Технологии разработки программного обеспечения

2020/ 2021, 1 курс, 2 семестр

Пудов Сергей Григорьевич

Лекция 3

- Git на локальной машине:
  - refresh
- Makefile
- Git branches
  - Create
  - Switch
  - Log and status
  - Merge
  - Конфликты
- Управление ветками

Git на локальной машине: повтор

Книга: https://git-scm.com/book/ru/v2

Помощь:

- *$ git help <verb>*
- *$ git <verb> --help*
- *$ man git-<verb>*

Создание репозитория:

- *$ git init*

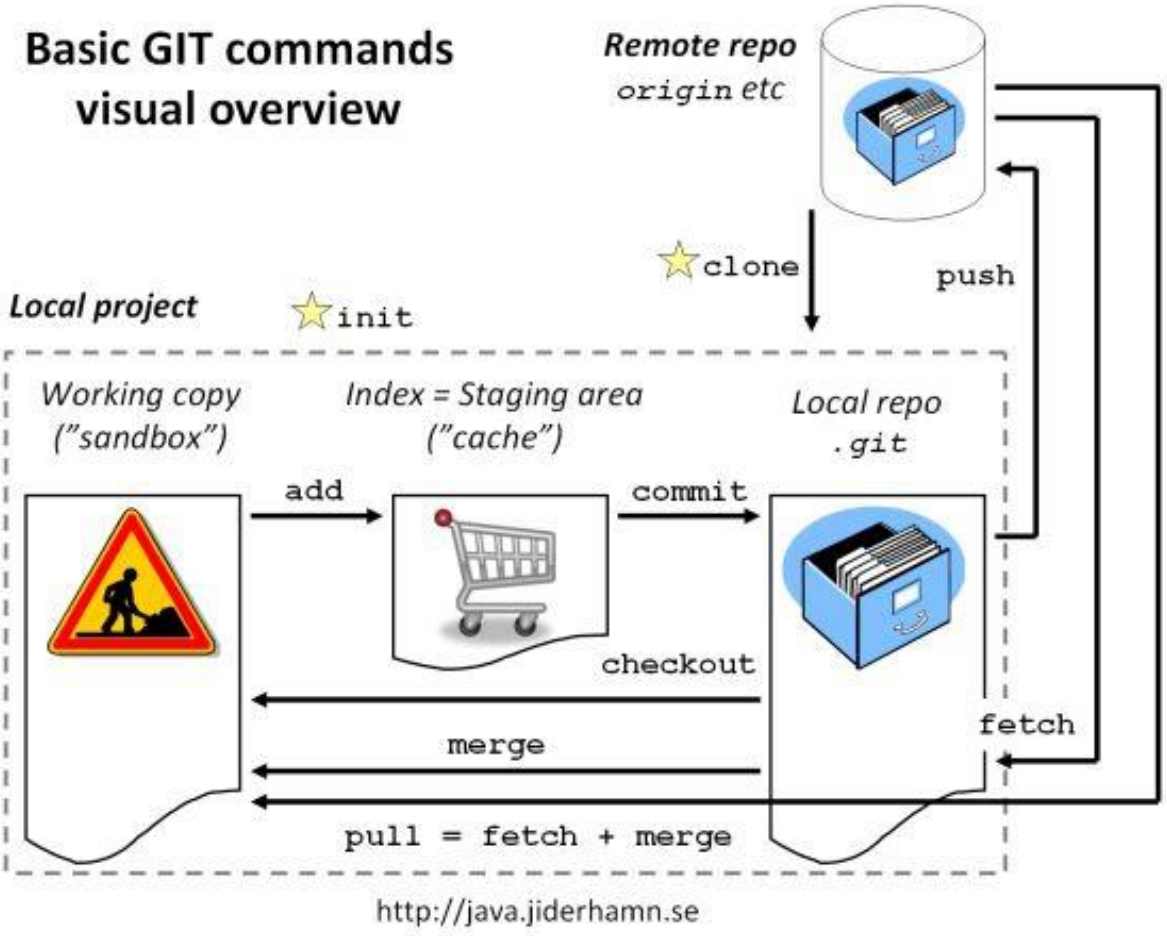
Клонирование удаленного репозитория

- *$ git clone*

Добавление файлов под версионный контроль:

- *$ git add <filename>*
- *$ git commit -m 'initial project version'*
- *$ git push [remote-name] [branch-name]*
- *$ git push origin master*

Git на локальной машине: повтор



http://www.about-dev.com/version-control/44-git-intro

Makefile

Сборка программы вручную:

- ***gcc -g -Wall -o myprog myprog.c***

Makefile:

<цель> …: <зависимость> …

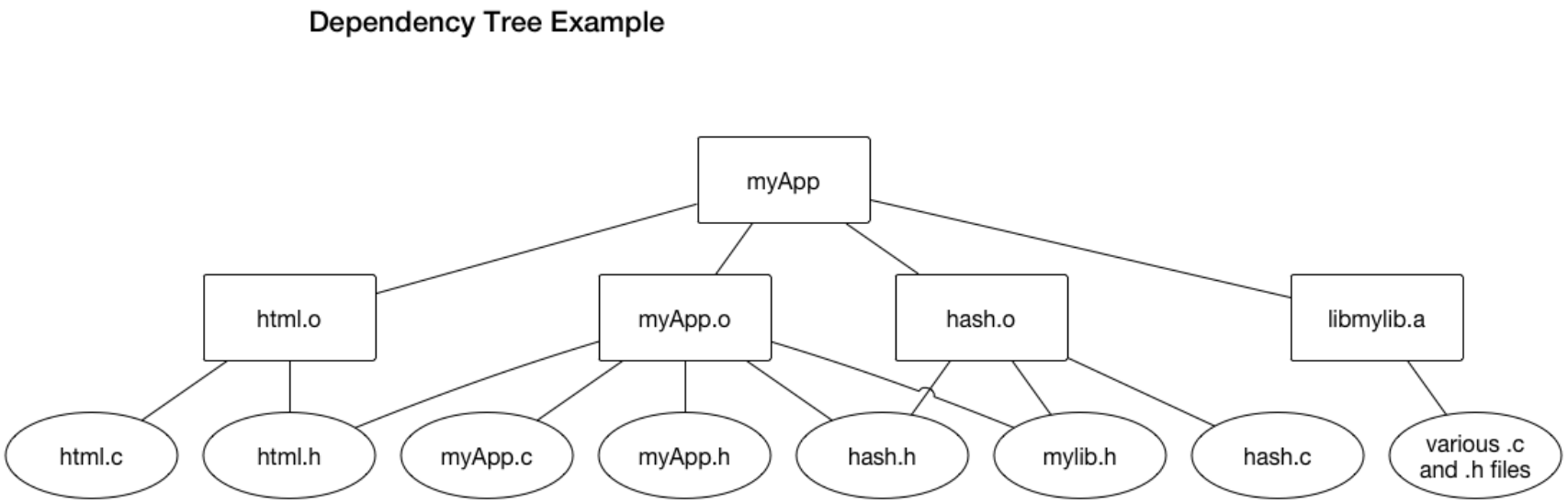
<команда>

Пример:

 # build an executable named myprog from myprog.c

```
all: myprog.c
        gcc -g -Wall -o myprog myprog.c

clean:
        $(RM) myprog
```

