CODE VERSIONING CON

Marco Console
console@dis.uniroma1.it

NOTA PRELIMINARE

- Il seguente materiale didattico è basato su un insieme di lucidi prepato dal **Prof. Leonardo Querzoni.**

- Ringrazio il **Prof. Leonardo Querzoni** per avermi concesso di usare il materiale da lui prodotto.

NOTA PRELIMINARE

Le slide di questa lezione sono state estratte dalle seguenti presentazioni:

- Introduction to Git - https://speakerdeck.com/schacon/introduction-to-git
- Wrangling Git - https://speakerdeck.com/schacon/wrangling-git
- Git & Git workflow models - http://www.slideshare.net/lemiorhan/git-and-git-workflow-models-as-catalysts-of-software-development

Per chi vuole approfondire consiglio di leggere:

- Scott Chacon and Ben Straub, “Pro Git”, https://progit.org

VERSION CONTROL

- Tre tipici problemi nello sviluppo di progetti software:
  - **Backup**
    - Tenere traccia delle successive modifiche del codice.
  - **Organizzazione**
    - Il codice e le risorse devono essere organizzate.
  - **Collaborazione**
    - Più di un programmatore può modificare il codice.

VERSION CONTROL

- Un approccio “pratico”:
  - Effettuare copie di backup manualmente
- Limiti:
  - funziona solo su piccola scala
  - troppo facile sbagliare
    - “Dove è finita la versione del il mese scorso?”
  - …

VERSION CONTROL

- Per risolvere questi problemi gli sviluppatori fanno uso di Version Control Systems (VCS):

- Cosa fanno per noi:
  - Gestiscono la storia dei cambiamenti apportati ad ogni singolo file
  - Permettono di recuperare una qualsiasi versione precedente
  - I cambiamenti sono annotati
  - Permettono una evoluzione non-lineare del codice (branching)
  - Permettono di definire l’evoluzione delle versioni del software
  - Gestiscono lo sviluppo concorrente

VERSION CONTROL

- Per risolvere questi problemi gli sviluppatori fanno uso di Version Control Systems (VCS):

- Tre tipologie:
  - Local Version Control - RCS, SCCS
  - Centralized Version Control - CVS, Subversion, Perforce
  - Distributed Version Control - Git, Mercurial, Bazaar

VERSION CONTROL

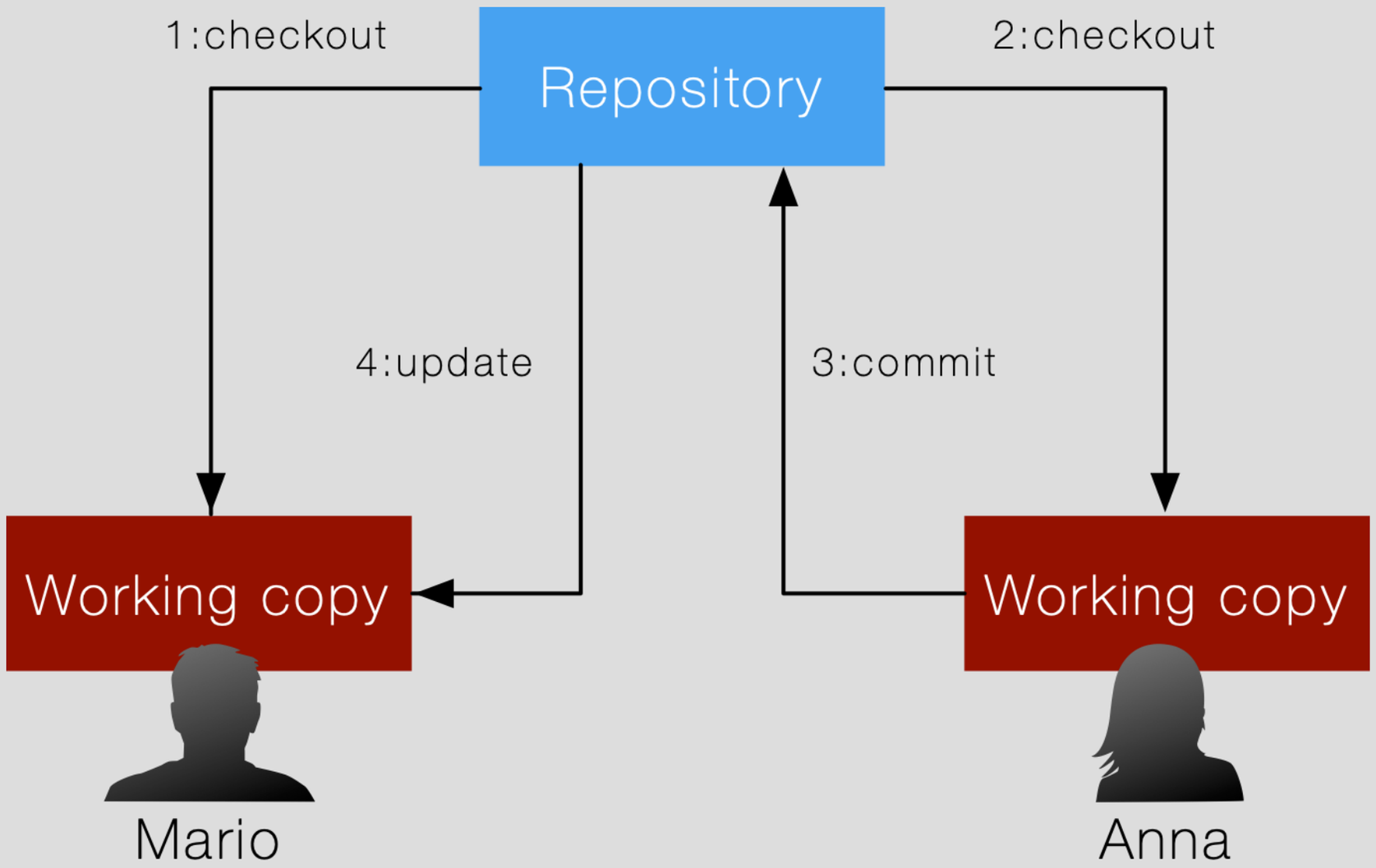
- Concetti di base
  - **Repository**
    - Il posto dove risiedono tutti i file gestiti dal sistema
    - Contiene tutte le versioni di tutti i file che costituiscono ogni singolo progetto
    - Può essere condiviso da più persone
  - **Working copy**
    - È una copia del repository
    - Ogni cambiamento al progetto avviene su una working copy
    - Una working copy è utilizzata da un singolo utente
    - Contiene dei metadati che aiutano il sistema a tenere traccia della relazione tra il contenuto della working copy e quello del repository

VERSION CONTROL

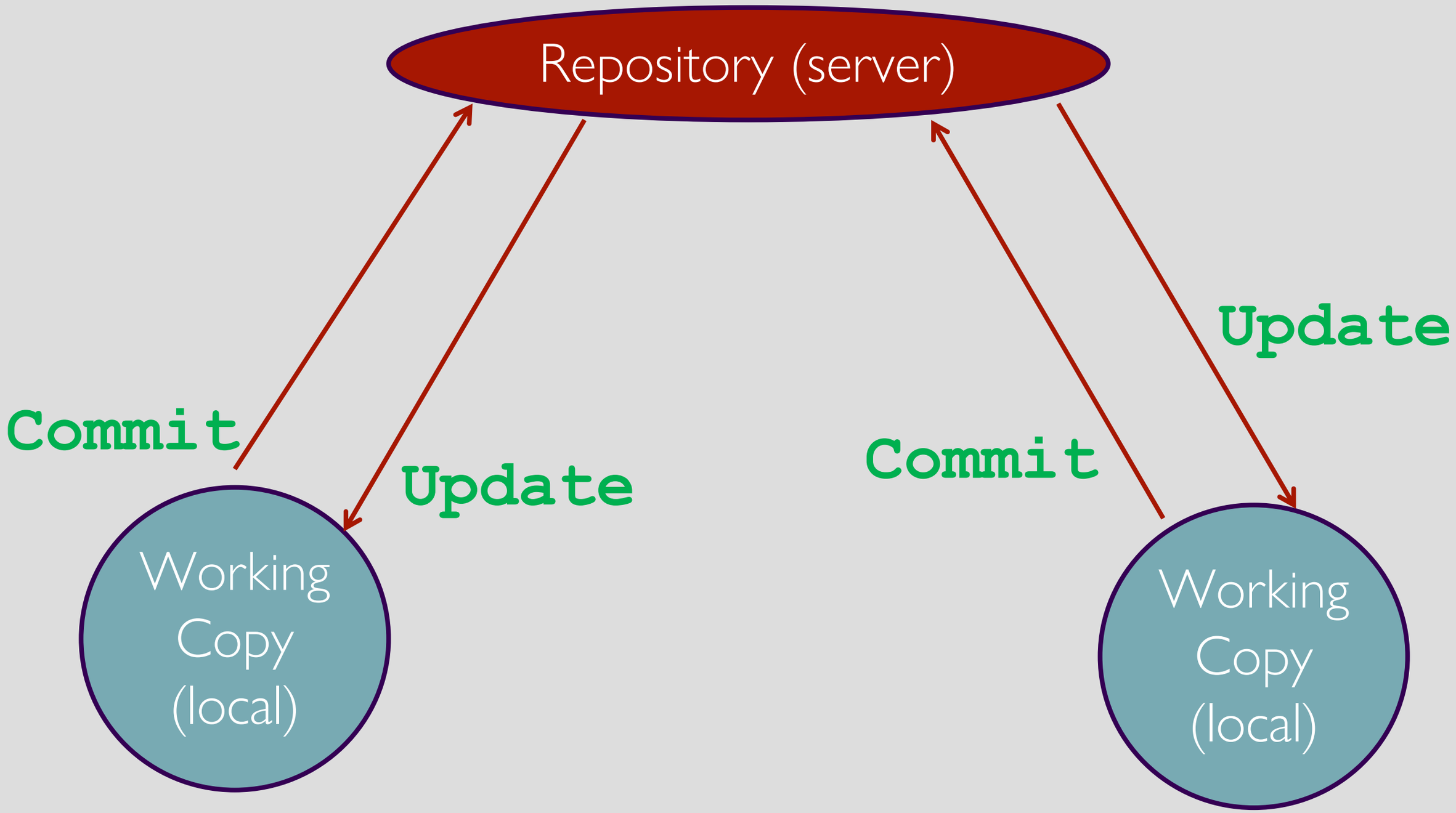
- Operazioni
  - **Checkout**
    - È l’operazione che crea una working copy a partire dal contenuto del repository
  - **Update**
    - È l’operazione che aggiorna il contenuto della working copy a partire dal repository
  - **Commit**

- È l’operazione che aggiorna il contenuto del repository con le modifiche apportate alla working copy
- Ogni commit è associato ad un messaggio il cui contenuto specifica i cambiamenti apportati al repository

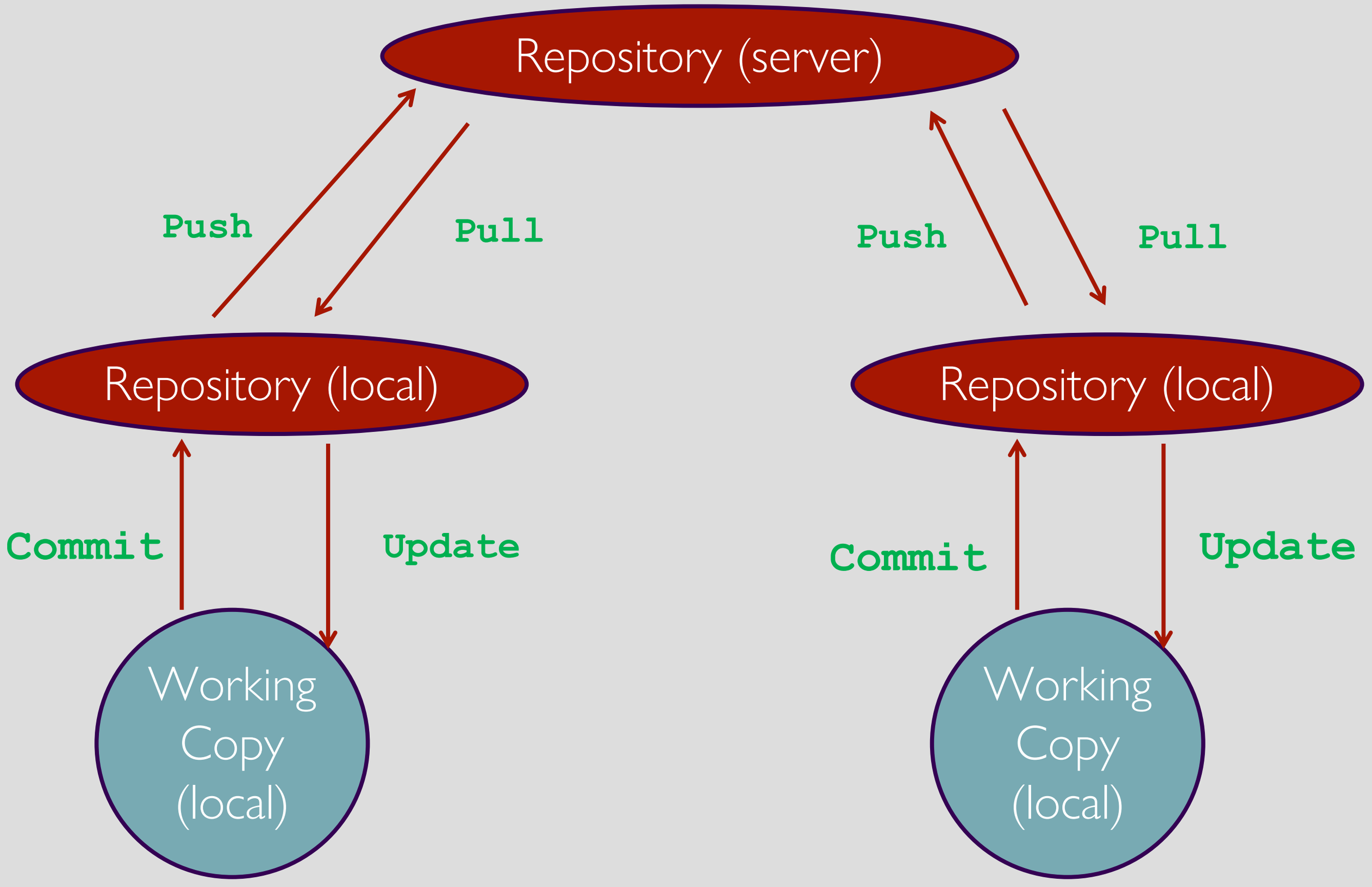
VERSION CONTROL



CENTRALIZED VCS



DISTRIBUTED VCS



CENTRALIZED VS DISTRIBUTED

- **Centralized:**
  - Facile da comprendere e usare.
  - Richiede spazio limitato all’utente.
  - L’uso concorrente può facilmente dar vita a inconsistenze.
- **Distributed:**
  - Più complesso da imparare.
  - L’utente scarica tutta la storia della repository sulla macchina locale.
  - Maggiore supporto per l’uso concorrente.

GIT

GIT – INTRODUZIONE

- GIT è un VCS di tipo distribuito.
- Creato da Linus Torvalds (Linux) per il kernel Linux
- Il significato del nome dipende da come va la giornata
  - dal README del primo commit…
  - https://github.com/git/git/blob/e83c5163316f89bfbde7d9ab23ca2e25604af290/README

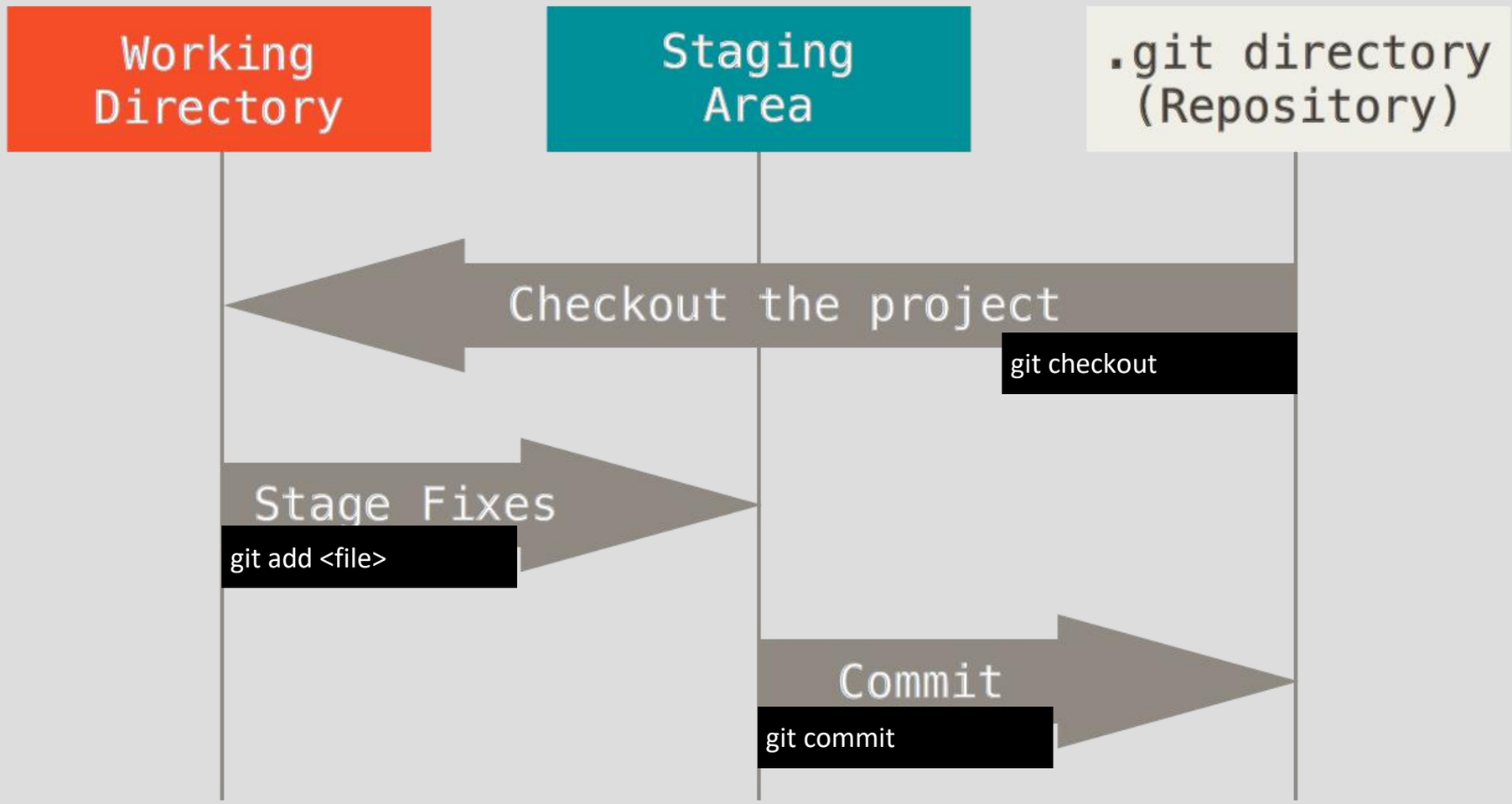
```
        GIT - the stupid content tracker

"git" can mean anything, depending on your mood.

 - random three-letter combination that is pronounceable, and not
   actually used by any common UNIX command.  The fact that it is a
   mispronounciation of "get" may or may not be relevant.
 - stupid. contemptible and despicable. simple. Take your pick from the
   dictionary of slang.
 - "global information tracker": you're in a good mood, and it actually
   works for you. Angels sing, and a light suddenly fills the room.
 - "goddamn idiotic truckload of sh*t": when it breaks

This is a stupid (but extremely fast) directory content manager.  It
```

GIT – CONCETTI FONDAMENTALI

Git si basa su quattro concetti principali:

- **Git Directory**
  - È il repository (locale)
- **Remotes**
  - Il\i repository remoti
- **Working Directory**
  - La cartella di lavoro locale
- **Staging Area**
  - È un file, contenuto nel repository, che elenca le modifiche che verranno rese **stabili** dal prossimo commit

Il tipico flusso di lavoro di una sessione Git



VERSIONAMENTO

I VCS classici lavorano per differenze (delta)

- Ogni modifica genera una differenza rispetto alla versione del file nel repository

- Il commit aggiunge nel repository le differenze

- Quando si deve ricostruire la versione attuale il VCS parte dalla versione originale ed applica le modifiche registrate nel repository

