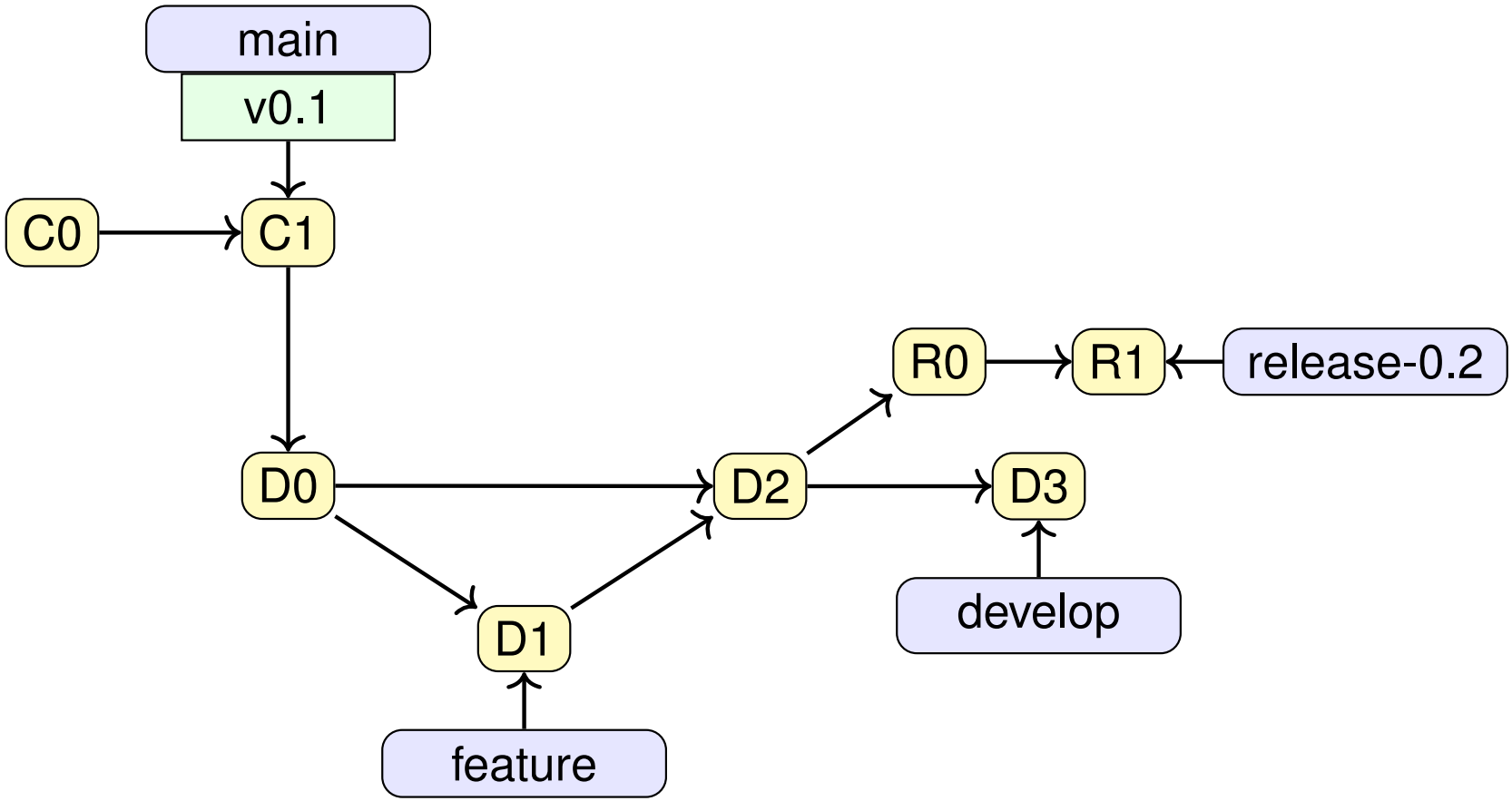
Git – Gitflow



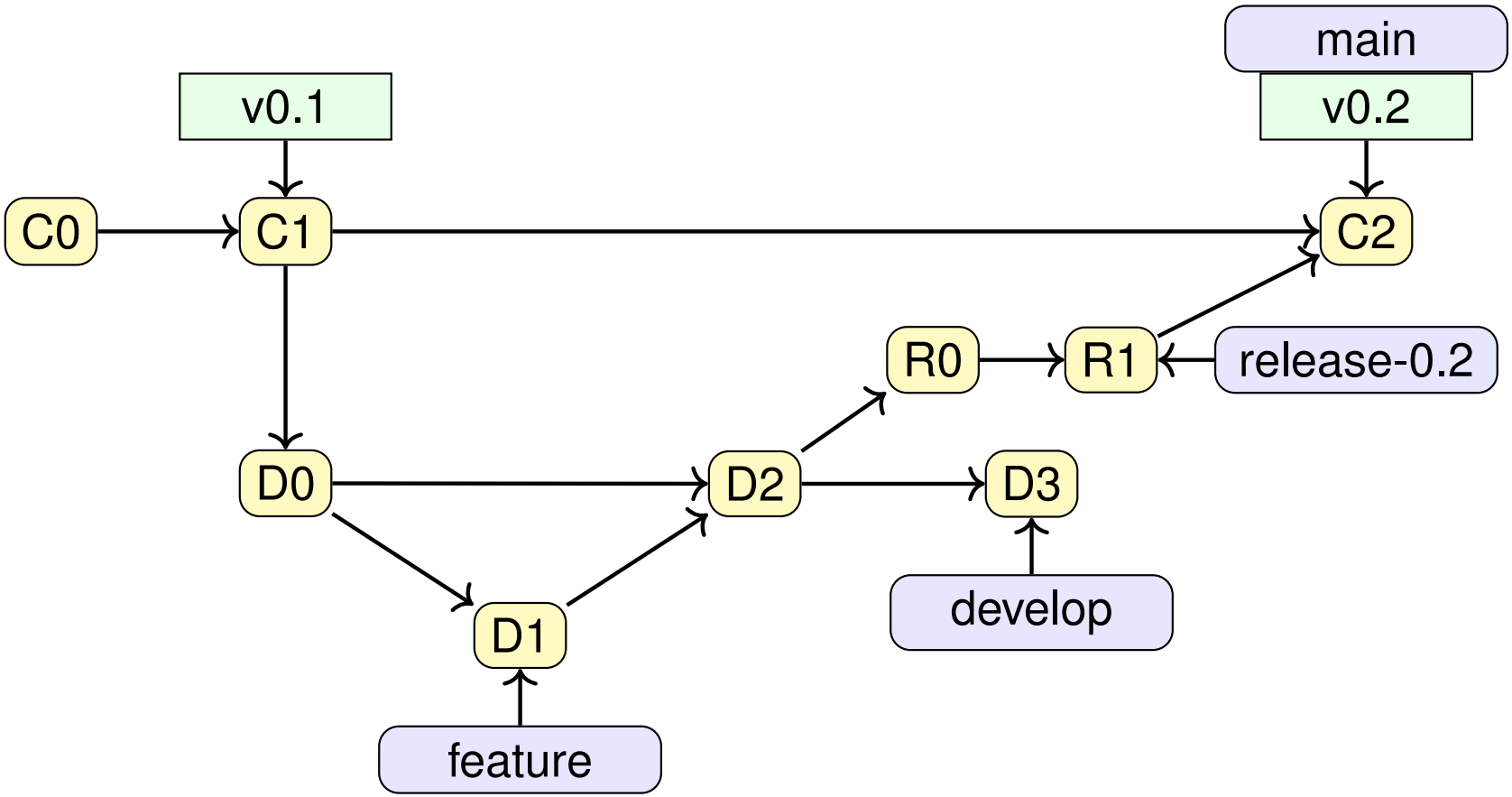
Git – Gitflow



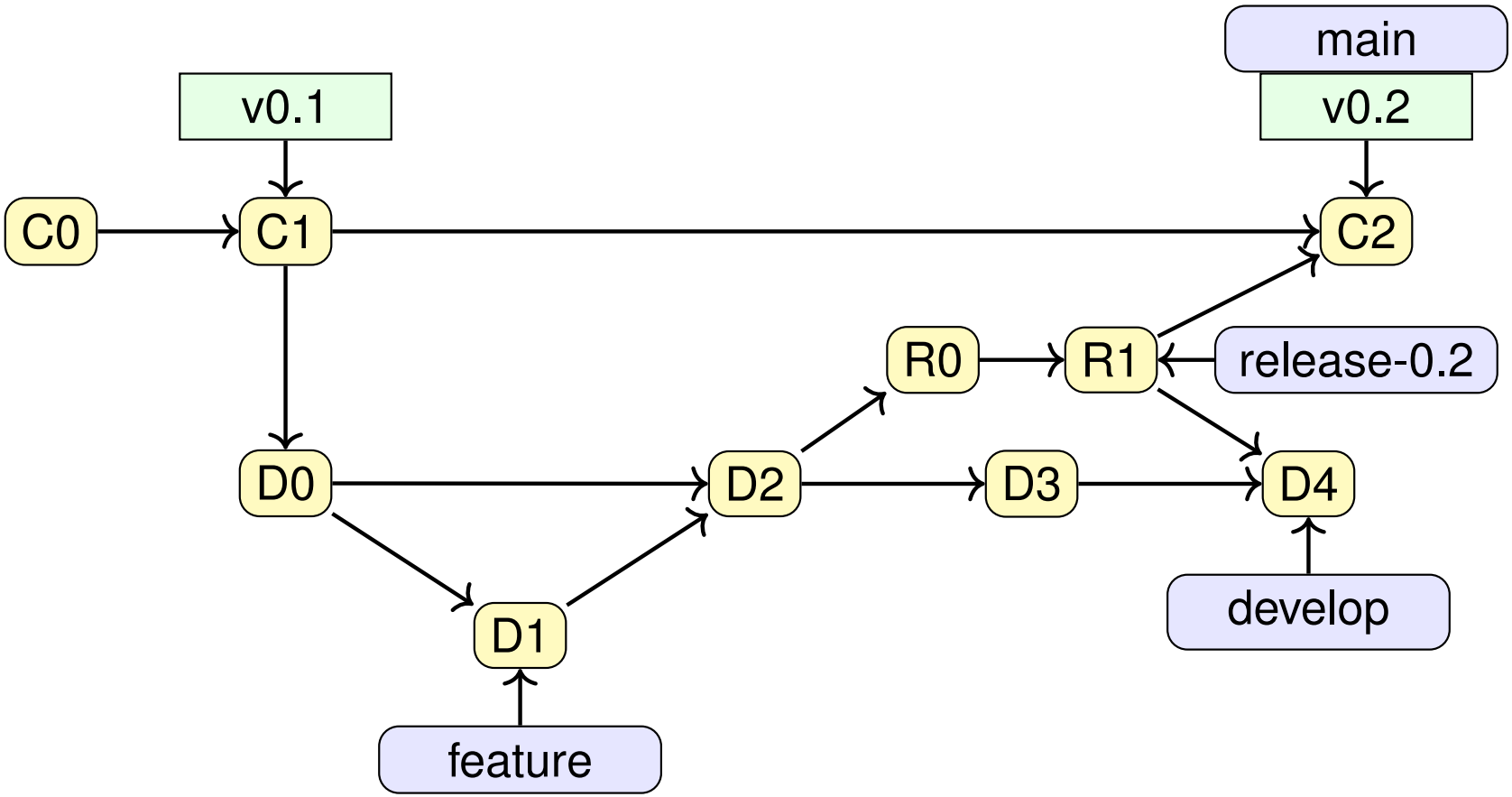
Git – Gitflow



Git – Gitflow


main
v0.1
v0.2
C0
C1
C2
R0
R1
release-0.2
D0
D2
D3
develop
D1
feature

Git – Gitflow


main
v0.1
v0.2
C0
C1
C2
R0
R1
release-0.2
D0
D2
D3
D4
D1
develop
feature

Git – Gitflow

- Release kan testas och färdigställas parallellt med utveckling av ny funktionalitet
- `main` visar tydligt alla versioner
- Enkelt och tydligt att göra hotfix

- Ännu mer att hålla reda på, namngivning är viktig!

www.ida.liu.se/~/

LINKÖPING
UNIVERSITY