Makefile

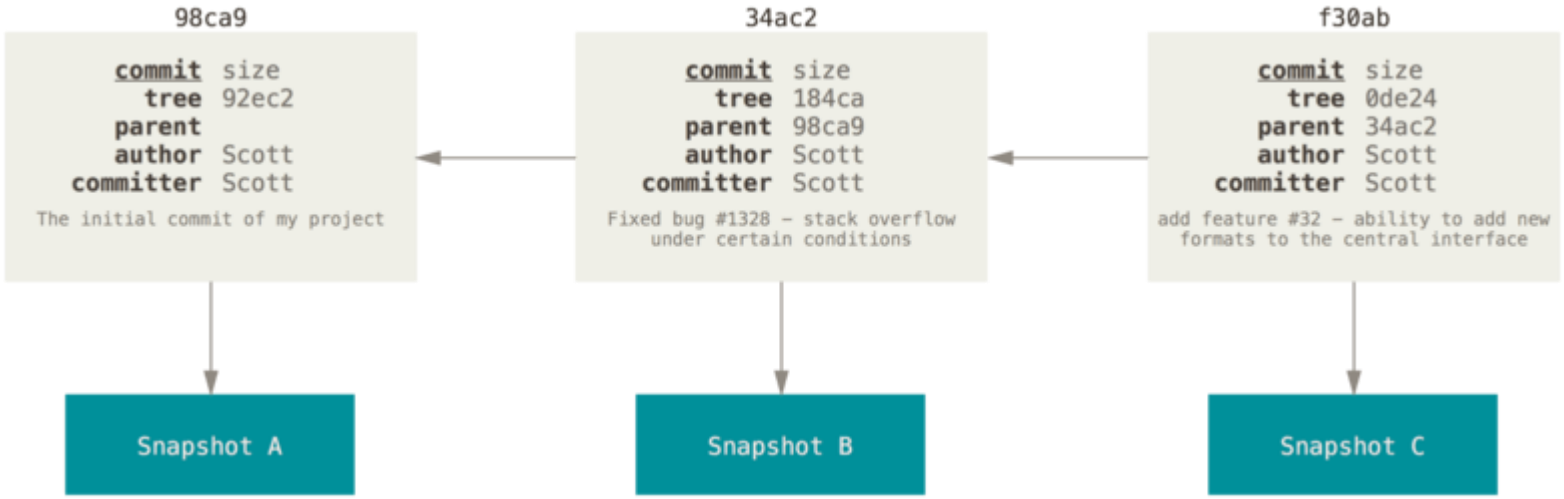


https://thayer.github.io/engs50/Notes/makefiles/extra/

Makefile

```
#  Makefile
CC = gcc
CFLAGS = -Wall -pedantic -std=c11
UTILDIR=../util/
UTILFLAG=-ltseutil
UTILLIB=$(UTILDIR)libmylib.a
UTILC=$(UTILDIR)file1.c $(UTILDIR)file2.c $(UTILDIR)file3.c $(UTILDIR)file4.c
UTILH=$(UTILC:.c=.h)

myApp:  myApp.o html.o hash.o $(UTILLIB)
	$(CC) $(CFLAGS) -o myApp myApp.o html.o hash.o -lmylib

myApp.o: myApp.c myApp.h html.h hash.h mylib.h
	$(CC) $(CFLAGS) -c myApp.c

html.o: html.c html.h
	$(CC) $(CFLAGS) -c html.c

hash.o: hash.c hash.h mylib.h
	$(CC) $(CFLAGS) -c hash.c

$(UTILLIB): $(UTILC) $(UTILH)
	cd $(UTILDIR); make
```

Ветвление в Git

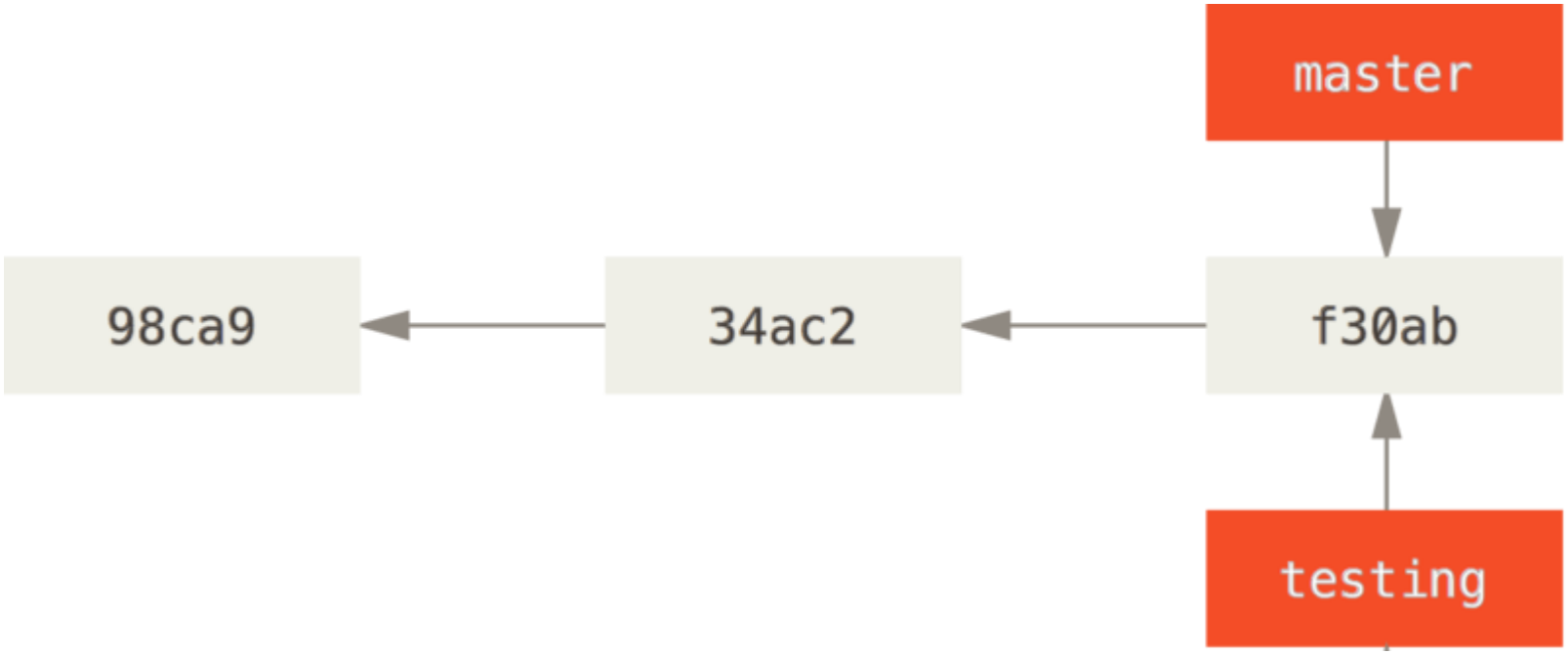


- Ветка (branch) в Git — это легко перемещаемый указатель на один из этих коммитов. Имя основной ветки по умолчанию в Git — ***master***

- Когда вы делаете коммиты, то получаете основную ветку, указывающую на ваш последний коммит. Каждый коммит автоматически двигает этот указатель вперед.