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

README.txt

hello.c

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

README.txt

hello.c

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

README.txt

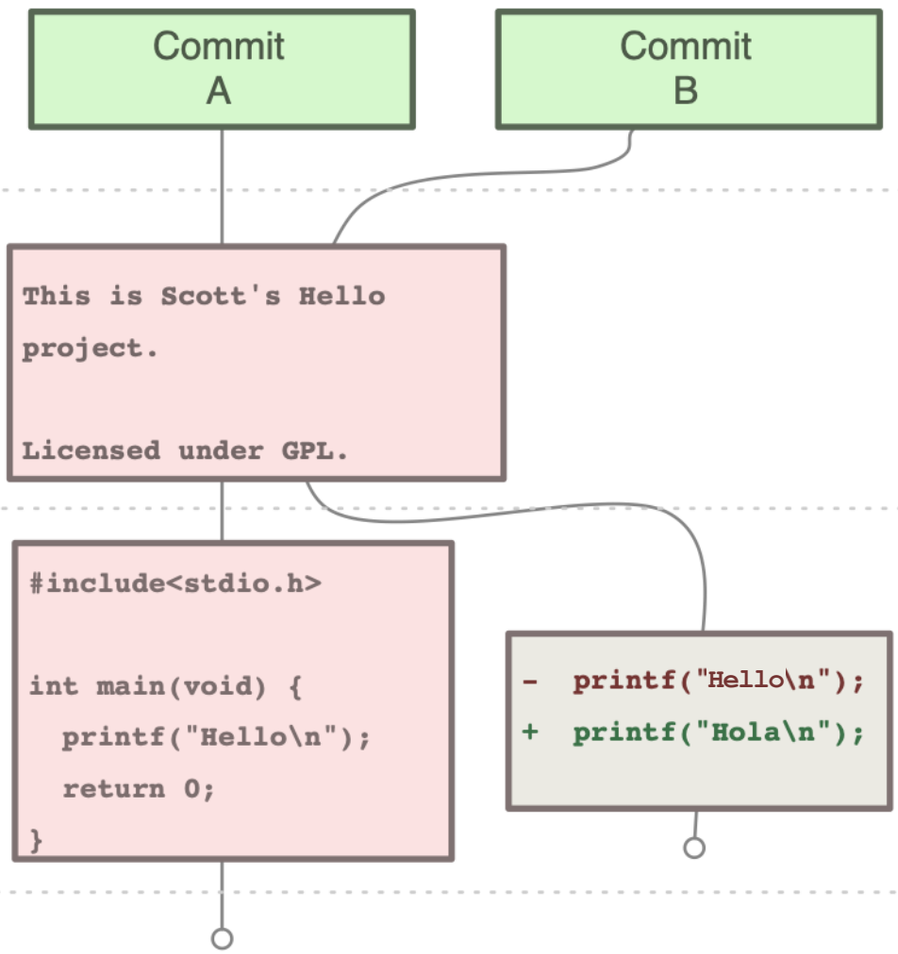
hello.c

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

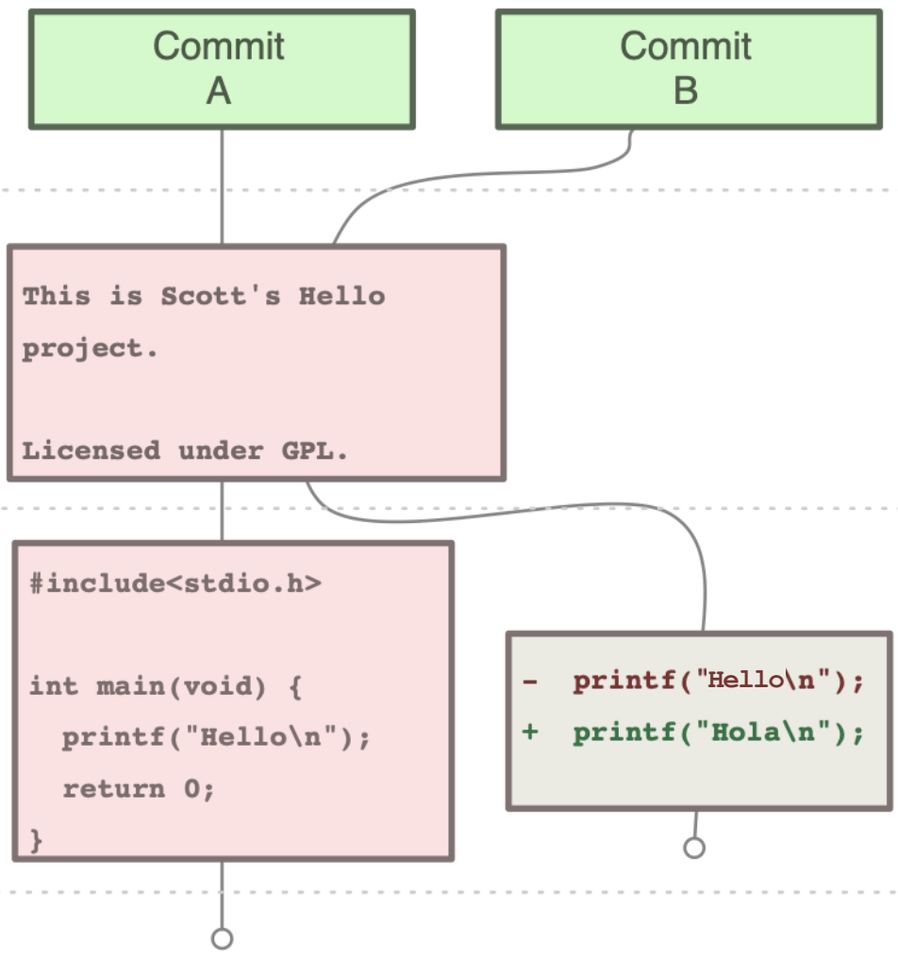
hello.c

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

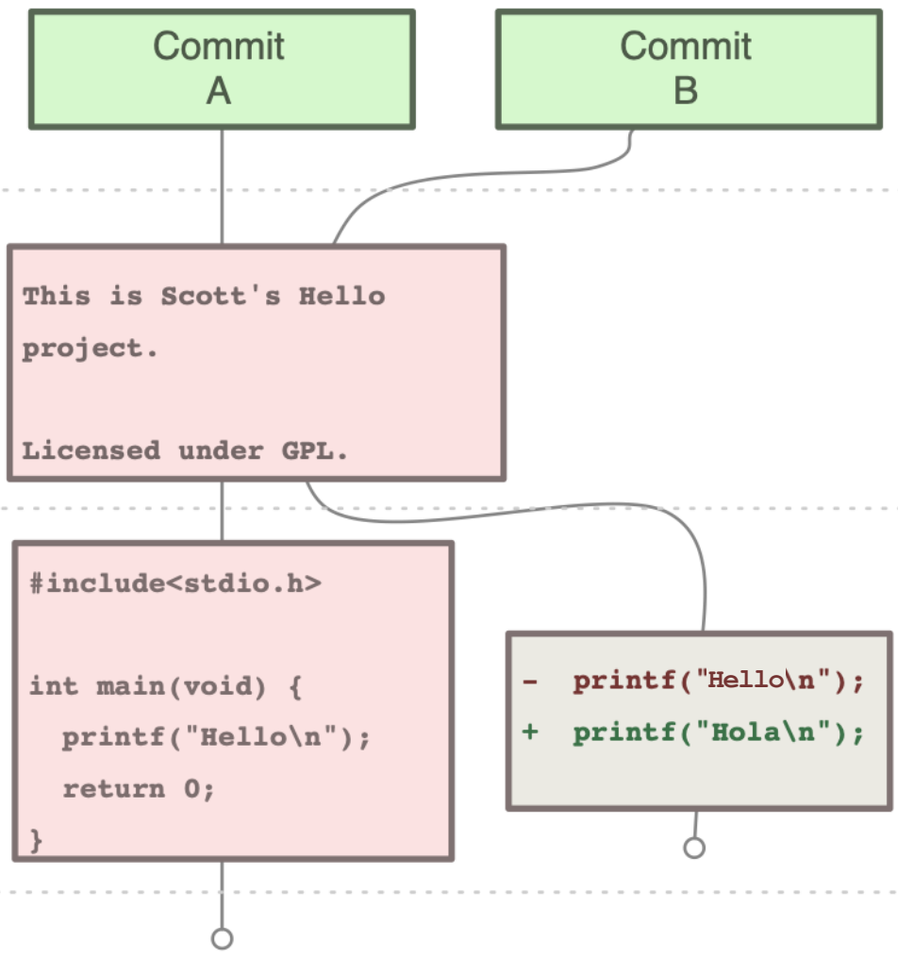
hello.c

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

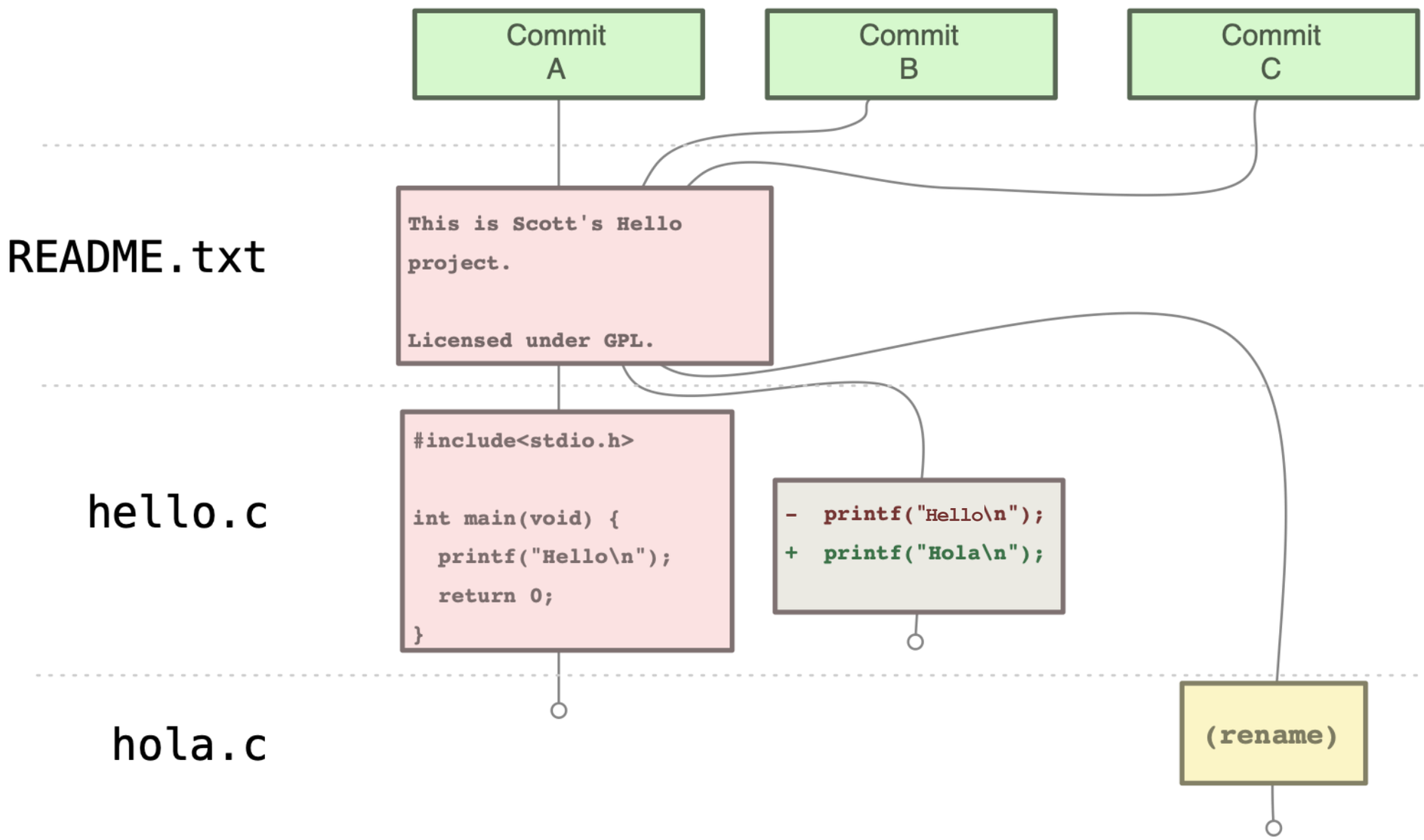
hello.c

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

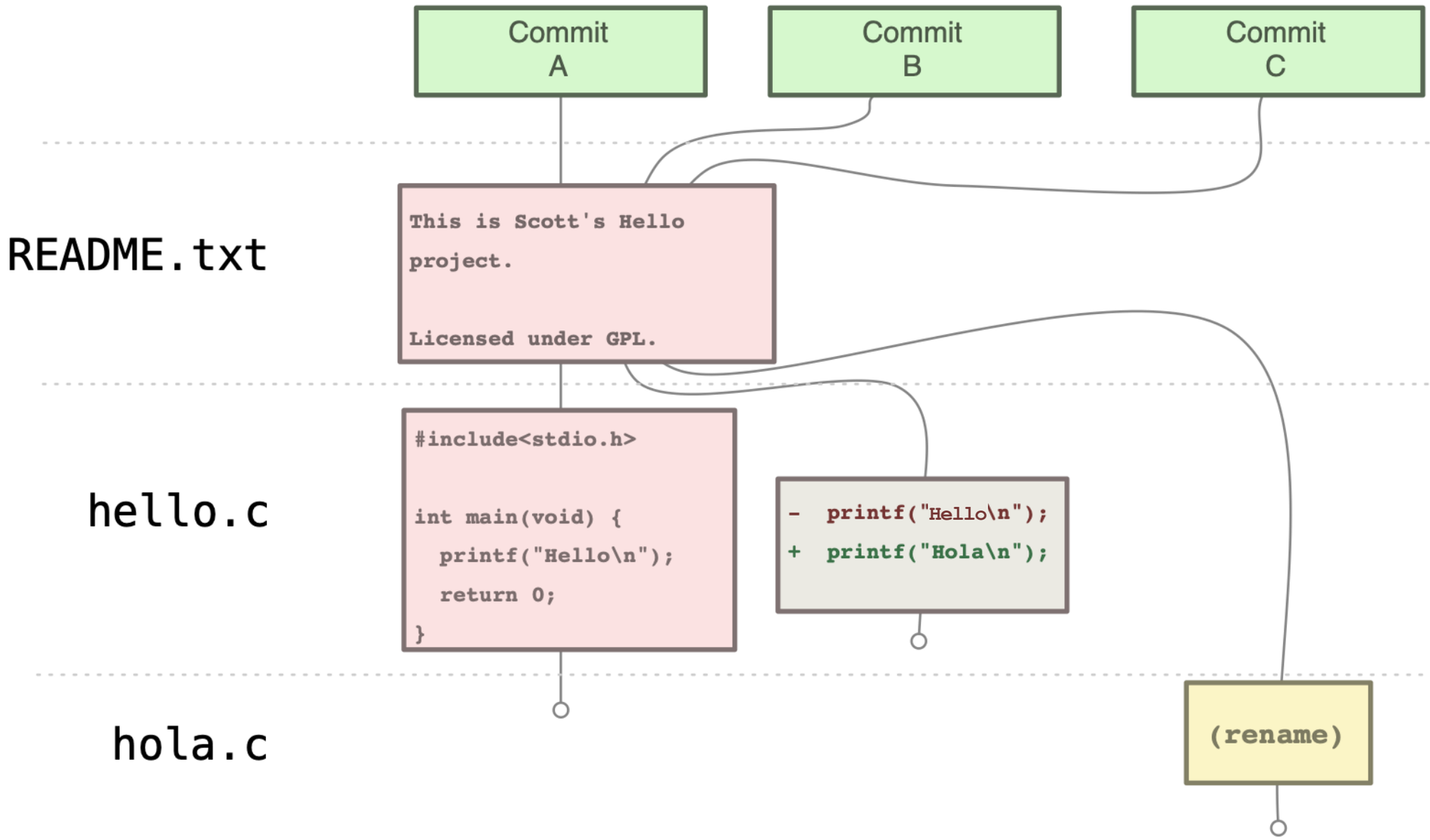


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

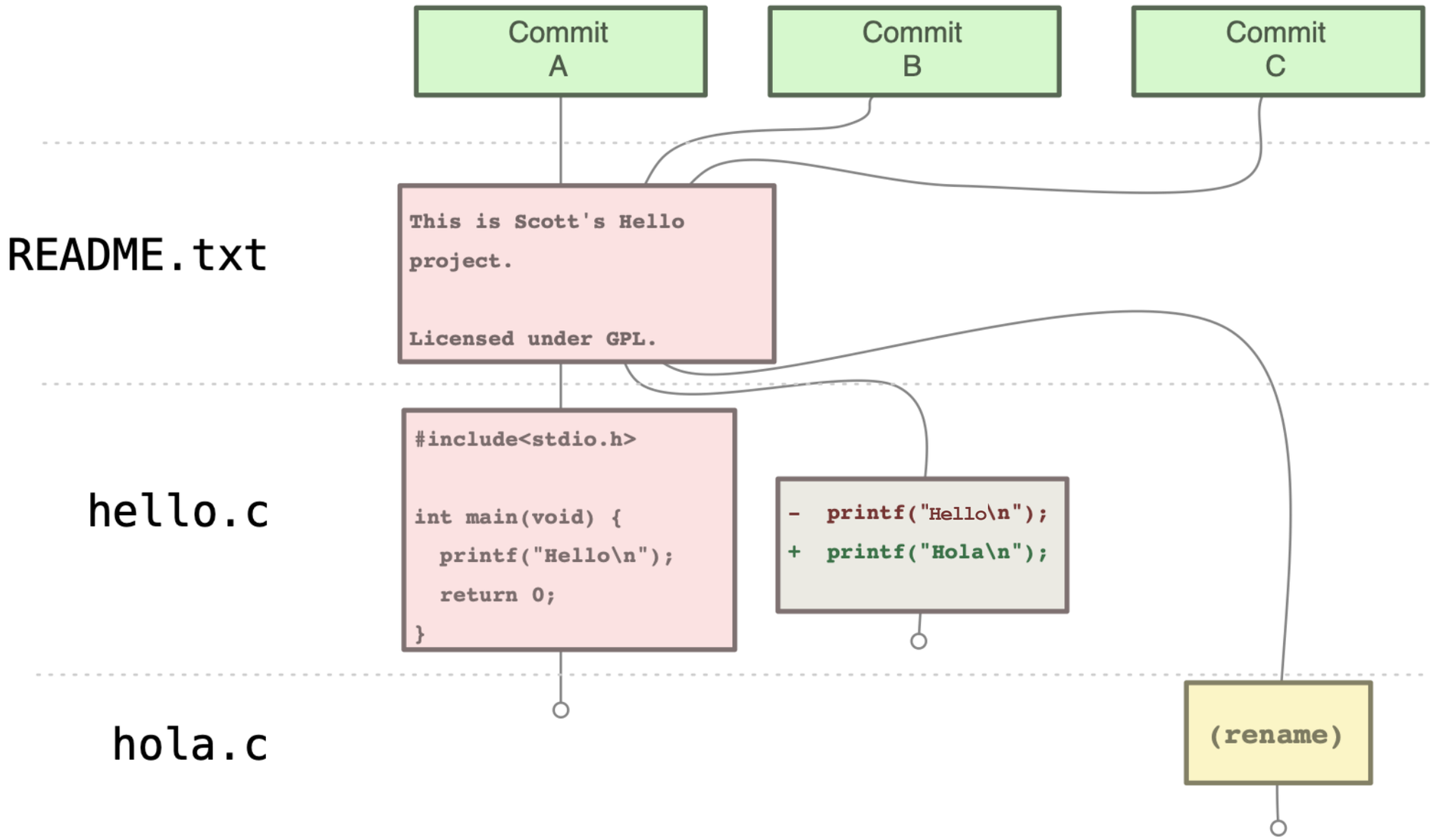


README.txt

```
This is Scott's
Hola project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

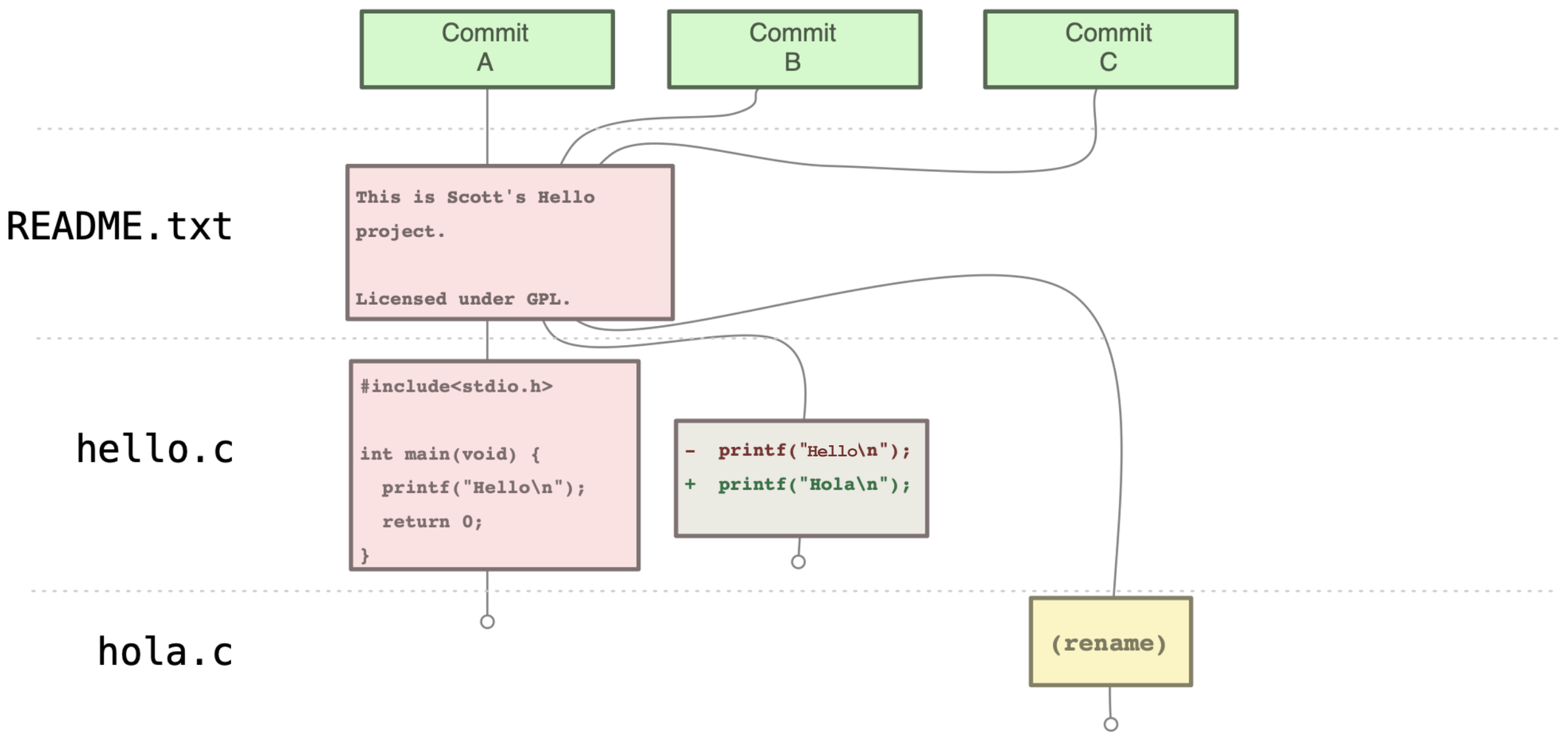
```
This is Scott's
Hola project.