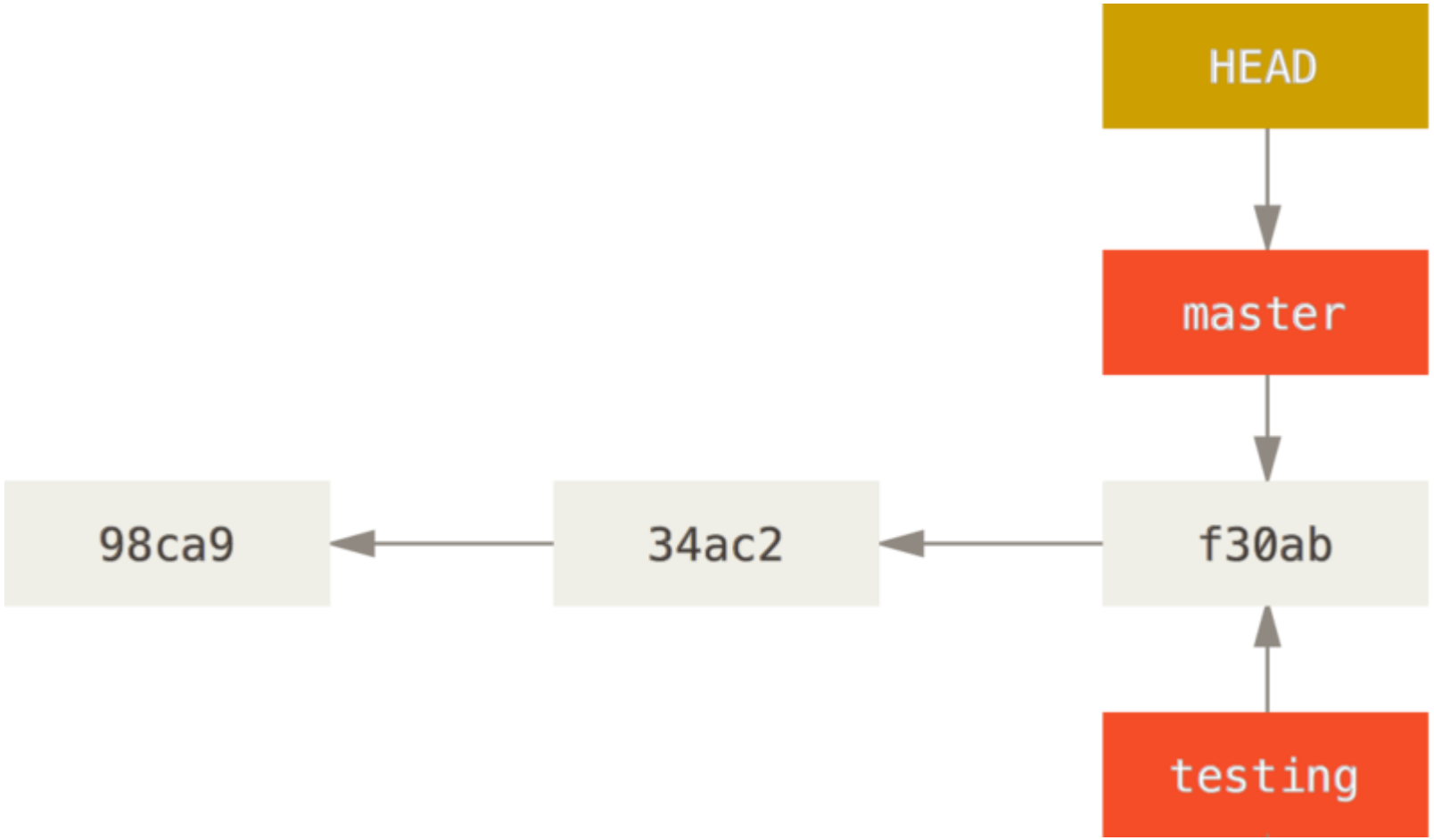
Ветвление в Git: создание новой ветки

- ***$ git branch testing***



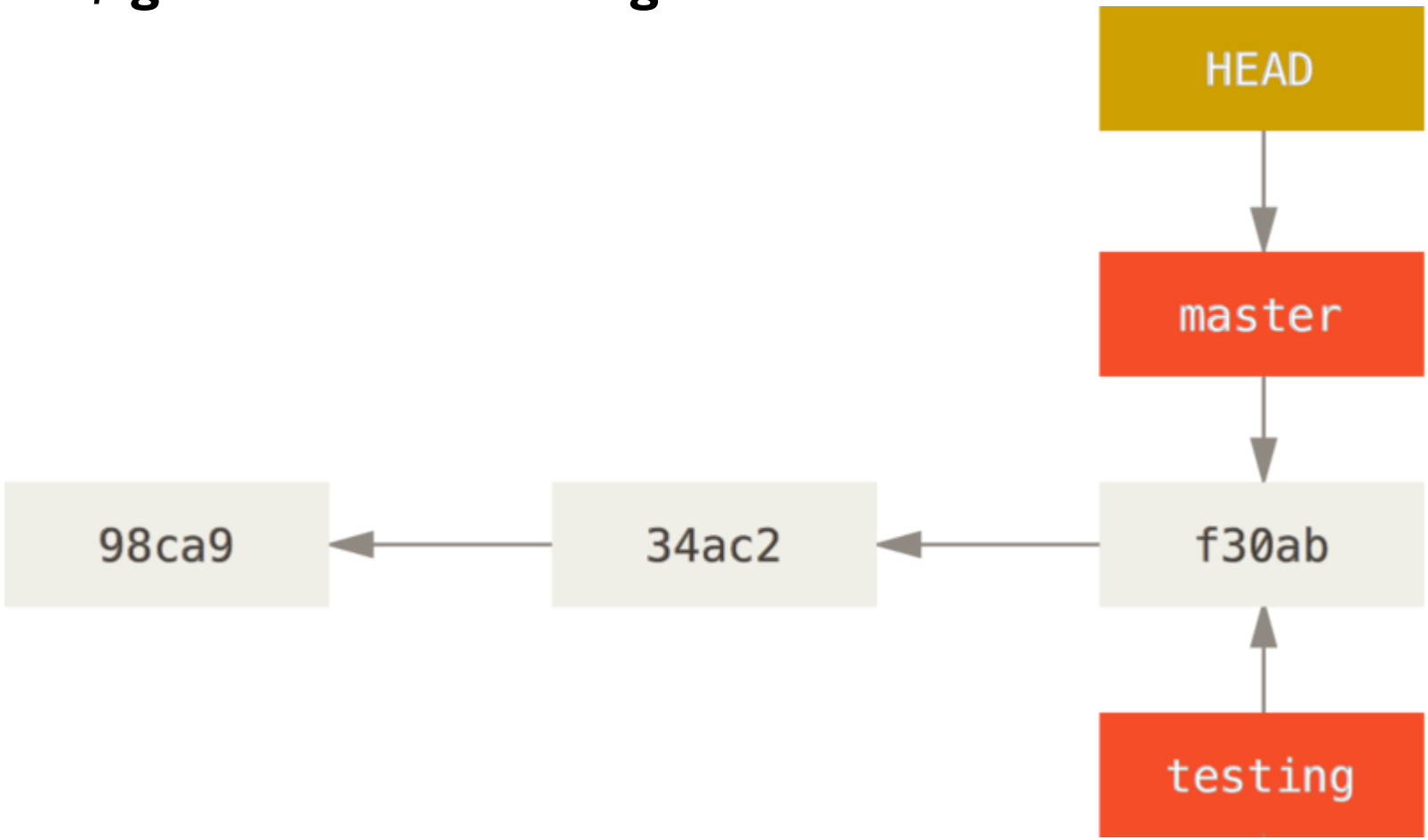
Команда ***git branch*** только создает новую ветку. Переключения не происходит.

Ветвление в Git: HEAD



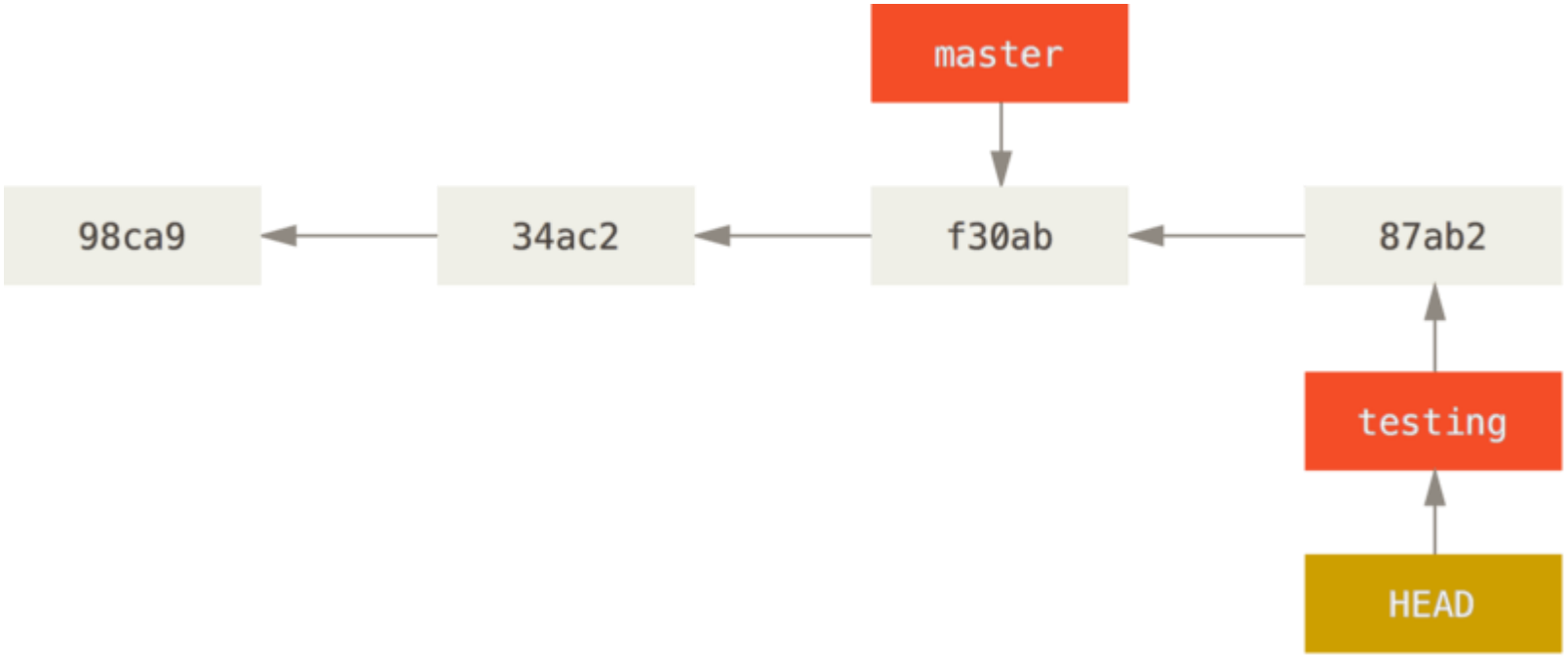
Ветвление в Git: HEAD

- ***$ git checkout testing***



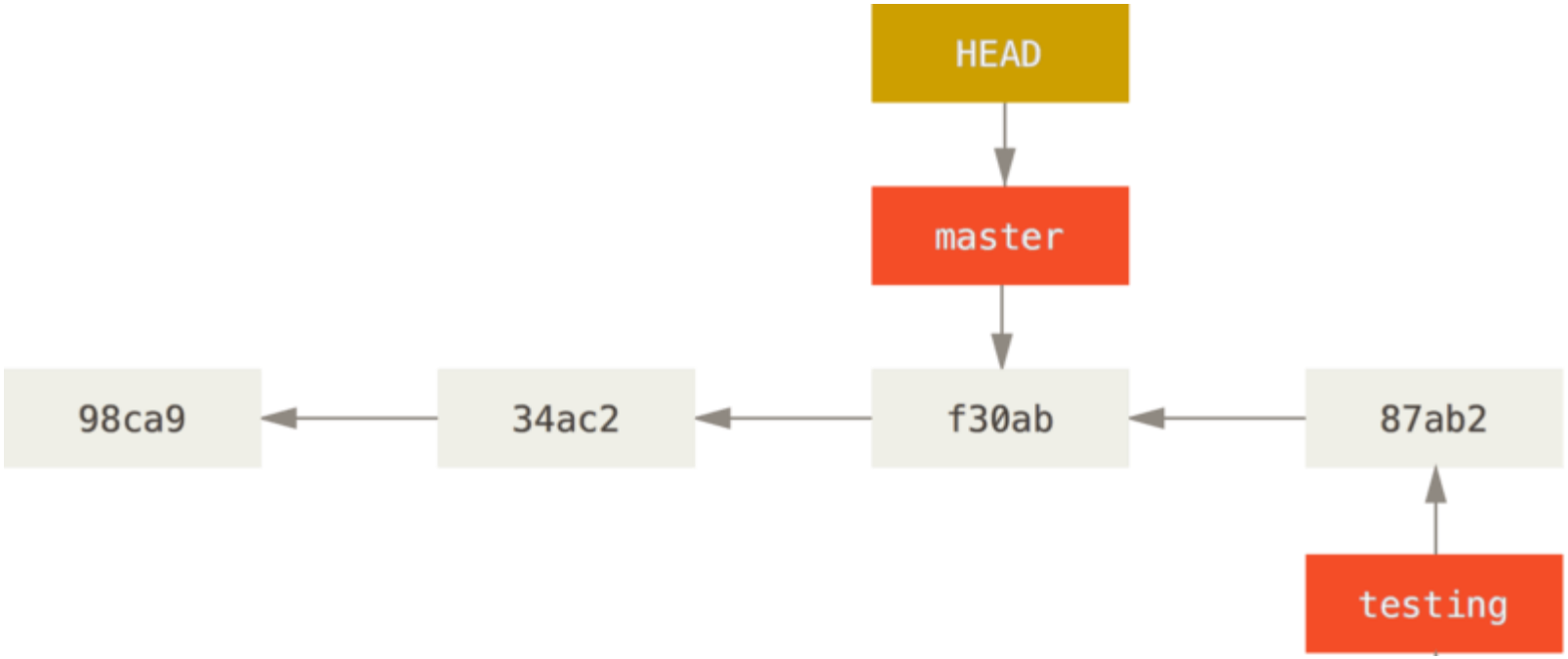
Ветвление в Git: branch

- ***$ vim test.rb***
- ***$ git commit -a -m 'made a change'***



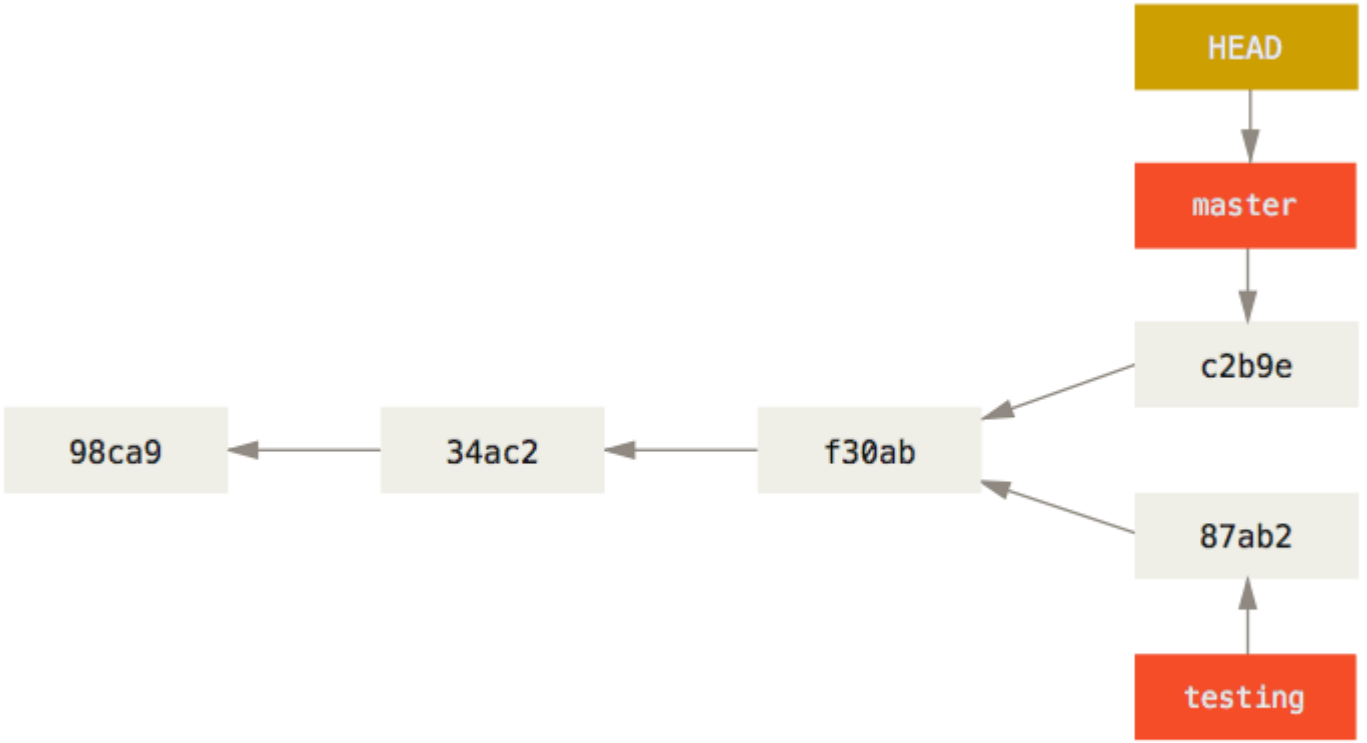
Ветвление в Git: branch

- ***$ git checkout master***



Ветвление в Git: branch

- ***$ vim test.rb***
- ***$ git commit -a -m 'made other changes'***



Ветвление в Git: branch

- ***$ git log --oneline --decorate --graph –all***

- ***$ git log --oneline --decorate --graph --all***
- **** c2b9e (HEAD, master) made other changes***