Licensed under GPL.
```

hola.c

```
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

hello.c

```
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

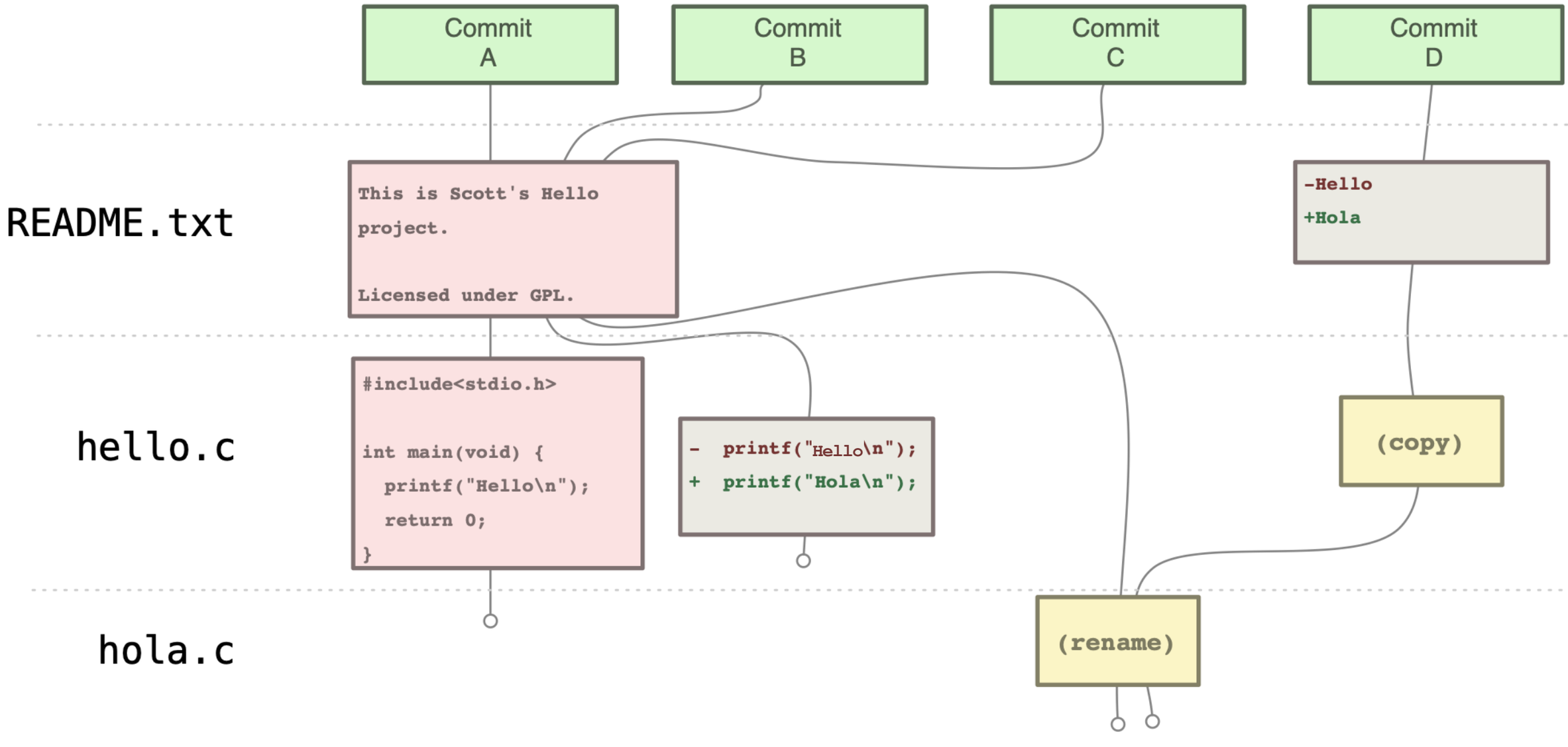
```
This is Scott's
Hola project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



VERSIONAMENTO IN GIT

Git utilizza degli snapshot

- Ogni nuova versione di un file viene memorizzato nella sua versione completa

- Per ridurre l’uso di spazio i dati vengono compressi

- Dati identici per file diversi vengono memorizzati una sola volta

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

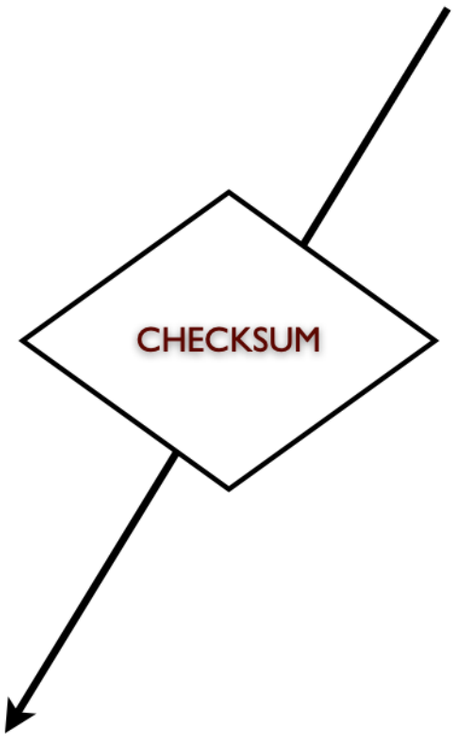
```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```



c3d

```
This is Scott's
Hello project.

Licensed under
GPL.
```

f13

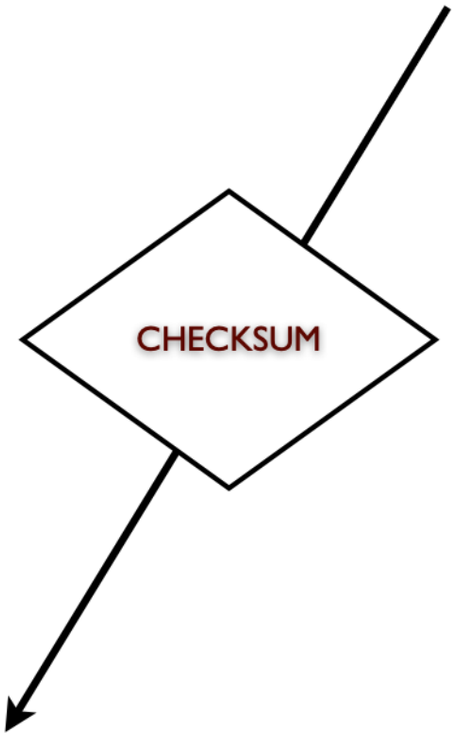
```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```



c3d

```
This is Scott's
Hello project.

Licensed under
GPL.
```

f13

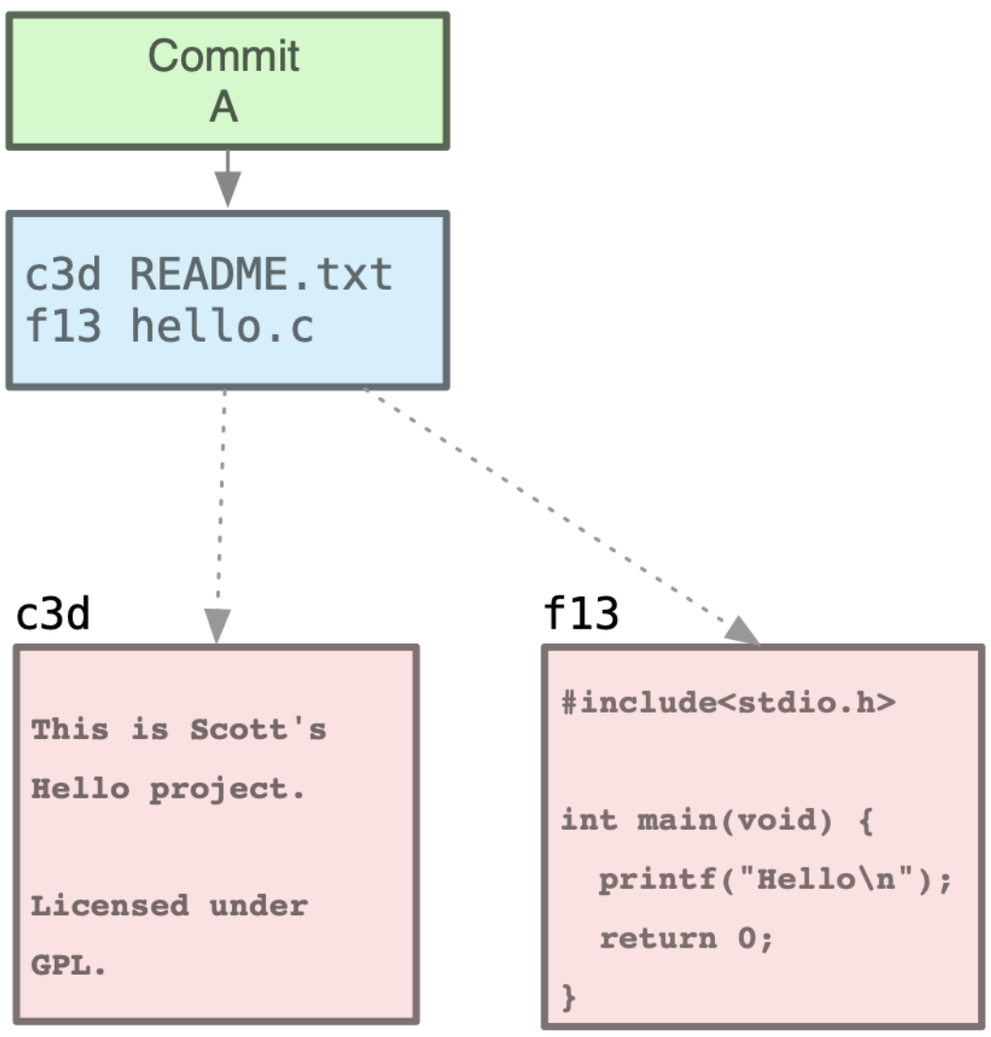
```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

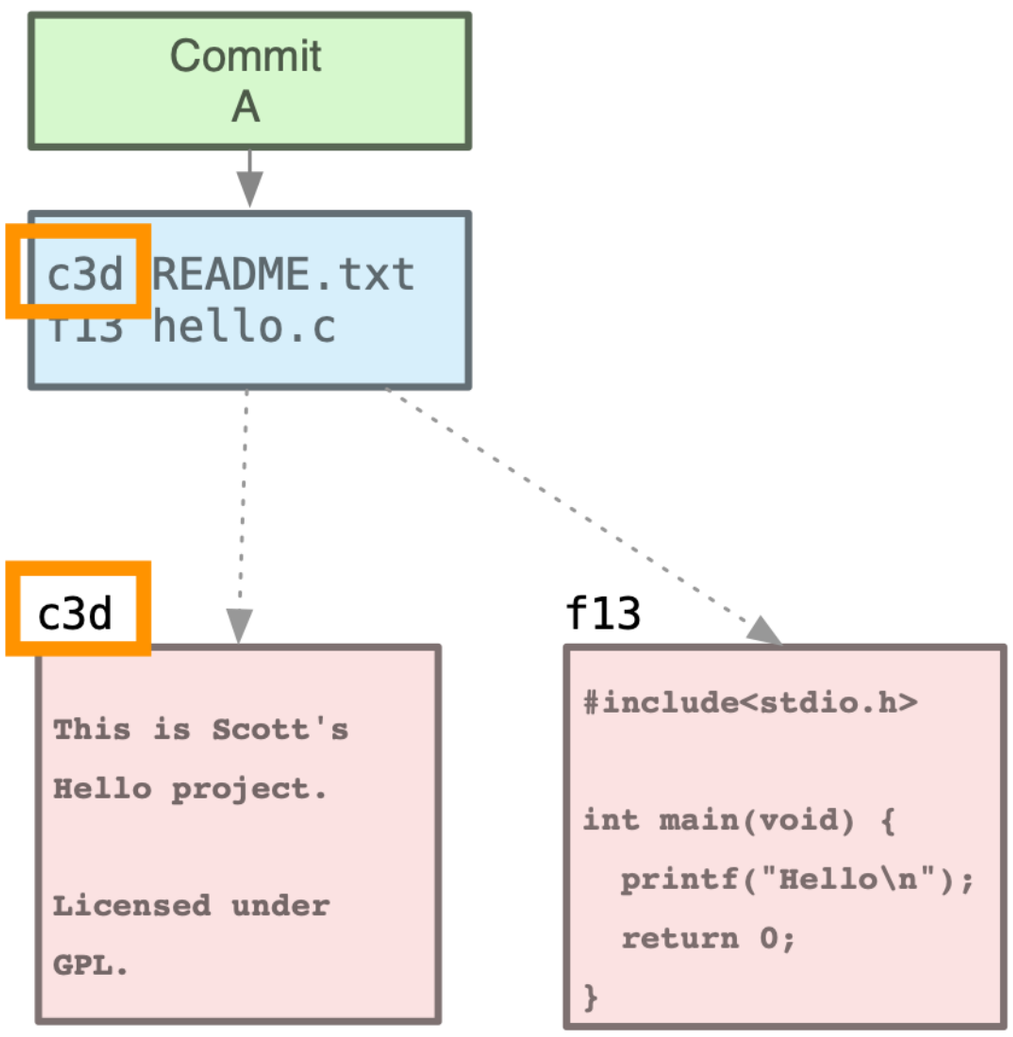


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

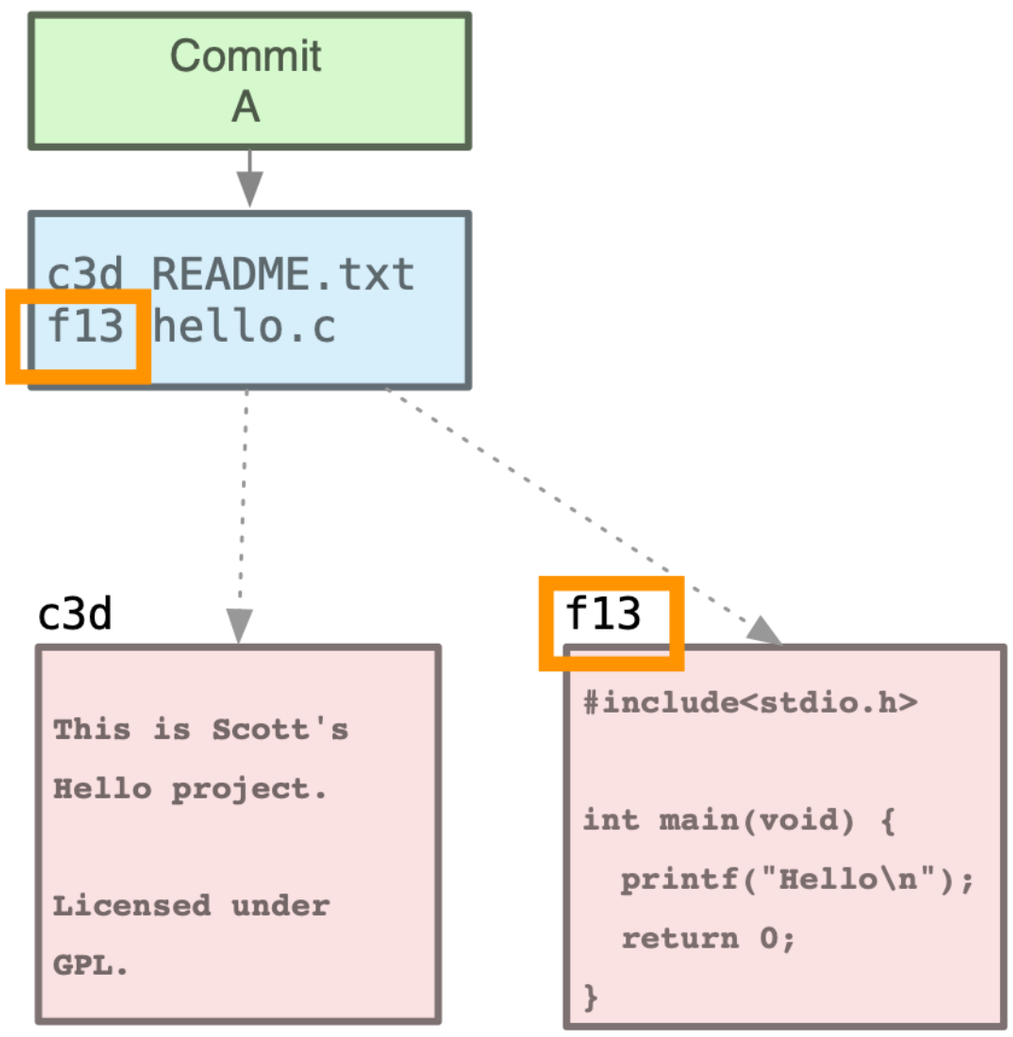


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```



README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

Commit
A

```
c3d README.txt
f13 hello.c
```

c3d

```
This is Scott's
Hello project.

Licensed under
GPL.
```

f13

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

Commit
A

```
c3d README.txt
f13 hello.c
```

c3d

```
This is Scott's
Hello project.