- ***| * 87ab2 (testing) made a change***
- ***|/***
- **** f30ab add feature #32 - ability to add new formats to the***
- **** 34ac2 fixed bug #1328 - stack overflow under certain conditions***
- **** 98ca9 initial commit of my project***

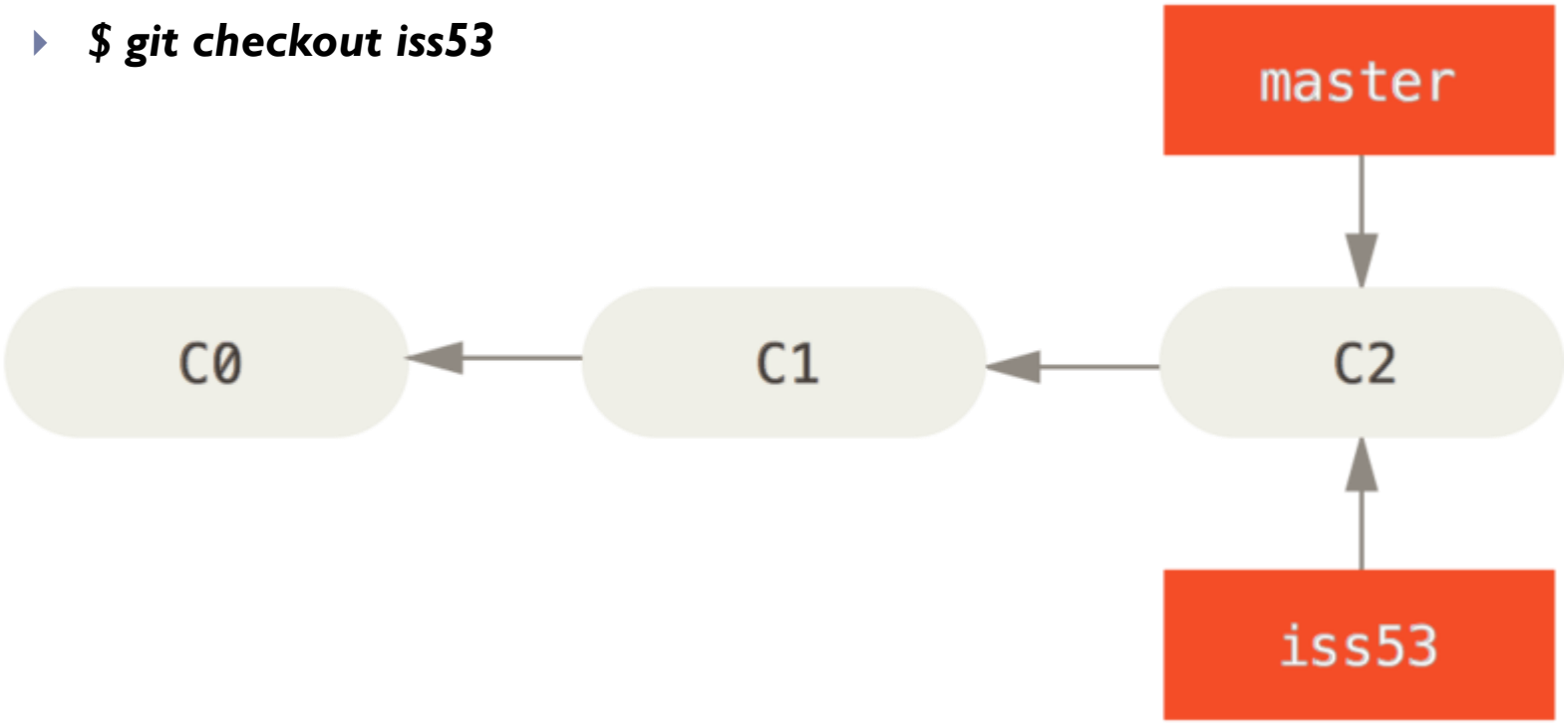
Создание и удаление веток совершенно не затратно, так как ветка в Git — это всего лишь файл, содержащий 40 символов контрольной суммы SHA-1 того коммита, на который он указывает.

Ветвление в Git: merge

- ***$ git checkout -b iss53***

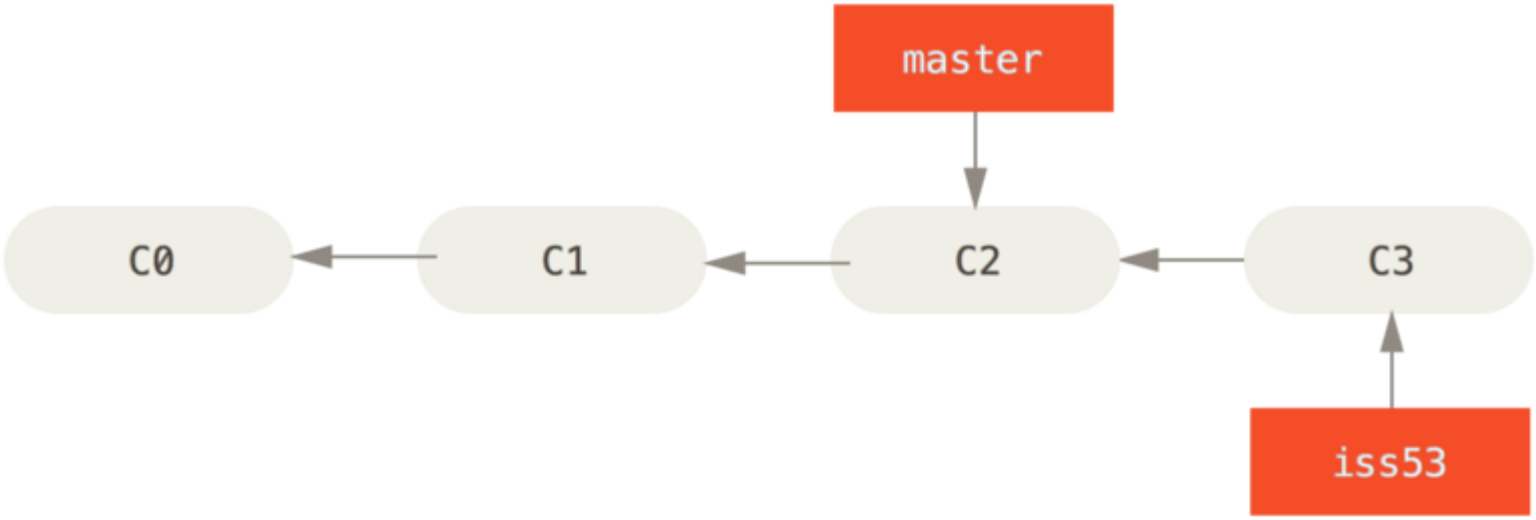
Switched to a new branch "iss53"

- ***$ git branch iss53***
- ***$ git checkout iss53***



Ветвление в Git: merge

- ***$ vim index.html***
- ***$ git commit -a -m 'added a new footer [issue 53]'***



Ветвление в Git: merge

- ***$ git checkout master***

Switched to branch 'master‘

- ***$ git checkout -b hotfix***

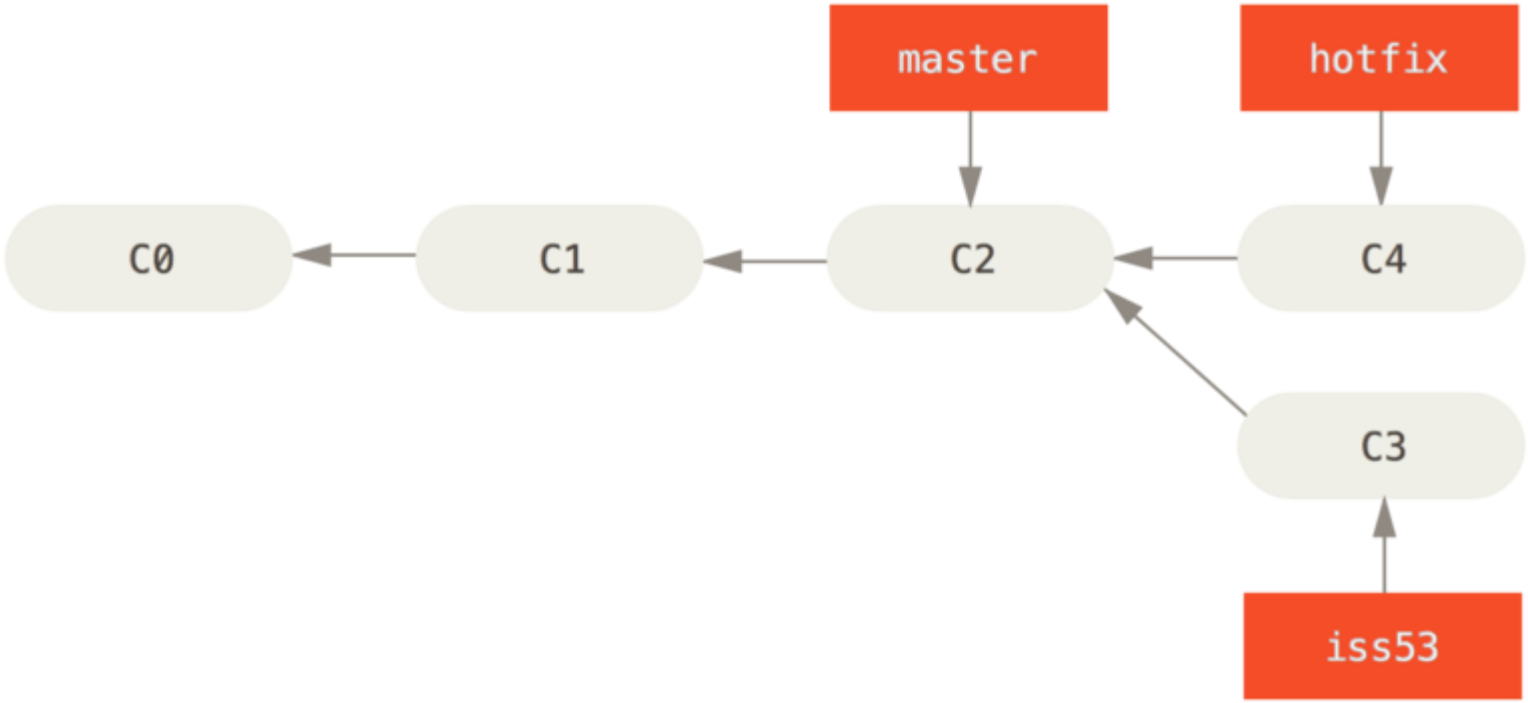
Switched to a new branch 'hotfix'

- ***$ vim index.html***
- ***$ git commit -a -m 'fixed the broken email address'***

[hotfix 1fb7853] fixed the broken email address

1 file changed, 2 insertions(+)

Ветвление в Git: merge



Ветвление в Git: fast-forward merge

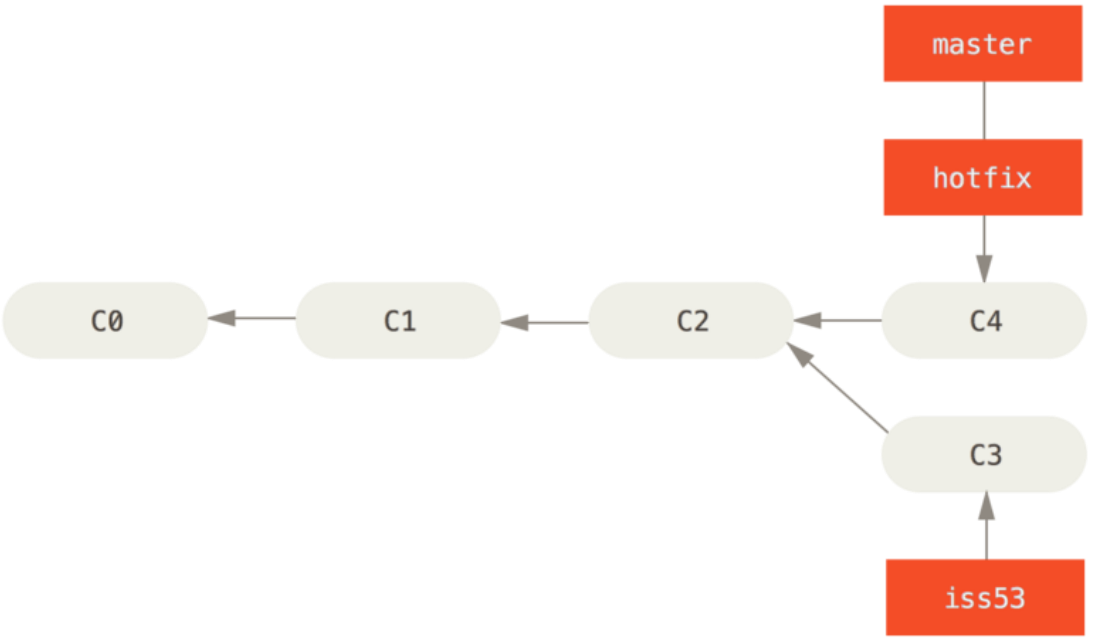
- ***$ git checkout master***
- ***$ git merge hotfix***

Updating f42c576..3a0874c

Fast-forward

index.html | 2 ++

1 file changed, 2 insertions(+)



Из-за того, что коммит, на который указывала ветка, которую вы слили, был прямым потомком того коммита, на котором вы находились, Git просто переместил указатель ветки вперед. Другими словами, если коммит сливается с тем, до которого можно добраться, двигаясь по истории прямо, Git упрощает слияние, просто перенося указатель метки вперед (так как нет разветвления в работе). Это называется ``fast-forward`` (перемотка).