Licensed under
GPL.
```

f13

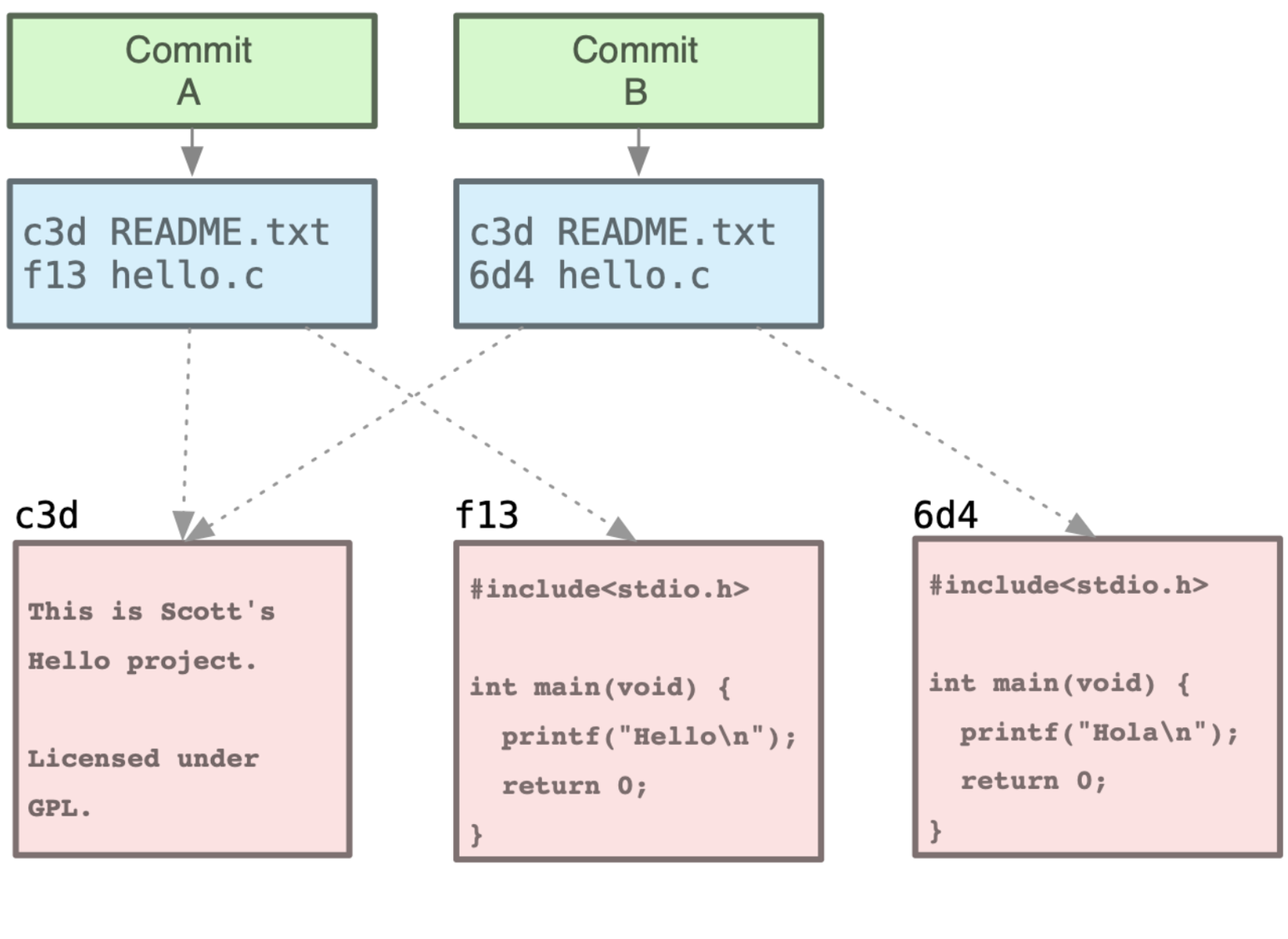
```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

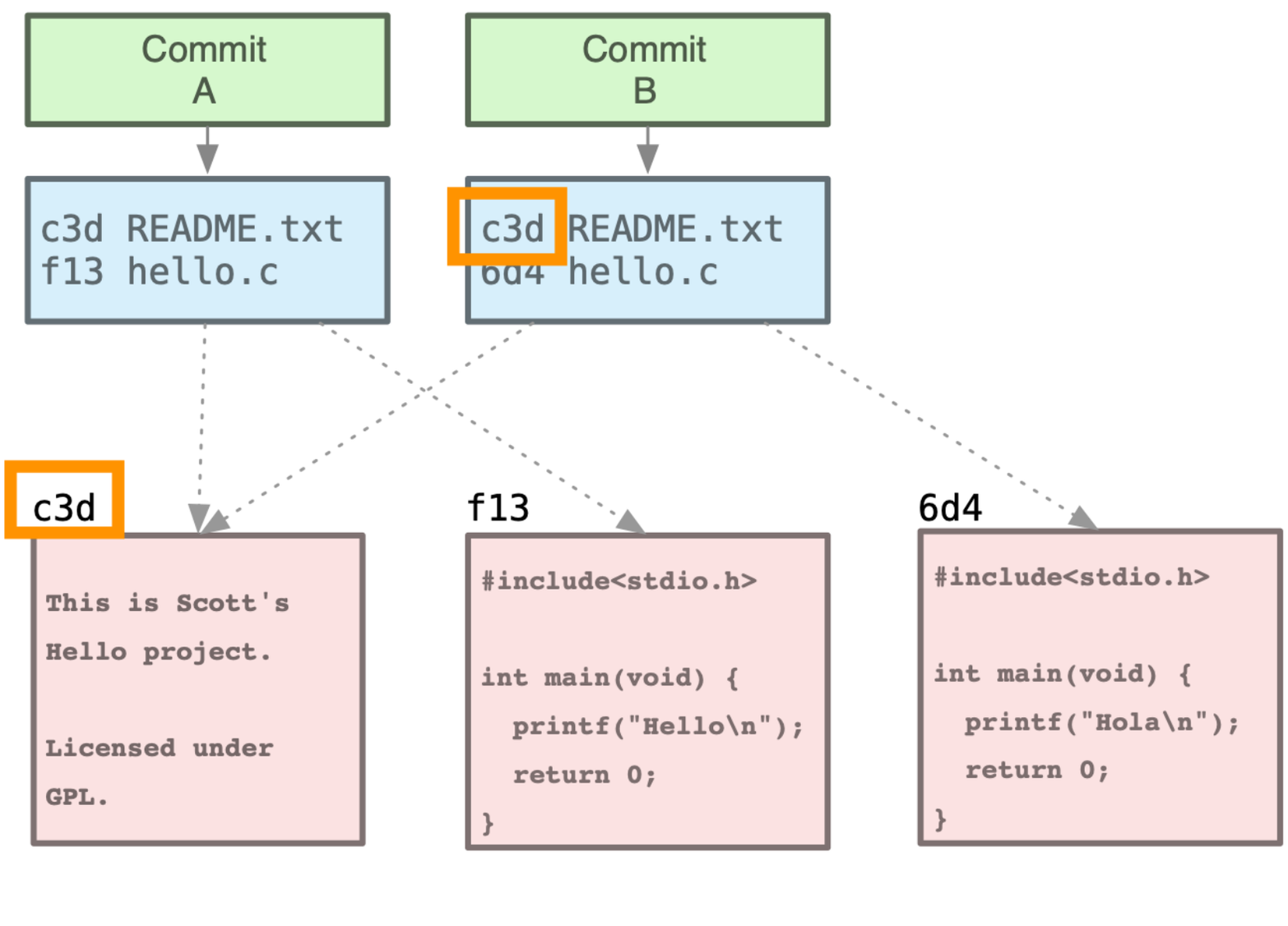


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

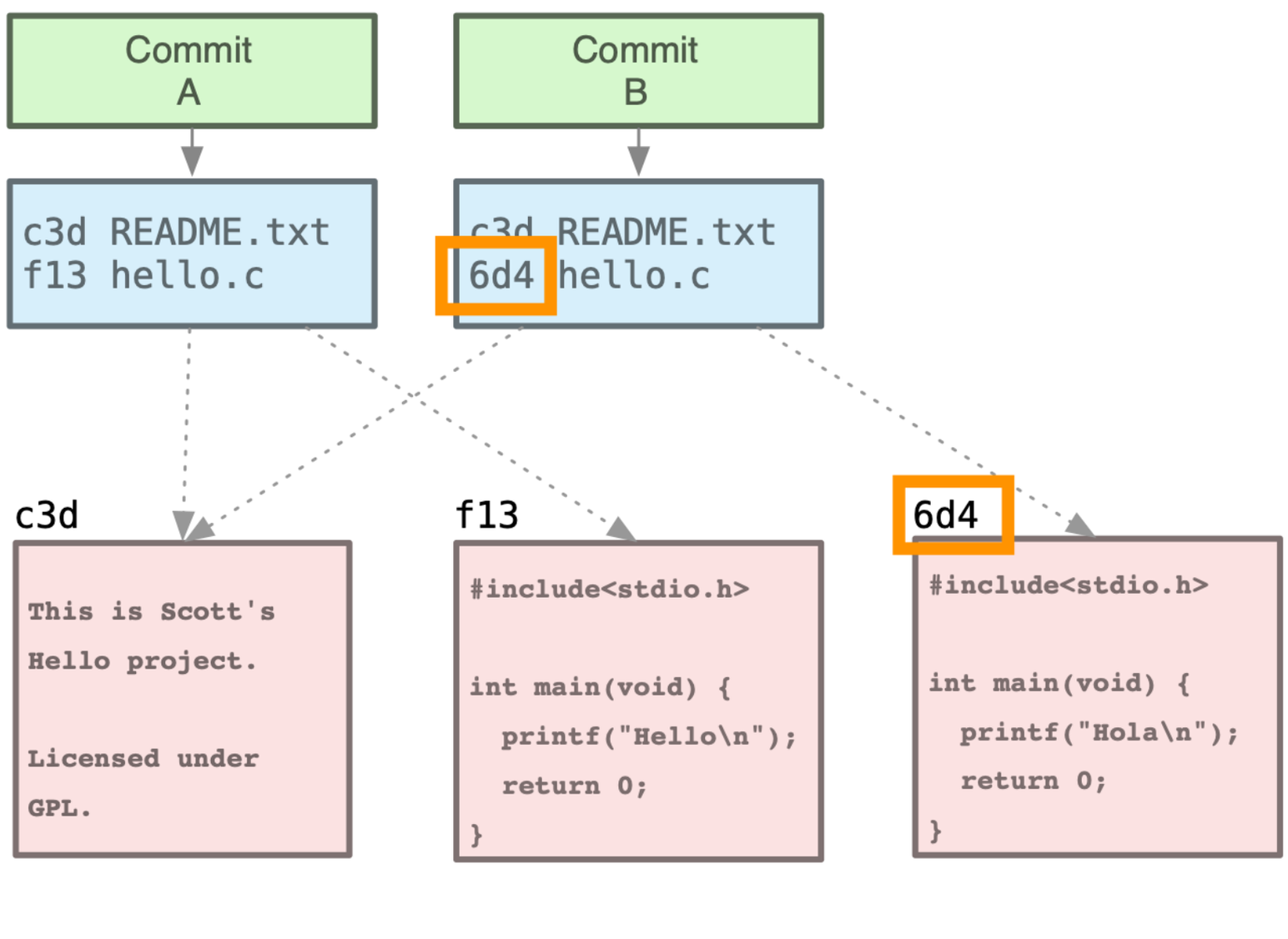


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

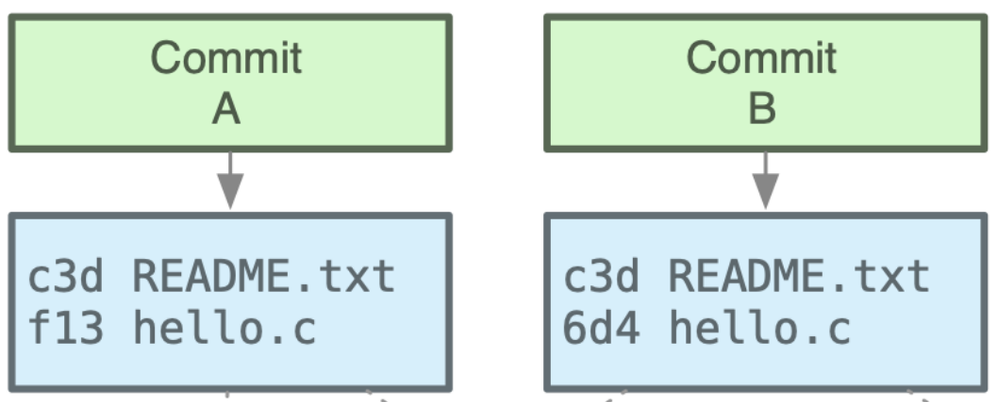


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



c3d

```
This is Scott's
Hello project.

Licensed under
GPL.
```

f13

```c
#include<stdio.h>

int main(void) {
  printf("Hello\n");
  return 0;
}
```

6d4

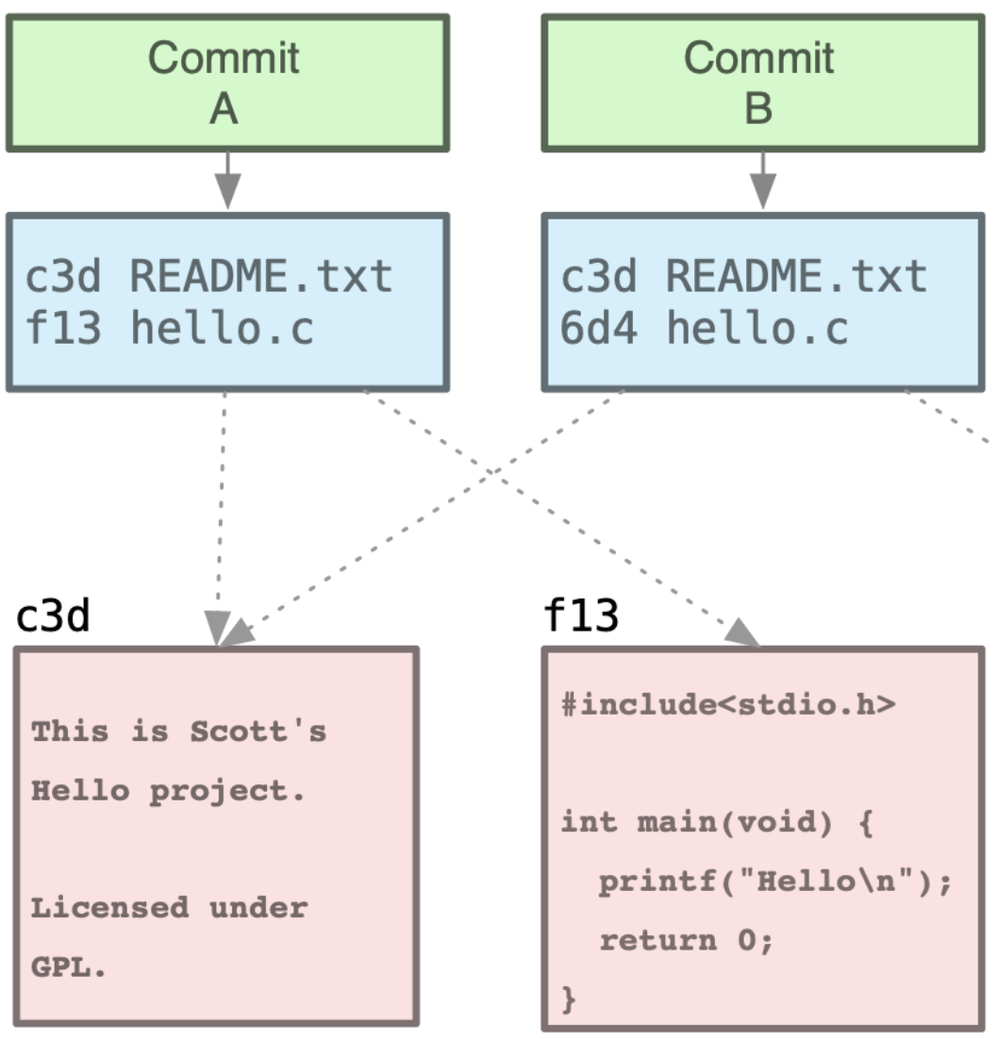
```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

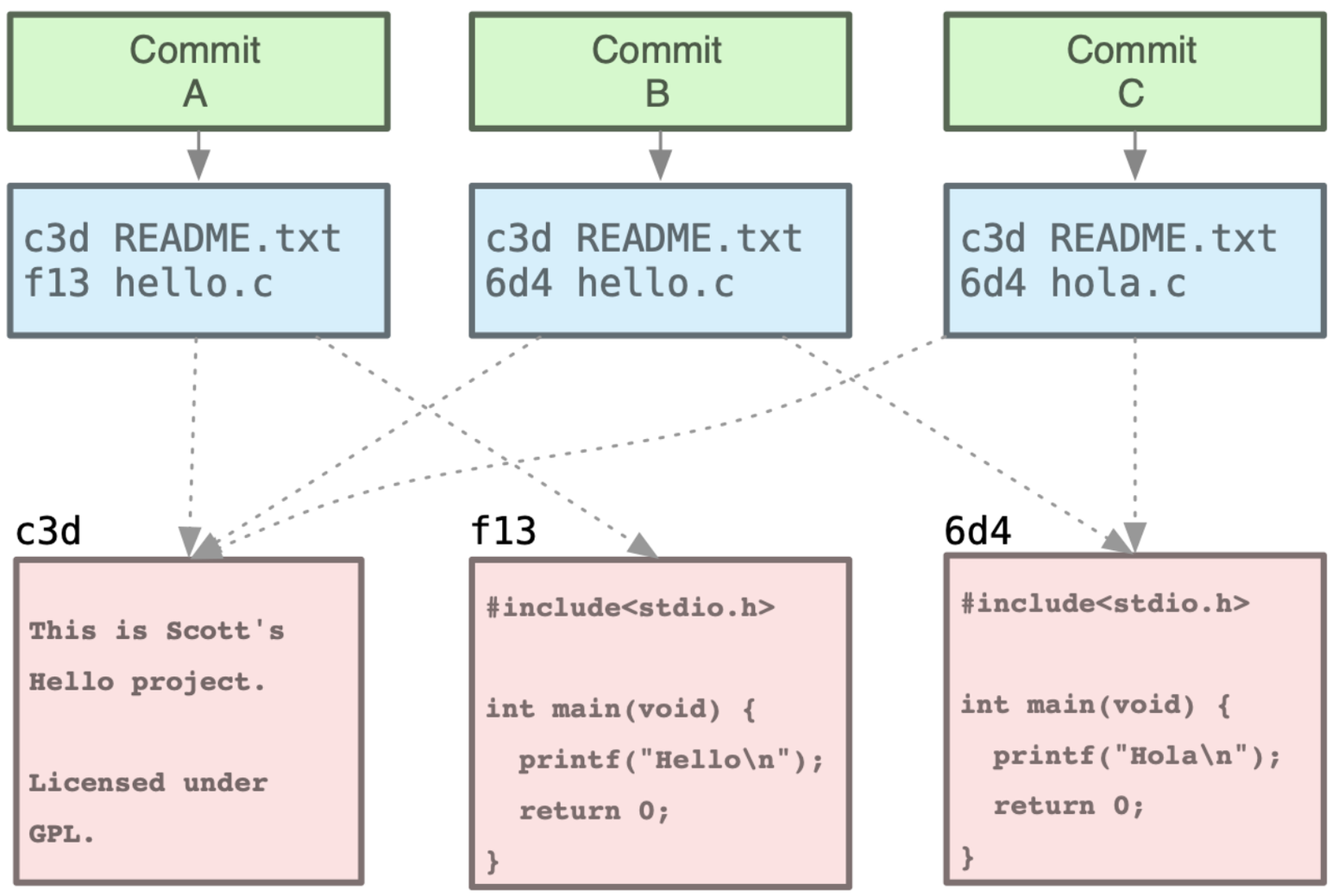


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

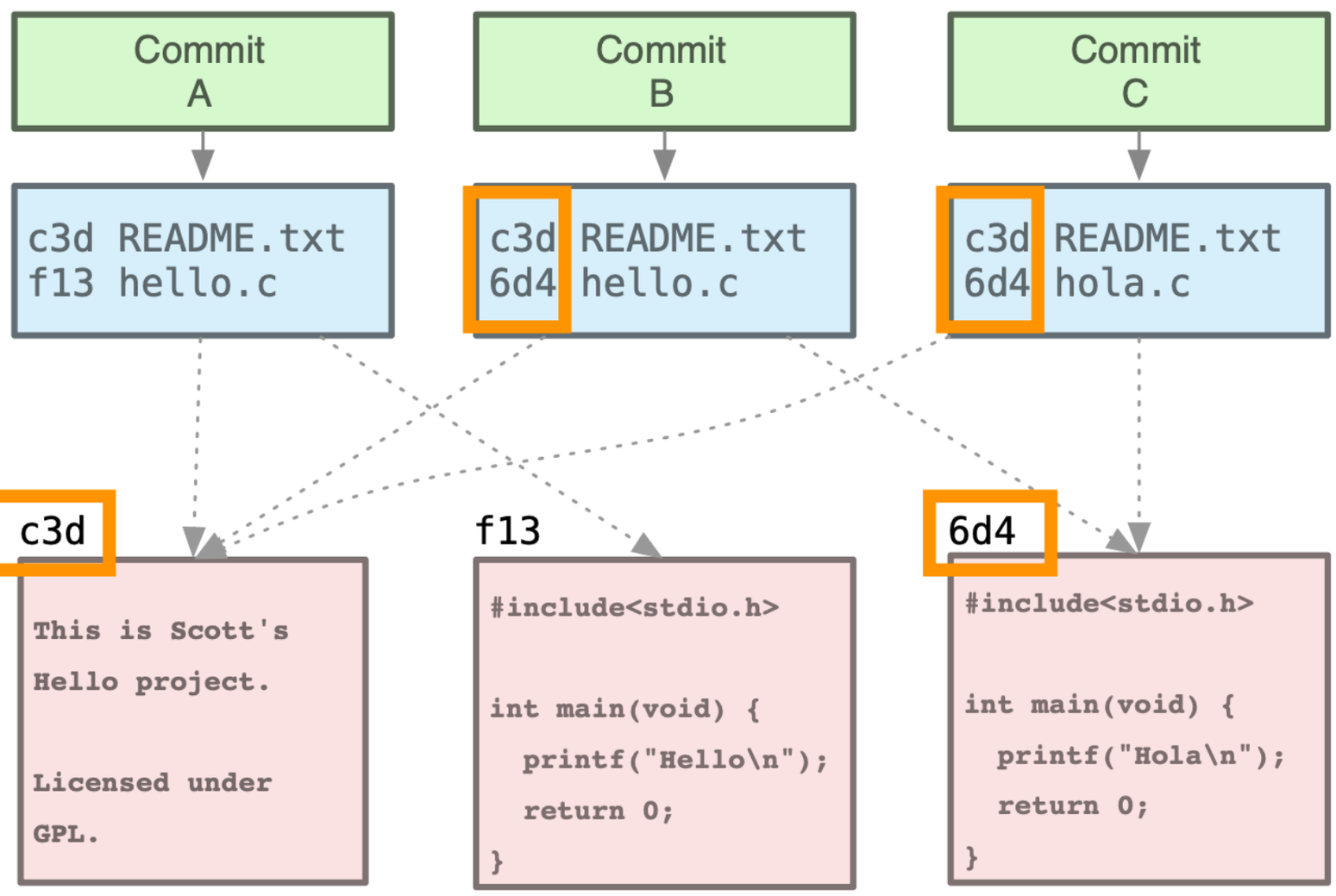


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

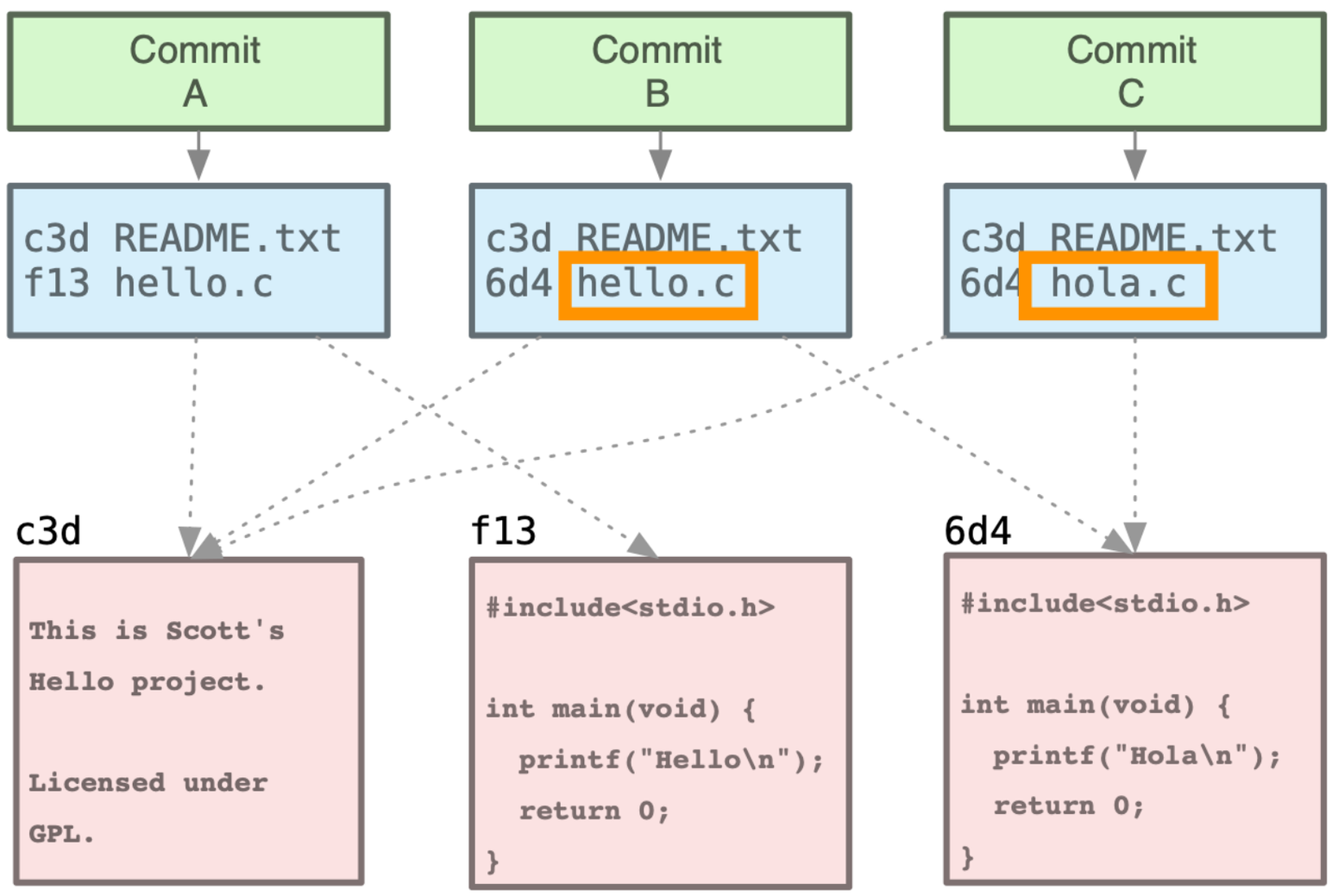


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

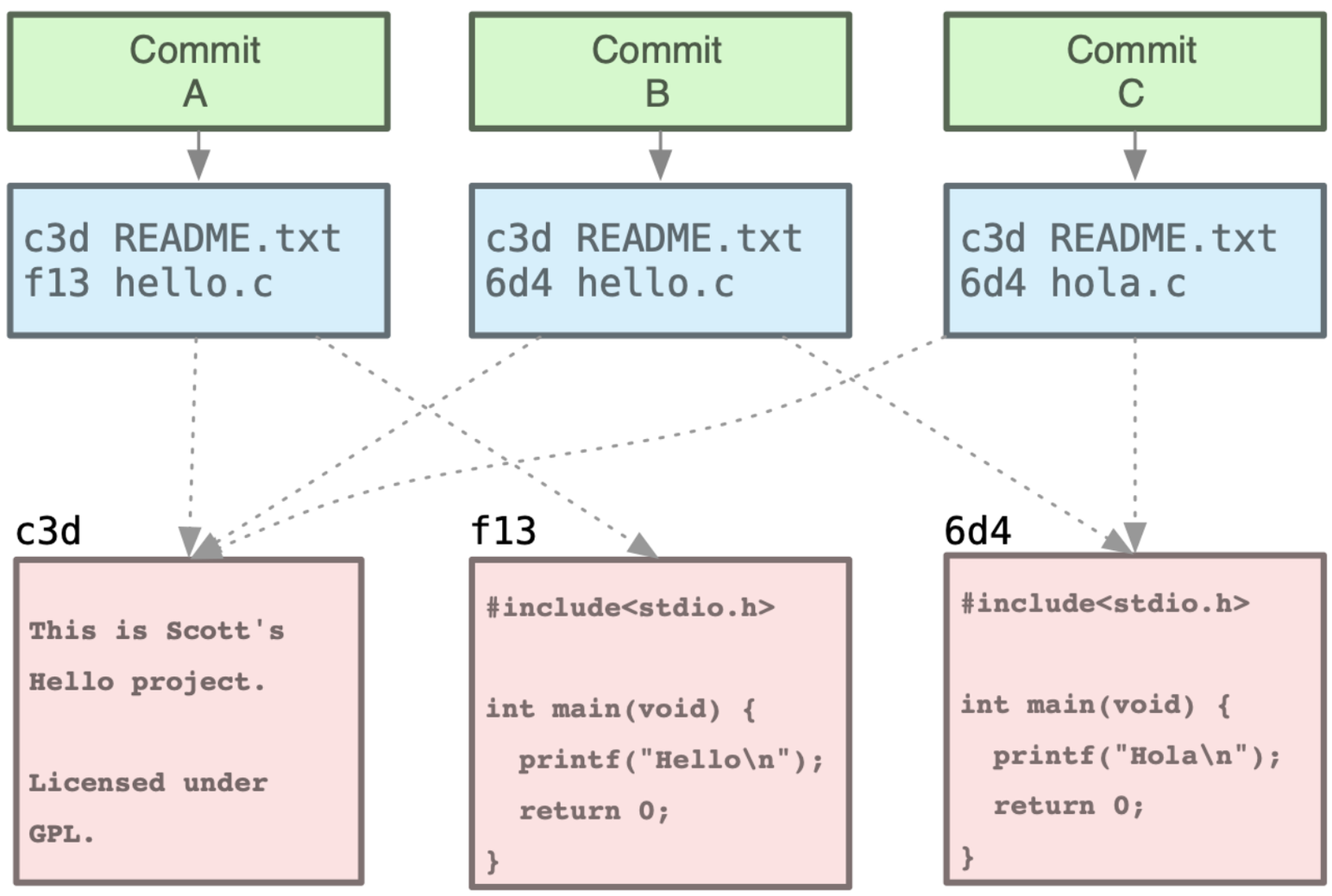


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

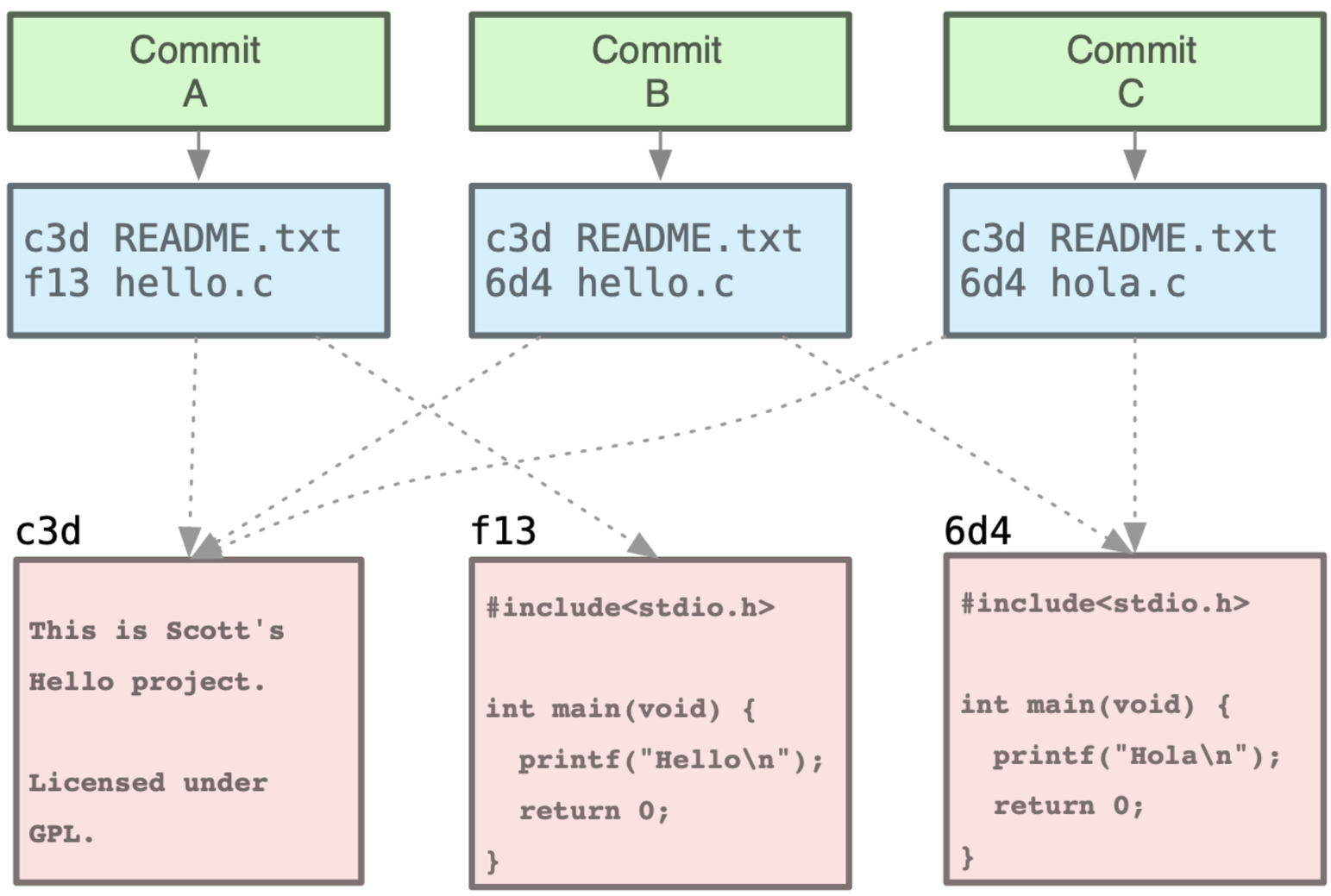


README.txt

```
This is Scott's
Hello project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

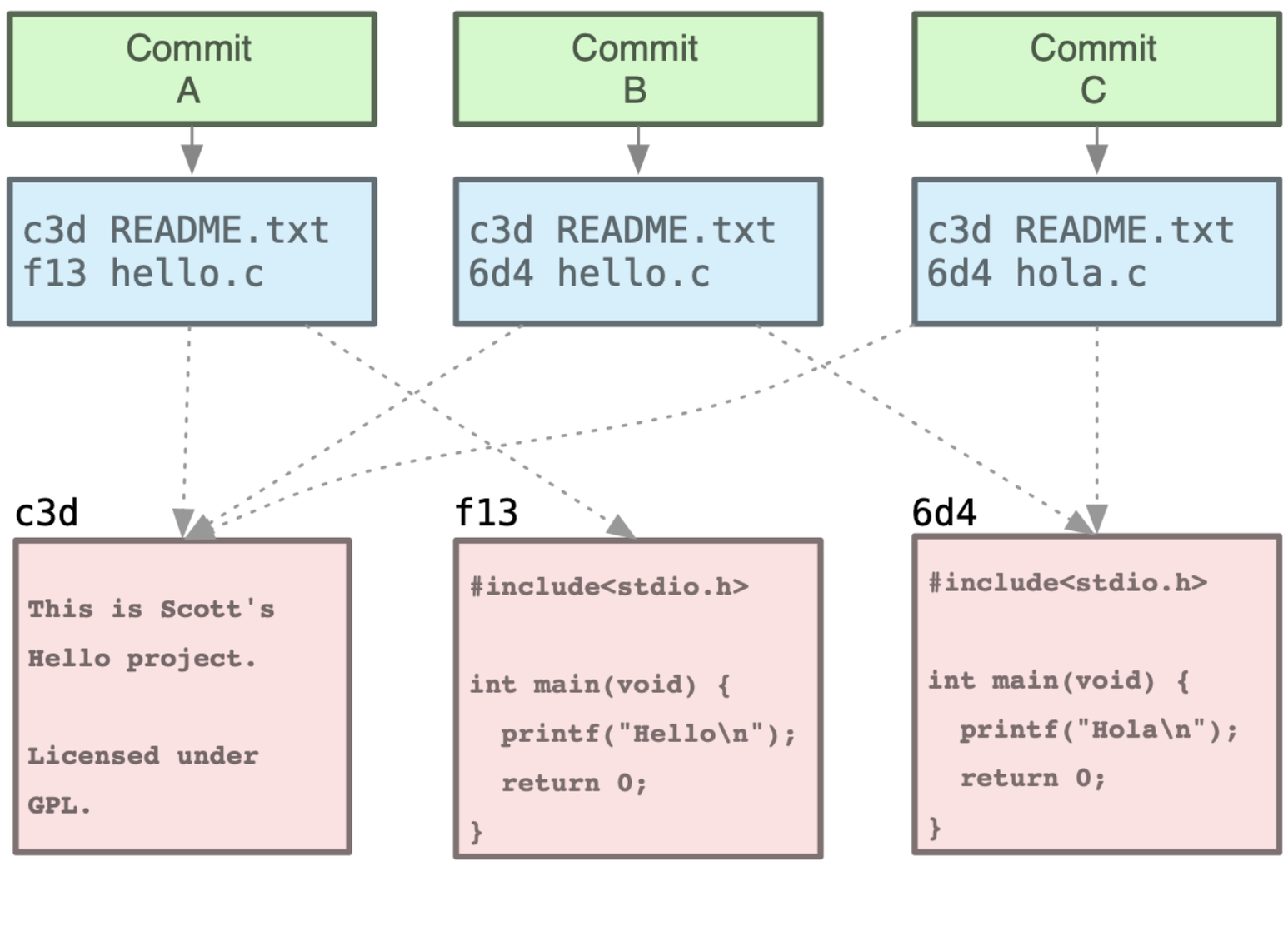


README.txt

```
This is Scott's
Hola project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

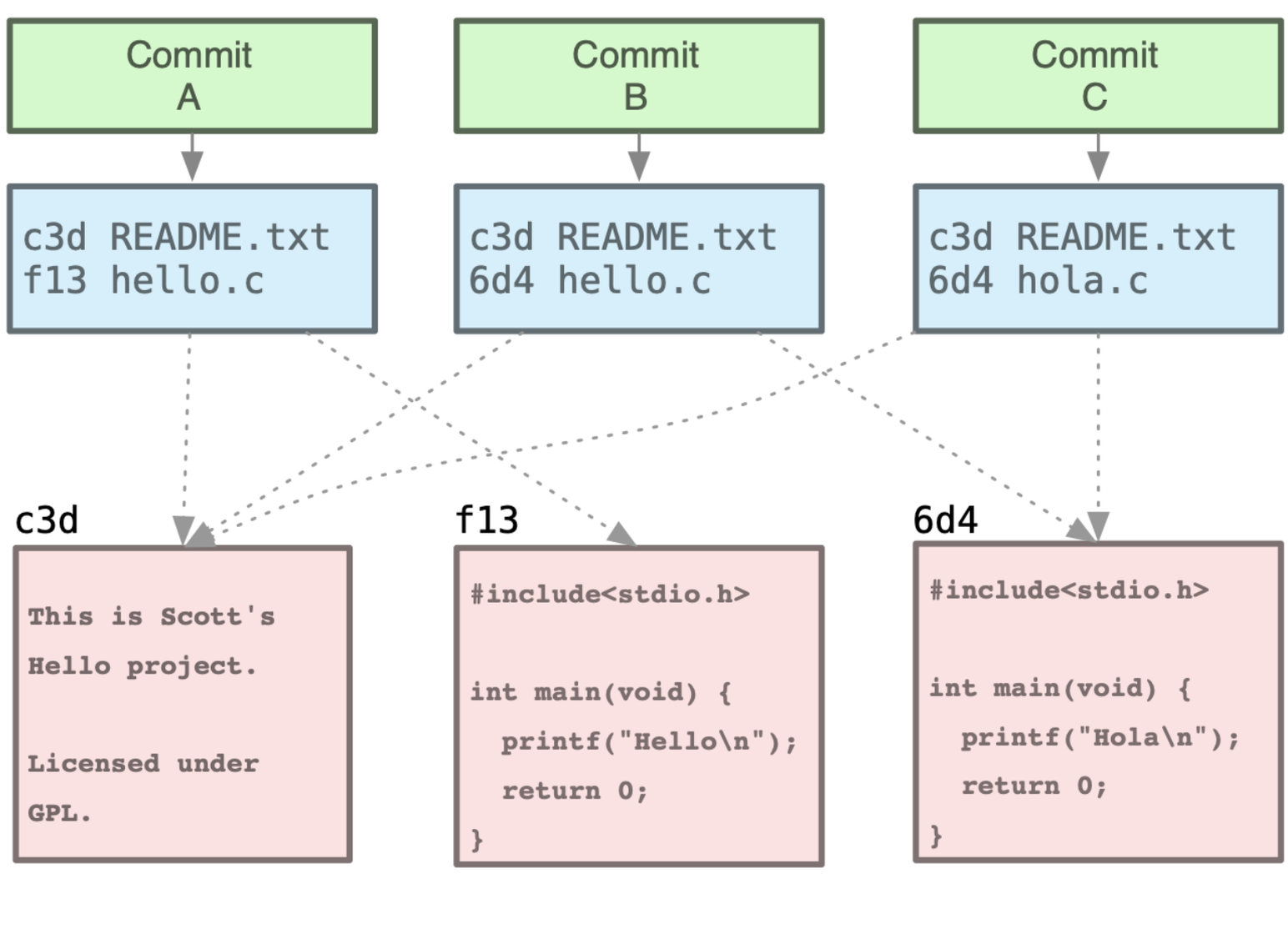
```
This is Scott's
Hola project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