Ветвление в Git: branch

- ***$ git branch -d hotfix***

Deleted branch hotfix (3a0874c).

Переключаем ветку и вернемся к работе над проблемой #53:

- ***$ git checkout iss53***

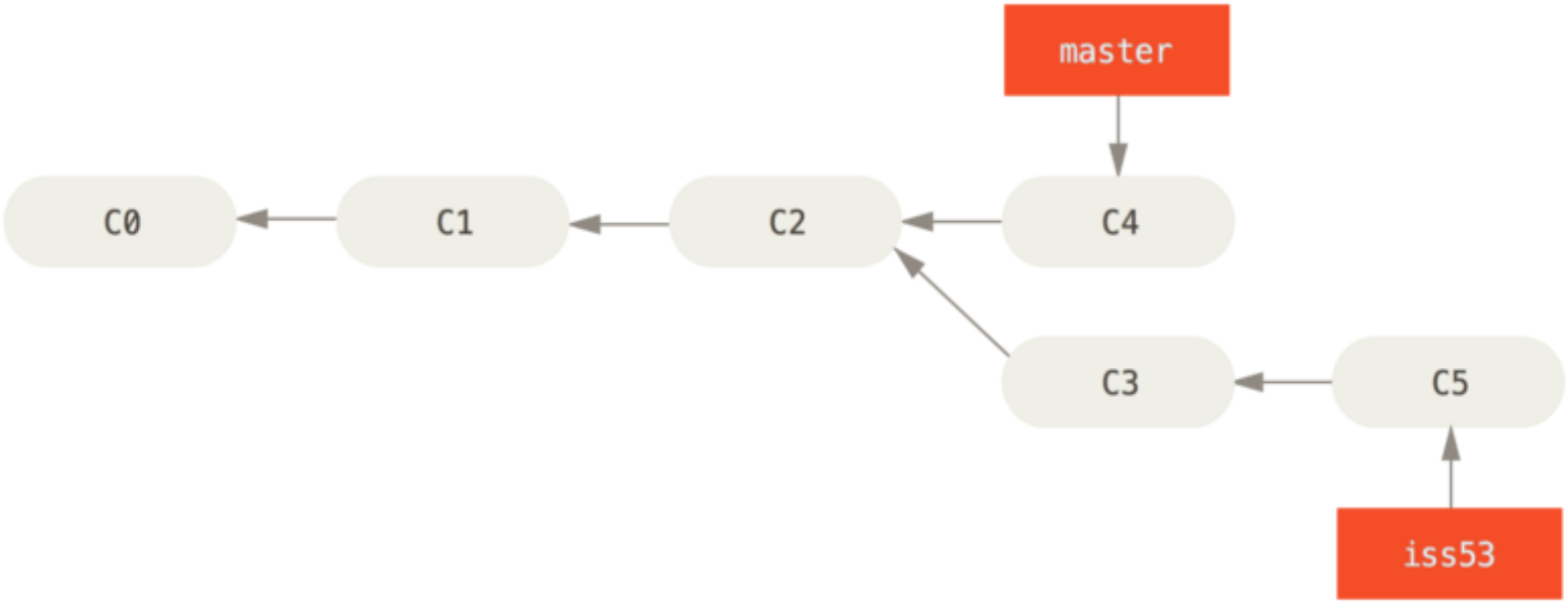
Switched to branch "iss53"

- ***$ vim index.html***
- ***$ git commit -a -m 'finished the new footer [issue 53]'***

[iss53 ad82d7a] finished the new footer [issue 53]

1 file changed, 1 insertion(+)

Ветвление в Git: branch



Ветвление в Git: recursive merge

Предположим, вы решили, что работа по проблеме #53 закончена, и ее можно влить в ветку master.

- ***$ git checkout master***

Switched to branch 'master'

- ***$ git merge iss53***

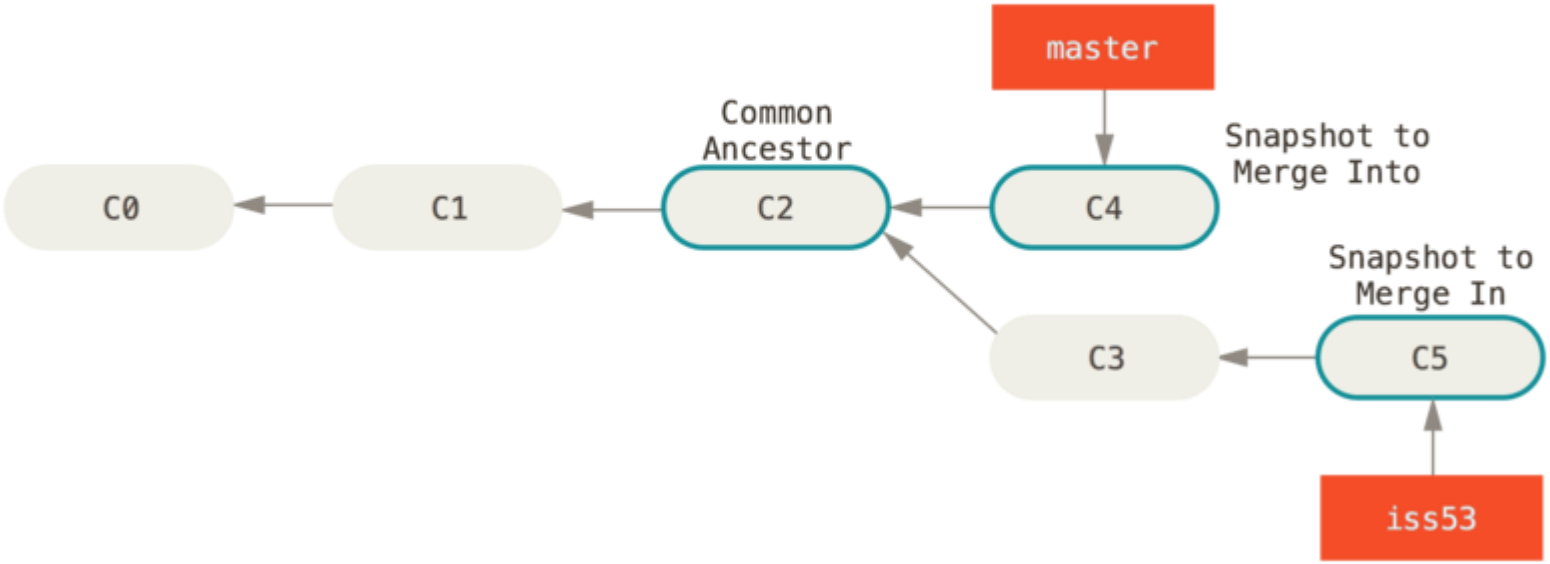
Merge made by the 'recursive' strategy.

index.html | 1 +

1 file changed, 1 insertion(+)