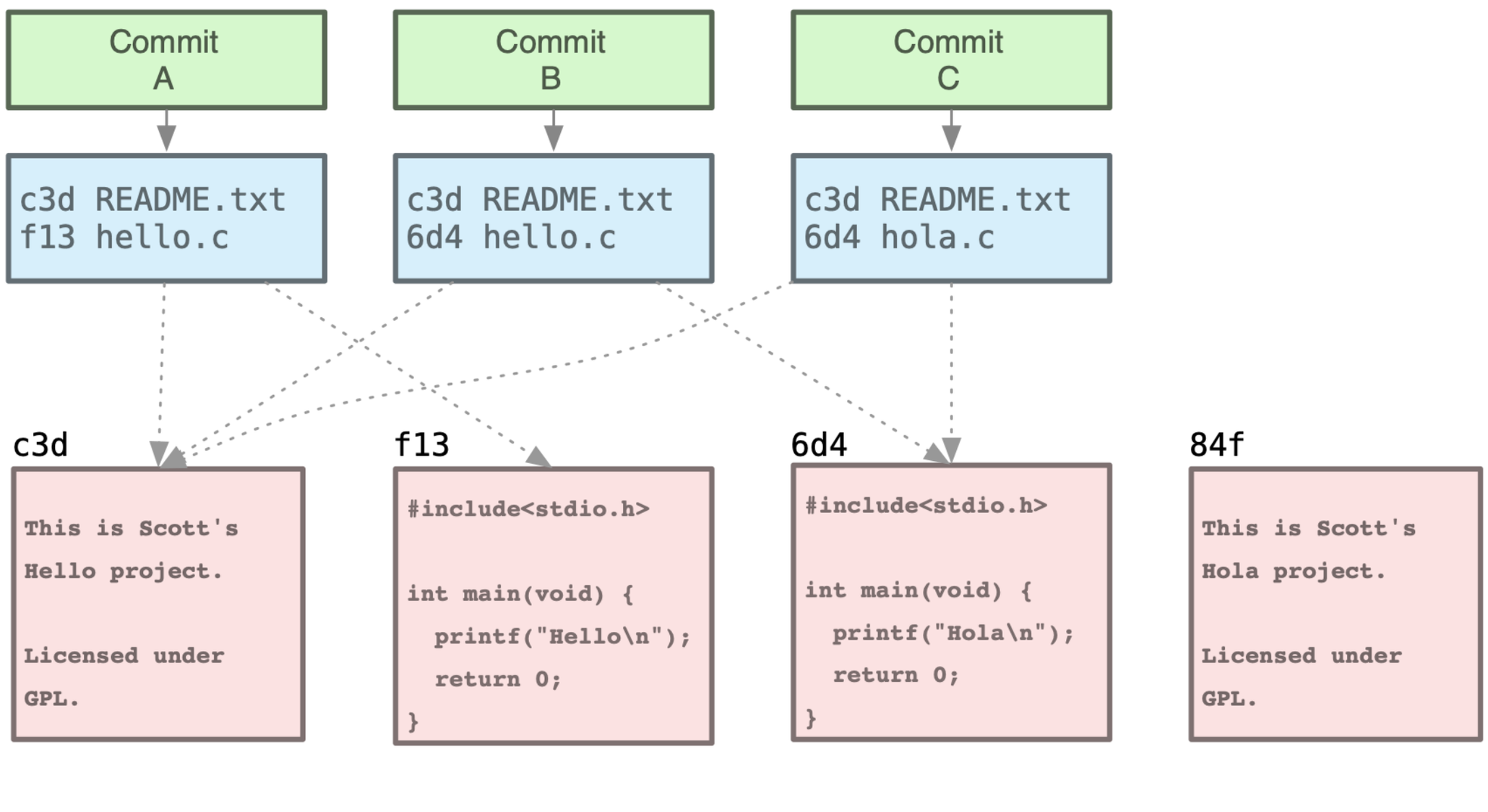
```
This is Scott's
Hola project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

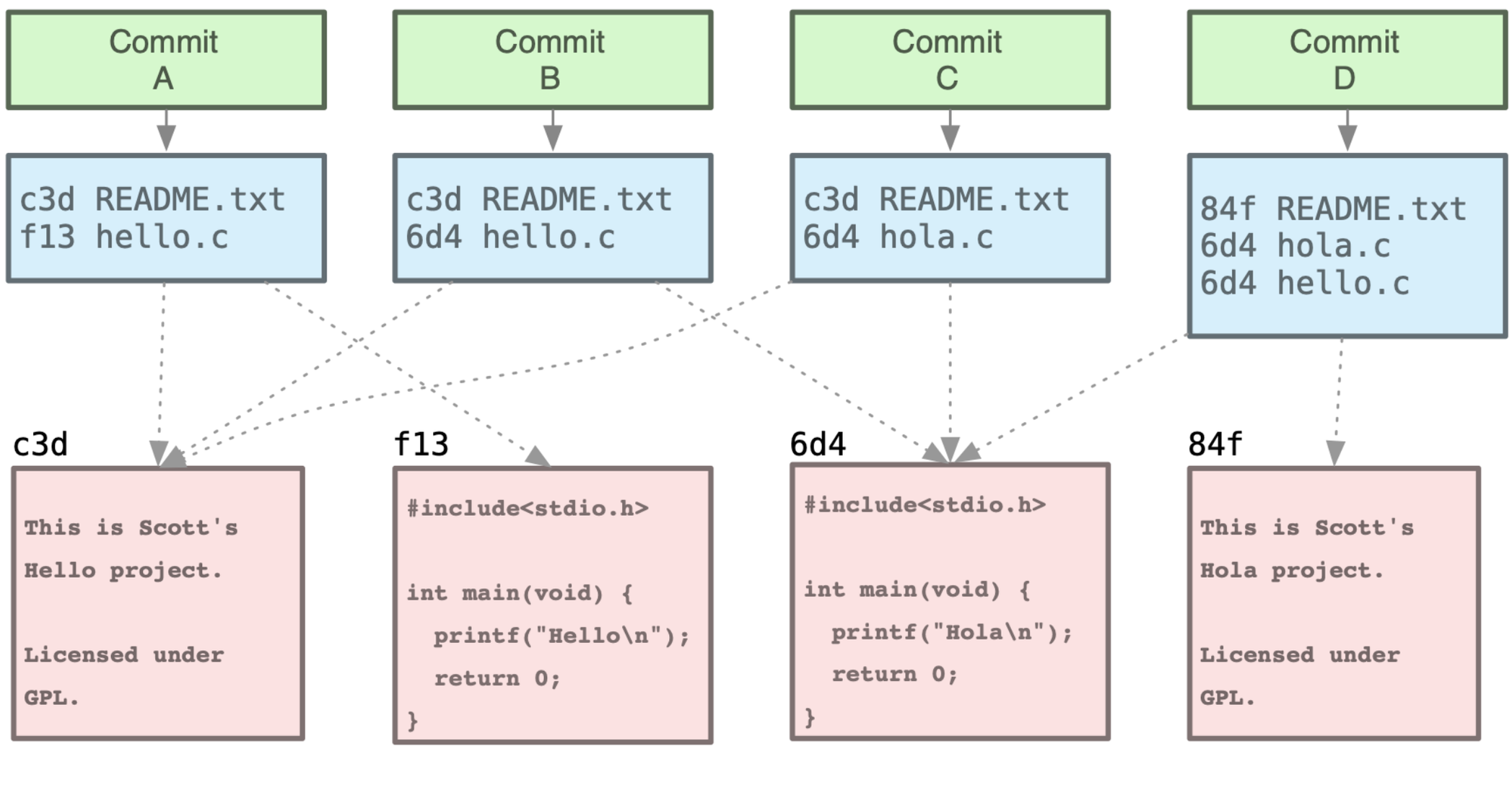
```
This is Scott's
Hola project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



README.txt

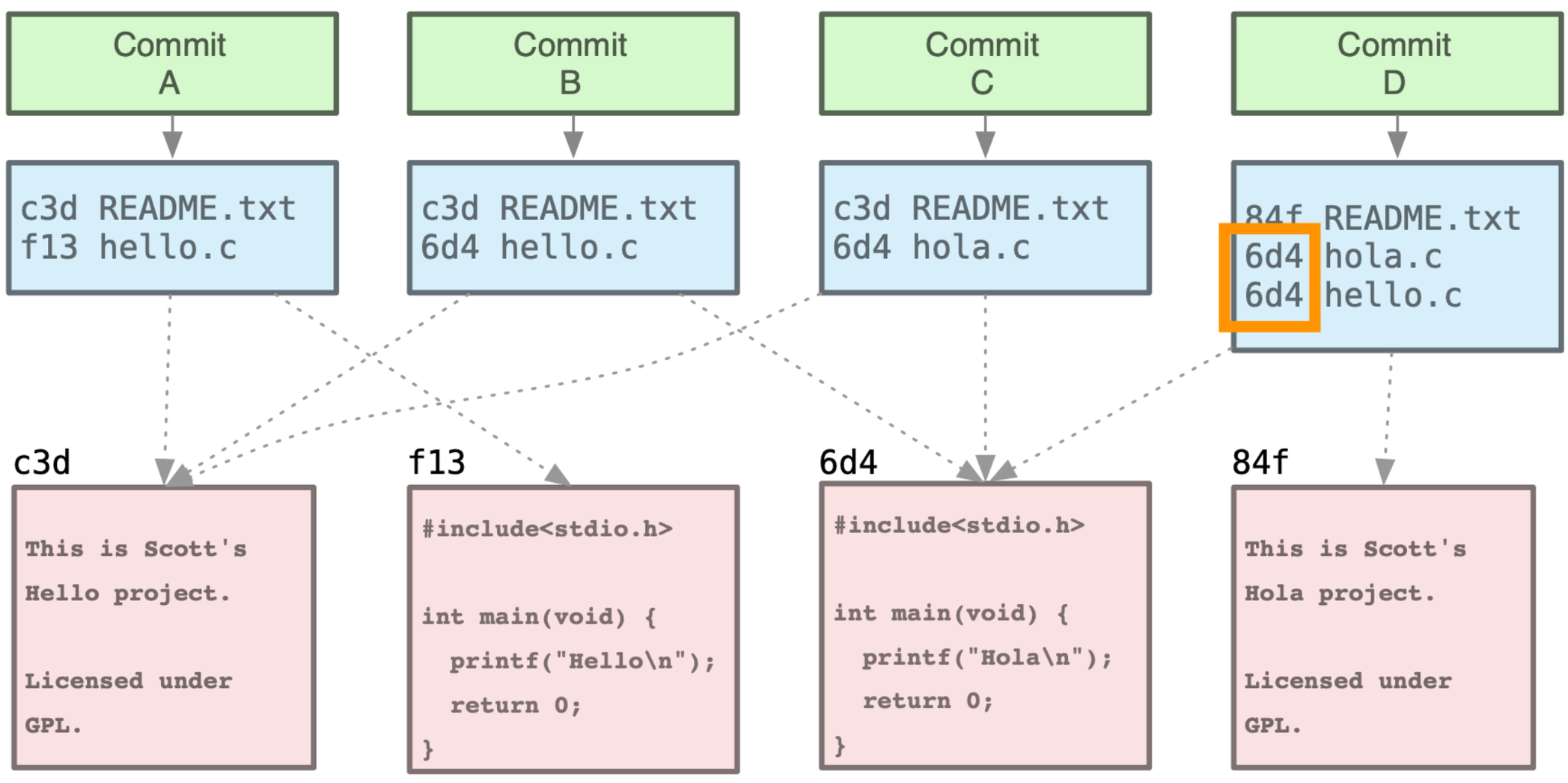
```
This is Scott's
Hola project.

Licensed under GPL.
```

hola.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```

hello.c

```c
#include<stdio.h>

int main(void) {
  printf("Hola\n");
  return 0;
}
```



COMANDI BASE

STRUTTURA COMANDI

```
git commit -a -m "Aggiunge il file reame.txt"
```

GIT

Comando

Opzioni

INIZIALIZZAZIONE

Setup iniziale

- Per prima cosa creiamo un repository

```
$ git init
```

- È necessario inizializzare la configurazione di Git
  - Dobbiamo identificare i nostri commit

```
$ git config --global user.name “Mario Rossi“
$ git config --global user.email “m.rossi@acme.inc.it”
$ git config --global color.ui true
```

- Le impostazioni **globali** sono memorizzate in ~/.gitconfig

INIZIALIZZAZIONE

- Inizializziamo la working directory
  - Per prima cosa creiamo un repository

```
$ git init
```

  - È necessario inizializzare la configurazione di Git
    - Dobbiamo identificare i nostri commit

```
$ git config --global user.name “Mario Rossi“
$ git config --global user.email “m.rossi@acme.inc.it”
$ git config --global color.ui true
```

  - Le impostazioni **globali** sono memorizzate in ~/.gitconfig

  - La lista delle impostazioni possibili è molto lunga
    - https://git-scm.com/docs/git-config

CONFIGURAZIONI – PRIORITÀ

- Git prevede vari tipi di configurazioni in ordine gerarchico
  1. System
  2. Global
  3. Local

- Le configurazioni vengono applicate in ordine gerarchico
  - In caso di definizione multipla, i valori sono sovrascritti

- Per interagire con le diverse configurazioni usiamo

```
$ git config --system <command>
```

```
$ git config --global <command>
```

```
$ git config --local <command>
```

CONFIGURAZIONI – COMANDI

- Per controllare i valori in una configurazione utilizziamo

```
$ git config <config> --list
```

- Per la lista complete dei parametri attualmente in uso
  - Applicati in ordine

```
$ git config --list
```

- Per aprire il file di configurazione in un editor

```
$ git config <config> -e
```

CONFIGURAZIONI – COMANDI

- Valore corrente di una variabile

```
$ git config --get <variable-name>
```

- Eliminare una variabile dalla configurazione

```
$ git config <config> --unset <variable-name>
```

- Cambiare valore a una variabile

```
$ git config <config> --replace-all <variable-name> <value>
```

STRUTTURA CARTELLA

Subito dopo il setup

```
.
├── .git
│   ├── HEAD
│   ├── branches
│   ├── config
│   ├── description
│   ├── hooks
│   │   ├── applypatch-msg.sample
│   │   ├── commit-msg.sample
│   │   ├── post-update.sample
│   │   ├── pre-applypatch.sample
│   │   ├── pre-commit.sample
│   │   ├── pre-push.sample
│   │   ├── pre-rebase.sample
│   │   ├── prepare-commit-msg.sample
│   │   └── update.sample
│   ├── info
│   │   └── exclude
│   ├── objects
│   │   ├── info
│   │   └── pack
│   └── refs
│       ├── heads
│       └── tags
└── hello_world.rb
```

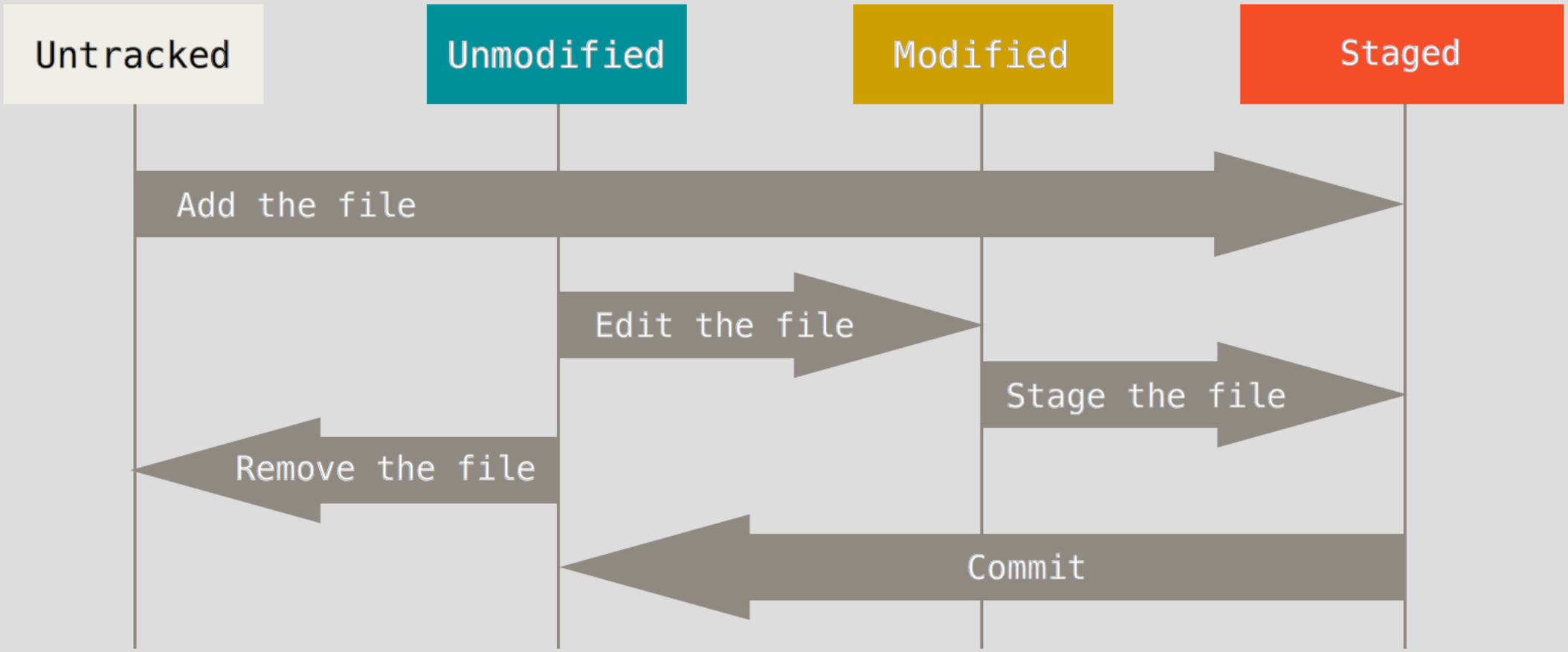
STATI DEI FILE

Ogni file in Git si trova in uno tra i seguenti stati:

- **untracked**: il file non è indicizzato nel repository, ma si trova nella working directory
- **unmodified**: il file è indicizzato e la working directory non mostra modifiche rispetto al repository
- **modified**: il file è indicizzato ma la versione nella working directory e quella nel repository sono differenti
- **staged**: il file è indicizzato, la versione nella working directory e quella ne repository sono differenti e la versione presente nella working directory è pronta per essere aggiunta al repository

VARIAZIONE DI STATO

Ogni file in Git si trova in uno tra i seguenti stati:



VARIAZIONE DI STATO

Il file hello_world.rb è in stato untracked, infatti:

```
$ git status
On branch master

Initial commit

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

  hello_world.rb

nothing added to commit but untracked files present (use "git add" to track)
```

VARIAZIONE DI STATO

Aggiungiamo il file hello_world.rb al repository:

```
$ git add hello_world.rb
```

Ora il file è nella staging area:

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to
unstage)

    new file:   hello_world.rb
```

VARIAZIONE DI STATO

Possiamo rimuovere hello_world.rb dalla staging area:

```
$ git reset hello_world.rb
```

Il file è nella staging area:

```
$ git status
On branch master

Initial commit

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

  hello_world.rb

nothing added to commit but untracked files present (use "git add" to track)
```

VARIAZIONE DI STATO

- Procediamo con il commit:

```
$ git commit -m "Aggiungi il file hello_world.rb"
[master (root-commit) d183b28] Aggiungi il file hello_world.rb
 1 file changed, 0 insertions(+), 0 deletions(-)
 create node 100644 hello_world.rb
```

- La working directory è allineata al repository!

```
$ git status
On branch master
nothing to commit, working directory clean
```

VARIAZIONE DI STATO

Subito dopo il commit

```
.
.
├── .git
│   ├── COMMIT_EDITMSG
│   ├── HEAD
[…]
│   ├── logs
│   │   ├── HEAD
│   │   └── refs
│   │       └── heads
│   │           └── master
│   ├── objects
│   │   ├── 32
│   │   │   └── 09658ac8d80bc9726d3a33d77e3dfc5fe6035e
│   │   ├── d1
│   │   │   └── 83b28caae0a7010dddb7273082c72ec6270012
│   │   ├── e6
│   │   │   └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
│   │   ├── info
│   │   └── pack
│   └── refs
│       ├── heads
│       │   └── master
│       └── tags
└── hello_world.rb
```

VARIAZIONE DI STATO

Modifichiamo la working copy:

```
$ echo "Un po' di testo" >> hello_world.rb
$ touch test.txt
```

Controlliamo lo stato:

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   hello_world.rb

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

  test.txt

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

VARIAZIONE DI STATO

Aggiorniamo la staging area:

```
$ git add .
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   hello_world.rb
    new file:   test.txt
```

Modifichiamo ancora un file prima del commit:

```
$ echo "Piccola modifica" >> hello_world.rb
```

VARIAZIONE DI STATO

Ora il file hello_world.rb esiste in tre versioni diverse:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  modified:   hello_world.rb
  new file:   test.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   hello_world.rb
```

Per rendere stabili le modifiche nella repository.

```
$ git commit -a -m “Aggiorna tutti i file”
[master 11f743d] Aggiorna tutti i file
 2 files changed, 2 insertions(+)
 create mode 100644 test.txt
```

GIT DIFF

- Il comando diff ci permette di controllare le differenze fra differenti versione del codice

```
$ git diff [--cached] [<commit-id>]
```

- Per le differenze fra working directory e staging area

```
$ git diff
```

- Per le differenze fra staging area e repository

```
$ git diff --cached
```

VARIAZIONE DI STATO

il comando log permette di vedere l'evoluzione del repository:

```
$ git log
commit 11f743dfe639b6dd4cb70f69e3a304b553cea4cd
Author: Marco Console <console@gmail.com>
Date:   Tue Apr 5 01:04:16 2016 +0200

    Aggiorna tutti i file

commit d183b28caae0a7010dddb7273082c72ec6270012
Author: Marco Console <console@gmail.com>
Date:   Tue Apr 5 00:45:47 2016 +0200

    Aggiungi il file hello_world.rm
```

VARIAZIONE DI STATO

Immaginiamo di voler recuperare la versione precedente di un file:

```
$ git checkout d183b2 hello_world.rb
$ cat hello_world.rb
```

Mmmmm…non era quello che volevamo fare:

```
$ git checkout -f HEAD
$ cat hello_world.rb
Un po’ di testo
Piccola modifica
```

STASH

STASHING

- La **`stash`** (riserva) rappresenta una ulteriore area in cui salvare le modifiche alla working copy

- Le modifiche spostate nella **`stash`** vengo eliminate dalla working directory e non verranno considerate per i commit

- Le modifiche **`stashed`** rimangono dormienti fino a quando non vengono esplicitamente riattivate
  - Ulteriori add e commit non influiscono sulla stash

- Alla riattivazione, le modifiche **`stashed`** vengono riapplicate allo stato attuale della working directory
  - Dobbiamo fare attenzione a possibili conflitti

STASHING – QUANDO?

- Quando le modifiche locali vengono aggiunte nella stash, la working directory si sincronizza all'ultimo commit.
  - Tutti i cambiamenti effettuati vengono inseriti nella stash e annullati

- Se la working directory è desincronizzata rispetto al repository, git potrebbe chiederci di fare **`stashing`**

- Se abbiamo fatto qualche errore nelle modifiche, possiamo annullare le modifiche senza perderle per sempre

- Attenzione: i cambiamenti nella **`stash`** vengono persi non possono sovrascrivere cambiamenti locali di cui non è stato fatto il commit!

STASHING – COMANDI

- Aggiungere i cambiamenti locali nella **`stash`**

```
$ git stash [push] [-m <message>]
```

- Vedere le entry della **`stash`**

```
$ git stash list
```

- Informazioni sulle singole entry

```
$ git stash show -p [<stash-id>]
```

- Applicare i cambiamenti nella stash

```
$ git stash apply [<stash-id>]
```

STASHING – COMANDI

- Applicare ed eliminare l'ultima modifica **`stashed`**

```
$ git stash pop
```

- Eliminare una modifica **`stashed`**

```
$ git stash drop [<stash-id>]
```

- Eliminare tutte le modifiche stashed

```
$ git stash clear
```

CAT E HASH

CAT-FILE

Non possiamo accedere agli oggetti git direttamente

```
$ cat .git/objects/xx/yyyyy...
```

Per farlo possiamo utilizzare git cat-file

```
$ git cat-file -t <hash>

$ git cat-file -p  <hash>
```

CAT-FILE

- Potremmo aver bisogno di sapere l’hash di un file

```
$ git hash-object <path>
```

- Se il file è stato aggiunto alla repository, troveremo lo stesso hash in **`.git/objects`**

COMMIT

COMMIT

Ora cerchiamo di capire come funziona internamente Git:

- Cosa succede quando invochiamo “commit”?
- Dove vengono salvati i file?
- Quali metadati sono aggiunti?
- Come è possibile ricostruire la “storia” di un file?

GIT OBJECTS

- Git utilizza (principalmente) tre tipi di oggetti:
  - **Commit**
    - Contengono informazioni relative ai commit.
  - **Tree**
    - Contengono informazioni relative alla struttura delle directory versionate
  - **BLOB**
    - Contengono i dati relativi ai file versionati (binari, testo …)

- **Ogni oggetto** è salvato in un file.
  - Nella cartella ./.git/objects
- **Ogni oggetto** è identificato dal suo checksum.
  - Checksum calcolato con SHA-256
- https://git-scm.com/book/it/v2/Git-Internals-Git-Objects

OGGETTO COMMIT

- Rappresenta un commit.
- Campi:
  - **Parent**: il riferimento (checksum) al commit precedente.
  - **Tree**: il riferimento (checksum) alla directory versionata.
  - **Author**: Autore della modifica.
  - **Committer**: Autore del commit (non necessariamente uguale a Author)
  - **Message**: commento allegato al commit.

```
$ git commit
Created commit 77d3001: descriptive commit message
 2 files changed, 4 insertions(+), 2 deletions(-)
```

```
$ git commit
Created commit 77d3001: descriptive commit message
 2 files changed, 4 insertions(+), 2 deletions(-)
```

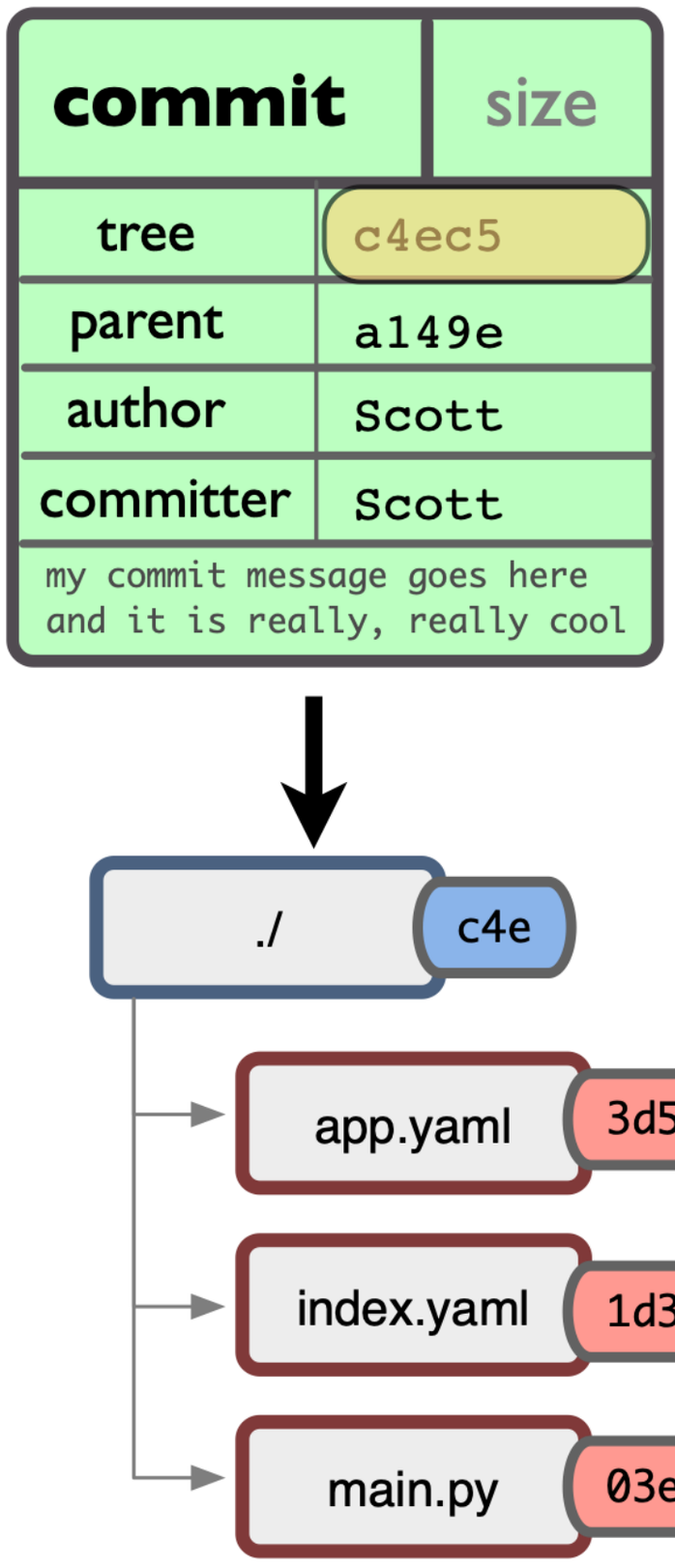
77d3001a1de6bf8f5e431972fe4d25b01e595c0b

77d3001a1de6bf8f5e431972fe4d25b01e595c0b

77d3001a1de6bf8f5e431972fe4d25b01e595c0b

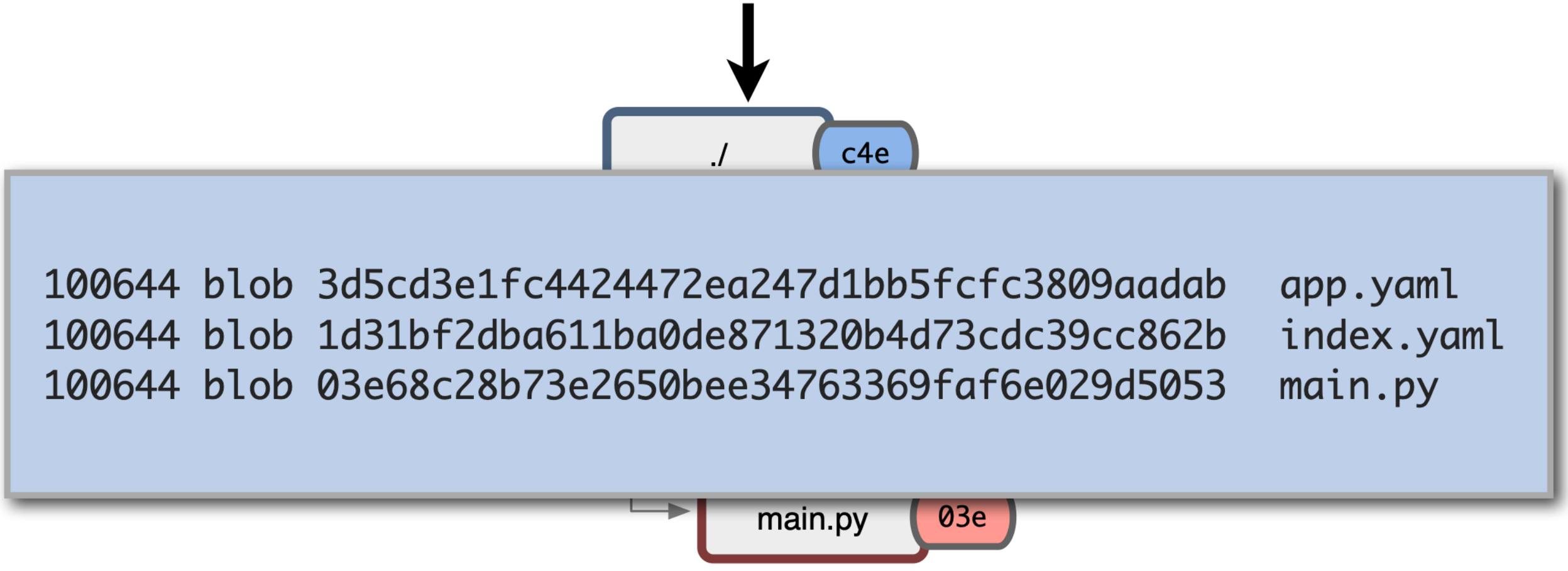
```
tree c4ec543b0322744e55c5efc9b6c4e449d398dbff
parent a149e2160b3f7573768cdc2fce24d0881f3577e1
author Scott Chacon <schacon@gmail.com> 1223402504 -0700
committer Scott Chacon <schacon@gmail.com> 1223402504 -0700

descriptive commit message
```

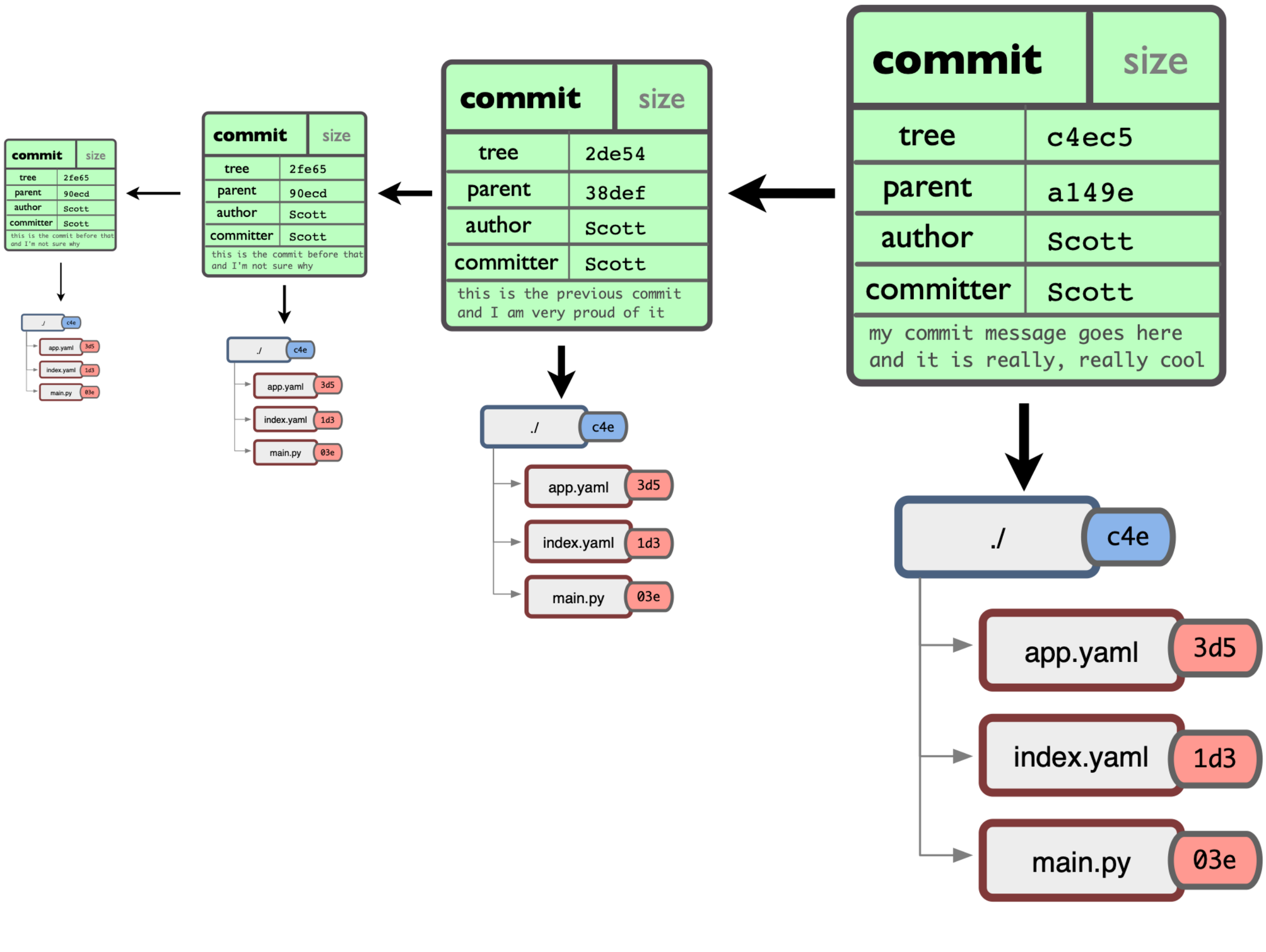
77d3001a1de6bf8f5e431972fe4d25b01e595c0b

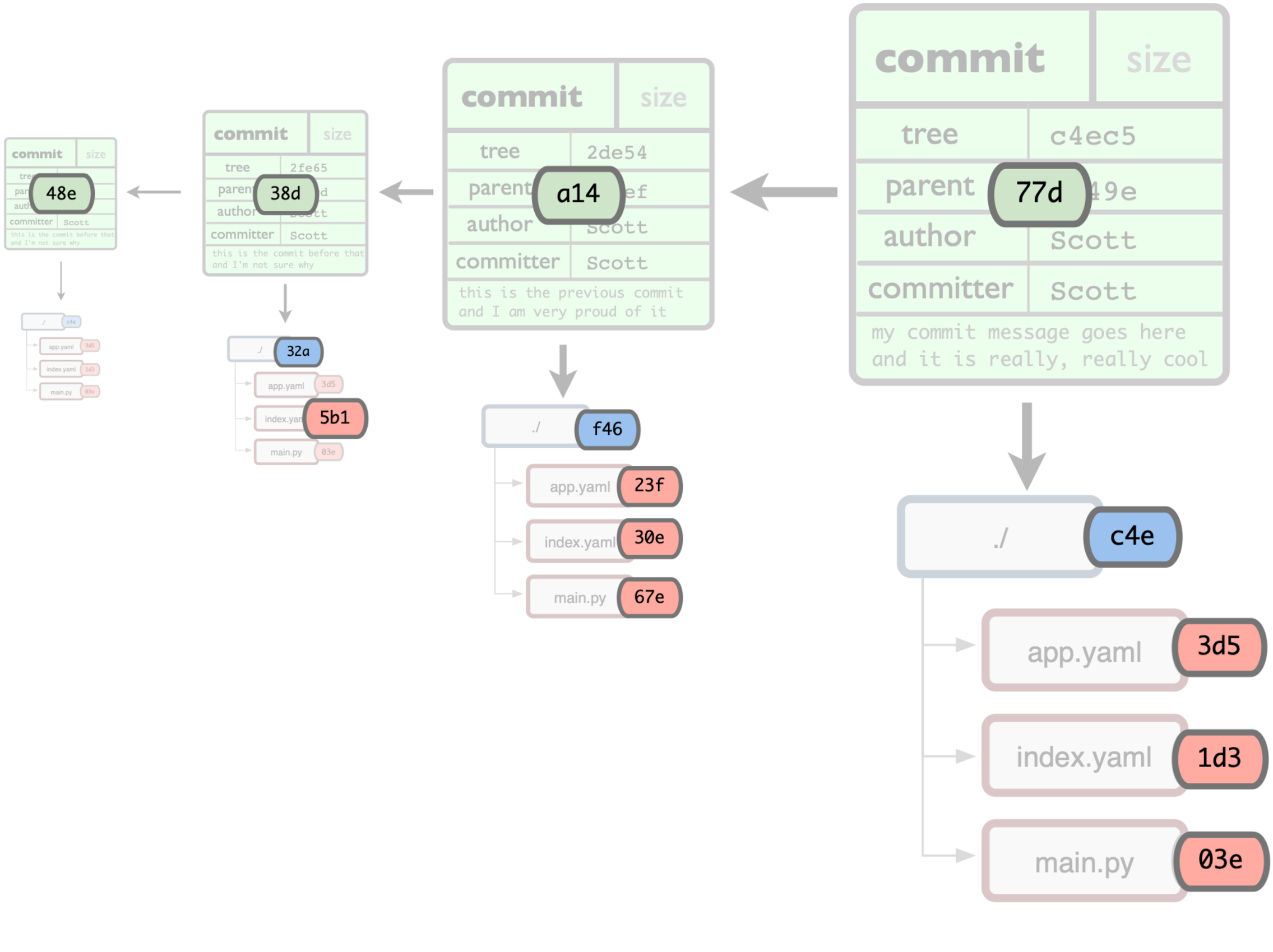


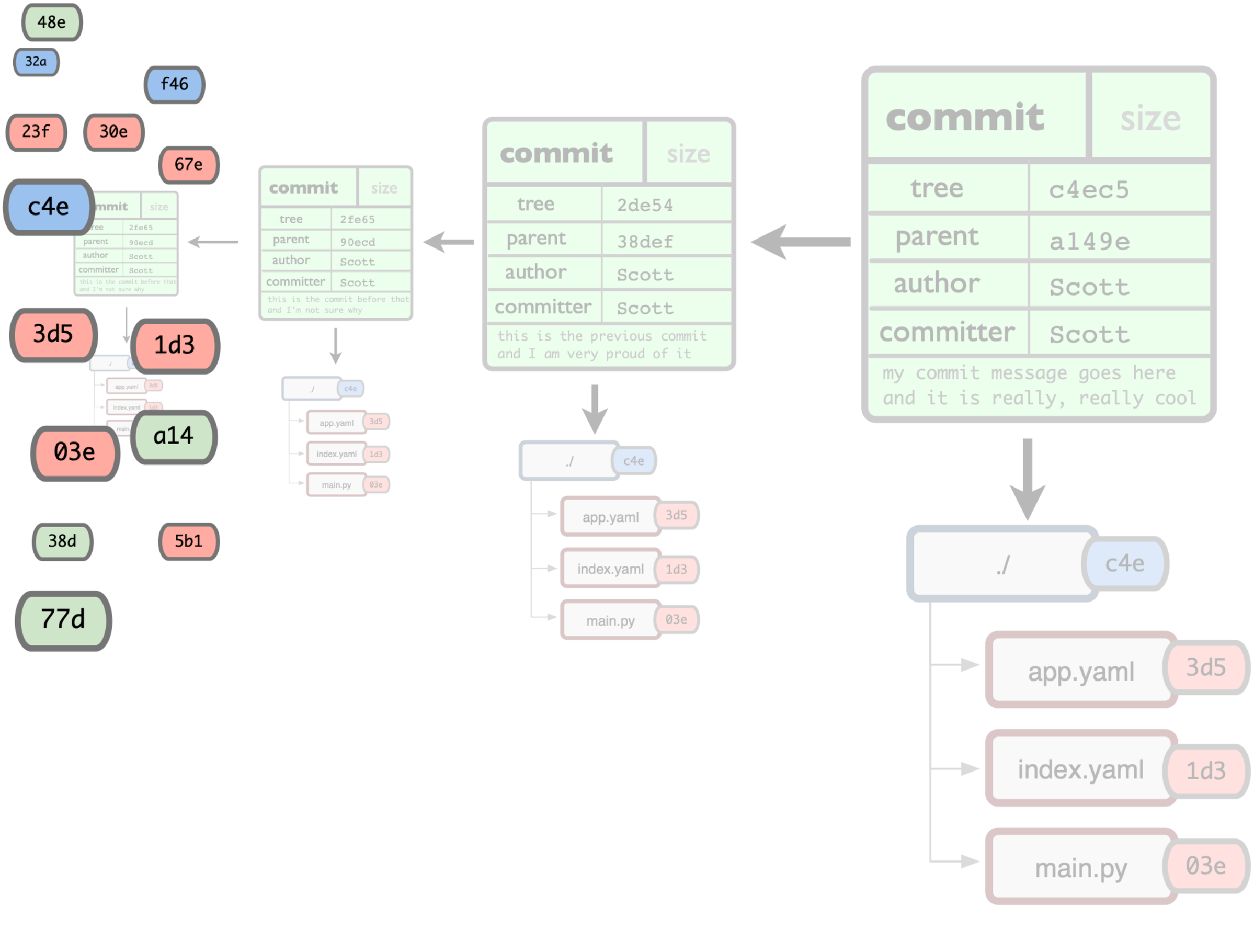
77d3001a1de6bf8f5e431972fe4d25b01e595c0b

commit	size
tree	c4ec5
parent	a149e
author	Scott
committer	Scott
my commit message goes here and it is really, really cool	




commit
size
tree
2fe65
parent
90ecd
author
Scott
committer
Scott
this is the commit before that and I'm not sure why
./
c4e
app.yaml
3d5
index.yaml
1d3
main.py
03e
commit
size
tree
2de54
parent
38def
author
Scott
committer
Scott
this is the previous commit and I am very proud of it
./
c4e
app.yaml
3d5
index.yaml
1d3
main.py
03e
commit
size
tree
c4ec5
parent
a149e
author
Scott
committer
Scott
my commit message goes here and it is really, really cool
./
c4e
app.yaml
3d5
index.yaml
1d3
main.py
03e