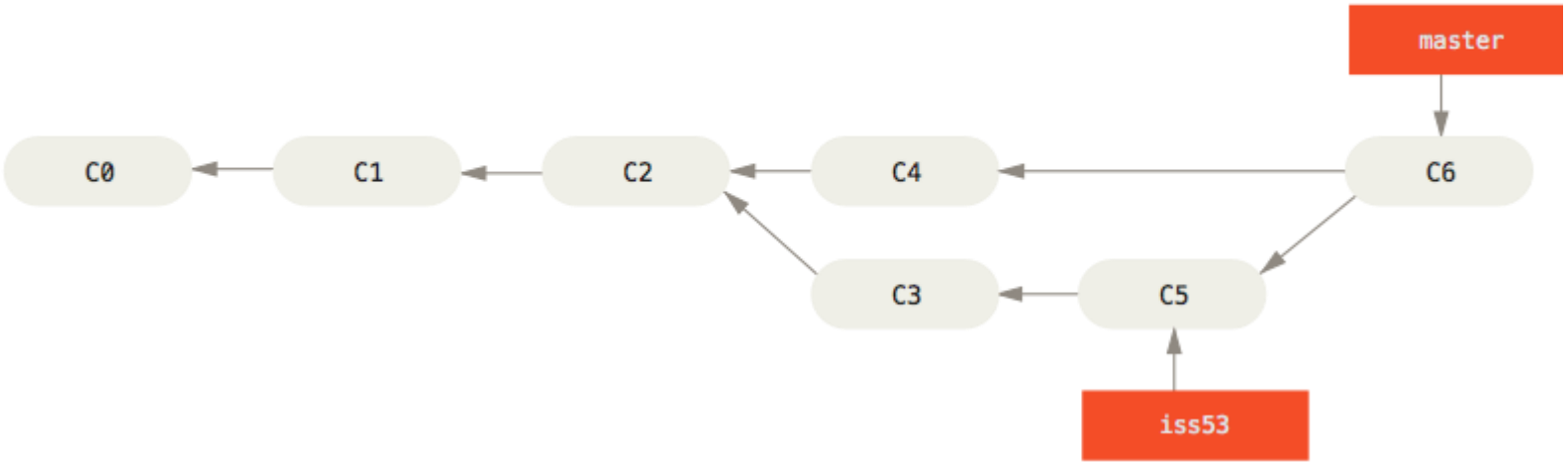
Так как коммит, на котором мы находимся, не является прямым потомком ветки, с которой мы выполняем слияние, Git придется немного потрудиться. В этом случае Git выполняет простое трехстороннее слияние двух снимков (snapshot) сливаемых веток и общего для двух веток родительского снимка.

Ветвление в Git: recursive merge



Ветвление в Git: recursive merge

Вместо того, чтобы просто передвинуть указатель ветки вперед, Git создает новый снимок-результат трехстороннего слияния, а затем автоматически делает коммит. Этот особый коммит называют коммитом слияния, так как у него более одного предка.



Теперь, когда работа влита, ветка iss53 больше не нужна. Вы можете закрыть вопрос в системе отслеживания ошибок и удалить ветку:

- ***$ git branch -d iss53***

Ветвление в Git: конфликты

Иногда процесс не проходит гладко. Если вы изменили одну и ту же часть одного и того же файла по-разному в двух объединяемых ветках, Git не сможет их чисто объединить.

- ***$ git merge iss53***

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

Git не создал коммит слияния автоматически. Он остановил процесс до тех пор, пока вы не разрешите конфликт.

Ветвление в Git: конфликты

- ***$ git status***

On branch master

You have unmerged paths.

  (fix conflicts and run "git commit")

Unmerged paths:

  (use "git add <file>..." to mark resolution)

    both modified: index.html

no changes added to commit (use "git add" and/or "git commit -a")

Ветвление в Git: конфликты

Все, где есть неразрешенные конфликты слияния, перечисляется как неслитое. Git добавляет в конфликтующие файлы стандартные пометки разрешения конфликтов, чтобы вы могли вручную открыть их и разрешить конфликты.

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

Разрешив каждый конфликт во всех файлах, запустите git add для каждого файла, чтобы отметить конфликт как решенный. Подготовка (staging) файла помечает его для Git как разрешенный конфликт.

Ветвление в Git: Управление ветками

Команда git branch делает несколько больше, чем просто создаёт и удаляет ветки. При запуске без параметров, вы получите простой список имеющихся у вас веток:

- ***$ git branch***

```
  iss53
* master
  testing
```

- ***$ git branch -v***

```
  iss53   93b412c fix javascript issue
* master  7a98805 Merge branch 'iss53'
  testing 782fd34 add scott to the author list in the readmes
```

Ветвление в Git: Управление ветками

Чтобы посмотреть те ветки, которые вы уже слили с текущей, можете выполнить команду ***git branch --merged***:

- ***$ git branch --merged***

iss53

* master

Те ветки из этого списка, перед которыми нет символа *, можно смело удалять командой ***git branch –d***

Чтобы увидеть все ветки, содержащие наработки, которые вы пока ещё не слили в текущую ветку, выполните команду ***git branch --no-merged***:

- ***$ git branch --no-merged***

testing

Ветвление в Git: Управление ветками

Так как она содержит ещё не слитые наработки, попытка удалить её командой ***git branch -d*** приведет к ошибке:

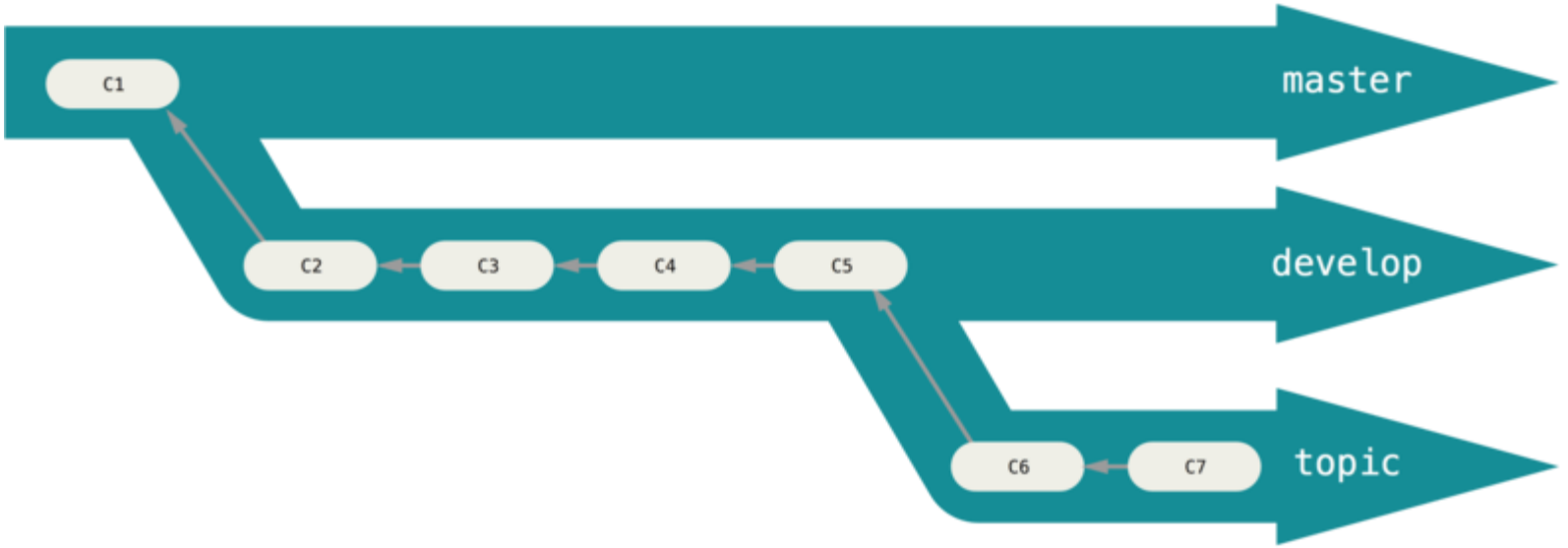
- ***$ git branch -d testing***

error: The branch 'testing' is not fully merged.

If you are sure you want to delete it, run 'git branch -D testing'.

Ветвление в Git: Управление ветками

В общем случае можно представить набор рабочих накопителей, в котором наборы коммитов перемещаются на более стабильный уровень только после полного тестирования:



Продолжение в следующей лекции…