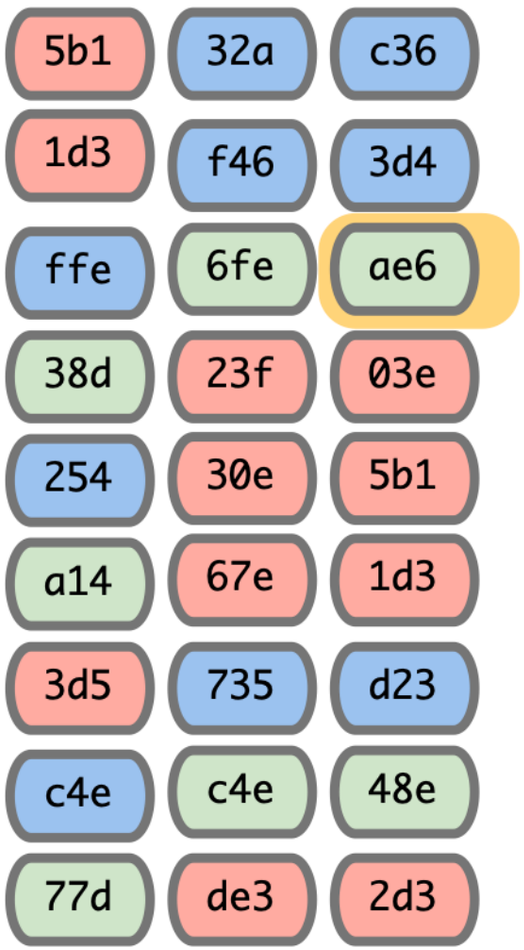

commit
size
tree
c4ec5
parent
77d
49e
author
Scott
committer
Scott
my commit message goes here
and it is really, really cool
./
c4e
app.yaml
3d5
index.yaml
1d3
main.py
03e
commit
size
tree
2de54
parent
a14
ef
author
Scott
committer
Scott
this is the previous commit
and I am very proud of it
./
f46
app.yaml
23f
index.yaml
30e
main.py
67e
commit
size
tree
2fe65
parent
38d
d
author
committer
Scott
this is the commit before that
and I'm not sure why
./
32a
app.yaml
3d5
index.yaml
5b1
main.py
03e
commit
size
tree
48e
committer
Scott
this is the commit before that
and I'm not sure why
./
c4e
app.yaml
3d5
index.yaml
1d3
main.py
03e


48e
32a
f46
23f
30e
67e
c4e
commit
size
tree
2fe65
parent
90ecd
author
Scott
committer
Scott
this is the commit before that and I'm not sure why
3d5
1d3
./
app.yaml
3d5
index.yaml
1d3
main.py
03e
a14
03e
38d
5b1
77d
commit
size
tree
2de54
parent
38def
author
Scott
committer
Scott
this is the previous commit and I am very proud of it
./
c4e
app.yaml
3d5
index.yaml
1d3
main.py
03e
commit
size
tree
c4ec5
parent
a149e
author
Scott
committer
Scott
my commit message goes here and it is really, really cool
./
c4e
app.yaml
3d5
index.yaml
1d3
main.py
03e

Repository

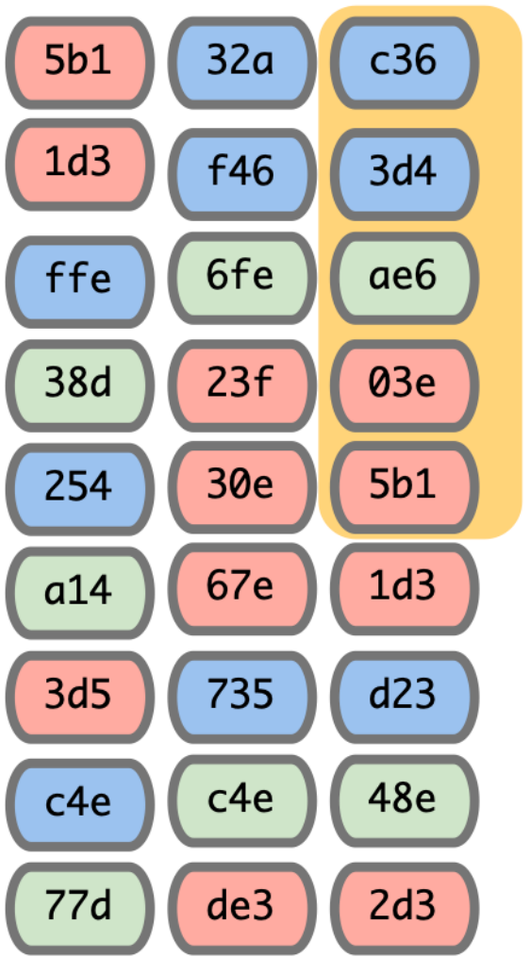

5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3

Repository

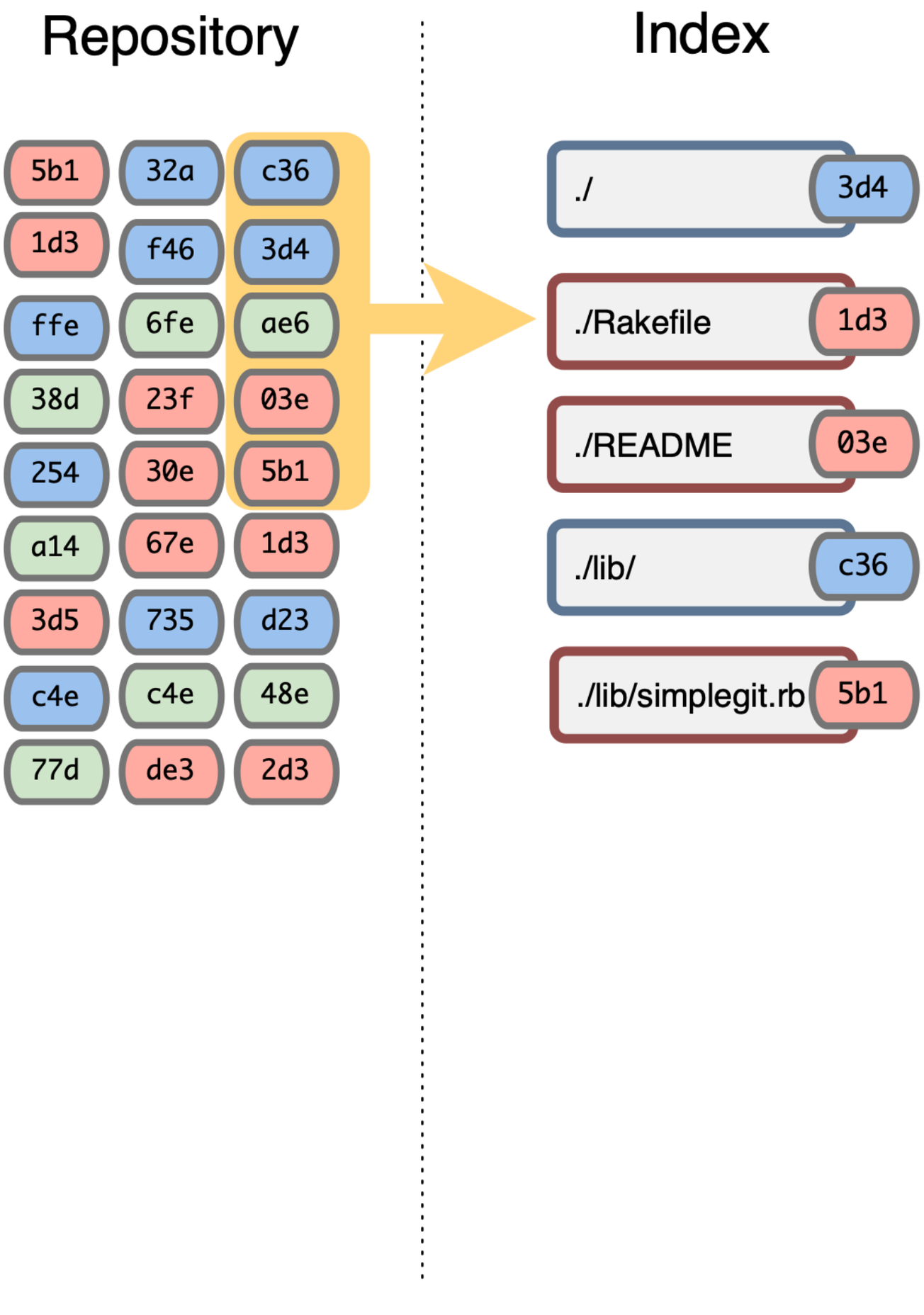


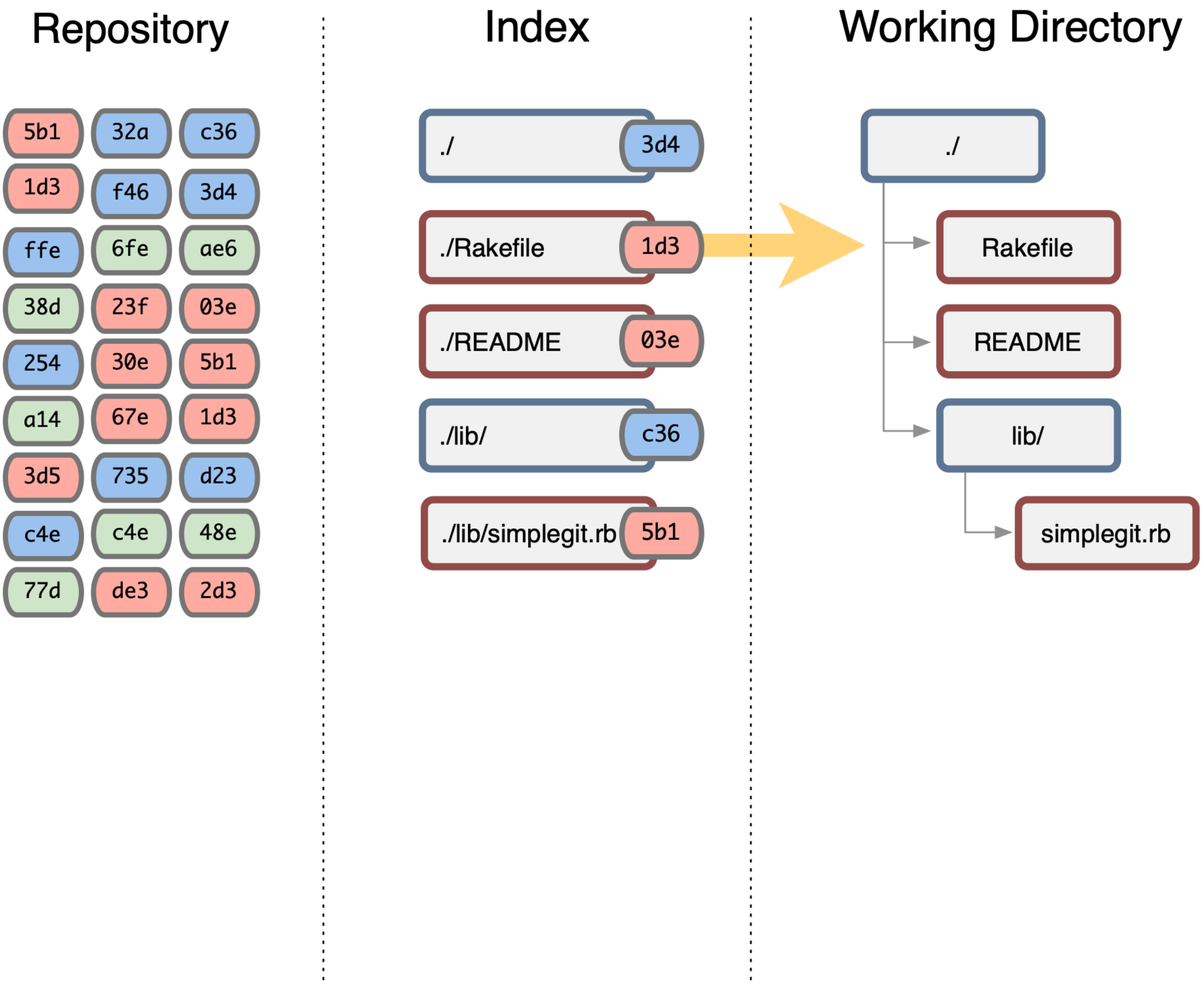
git checkout branch

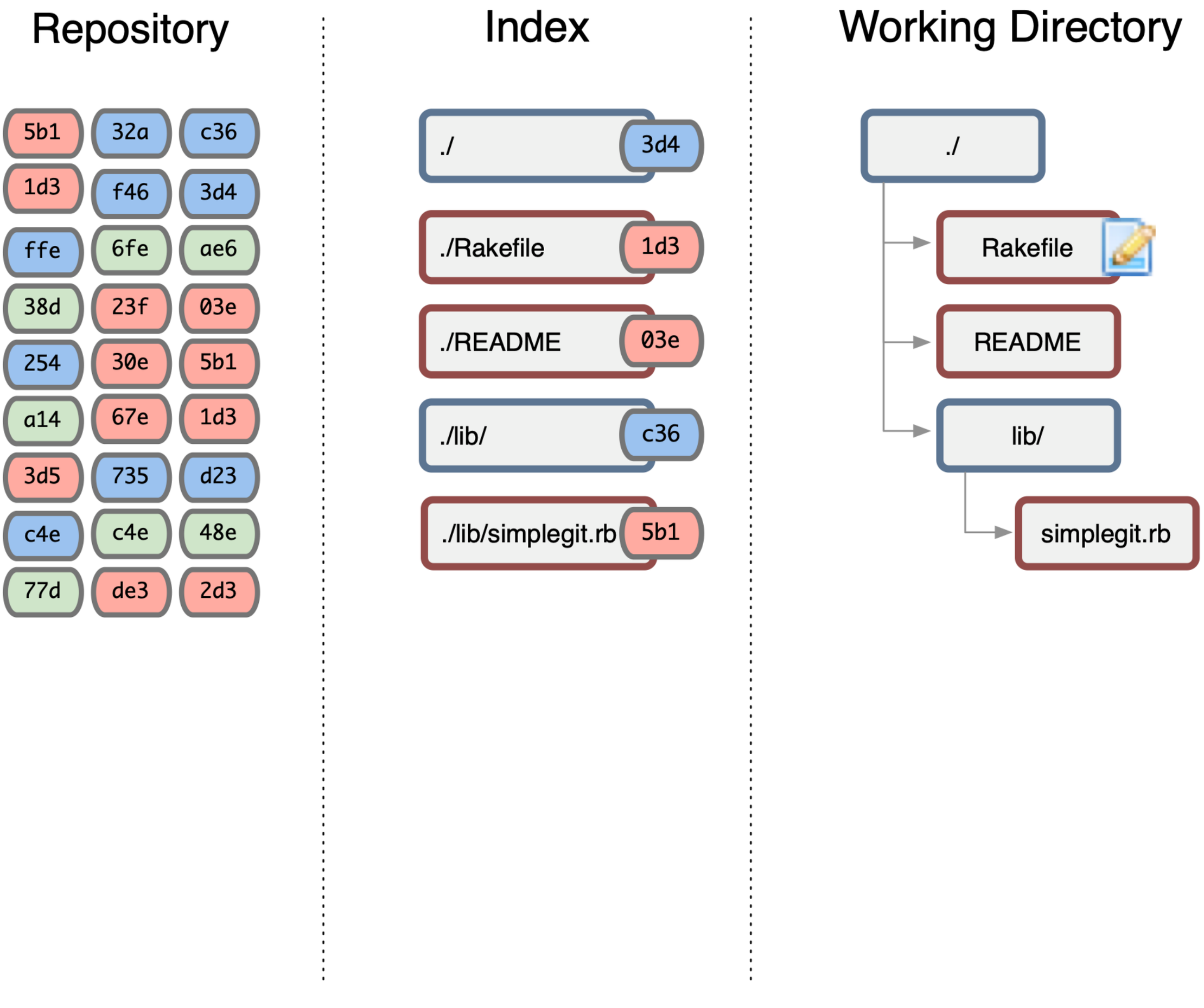
Repository

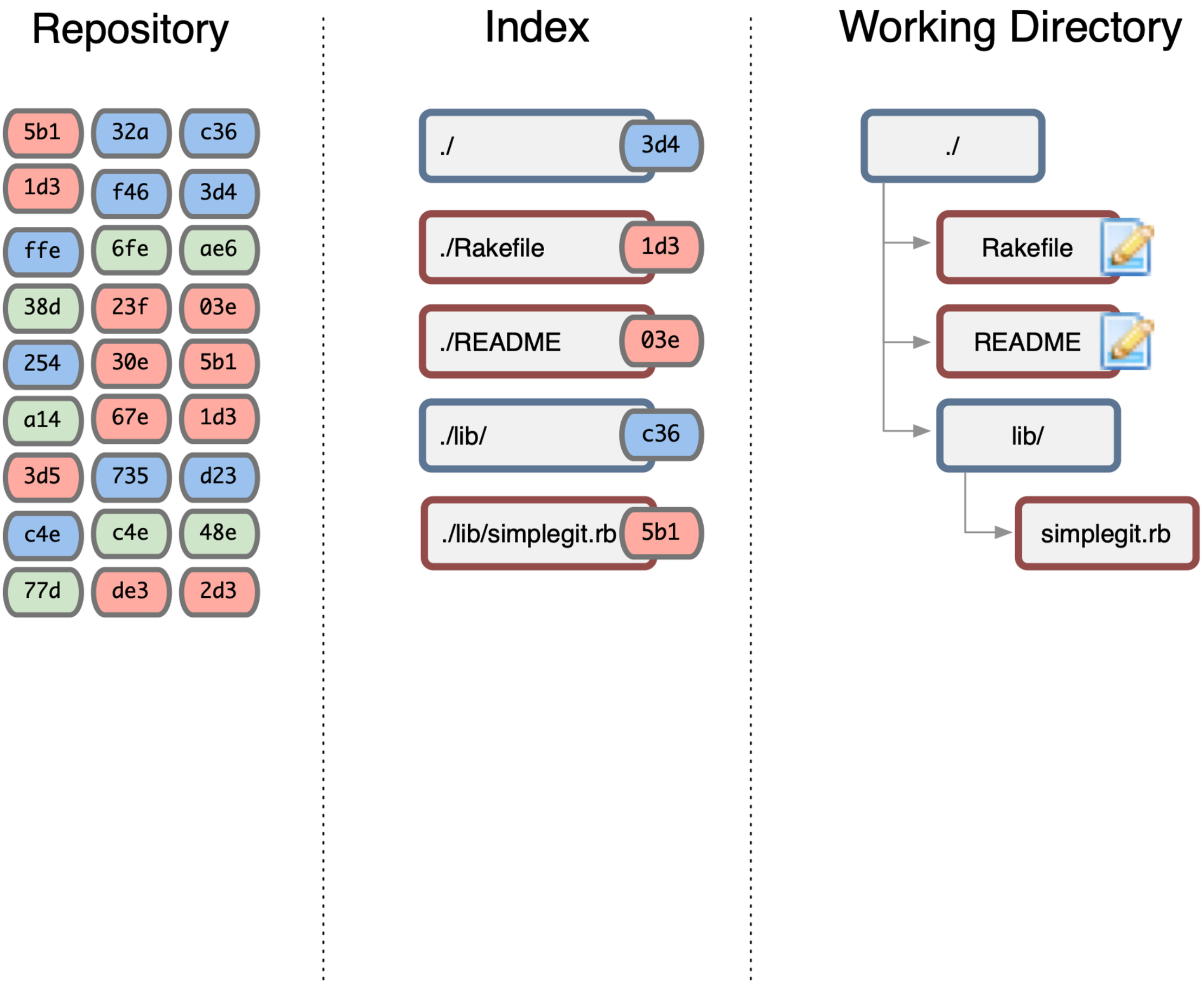


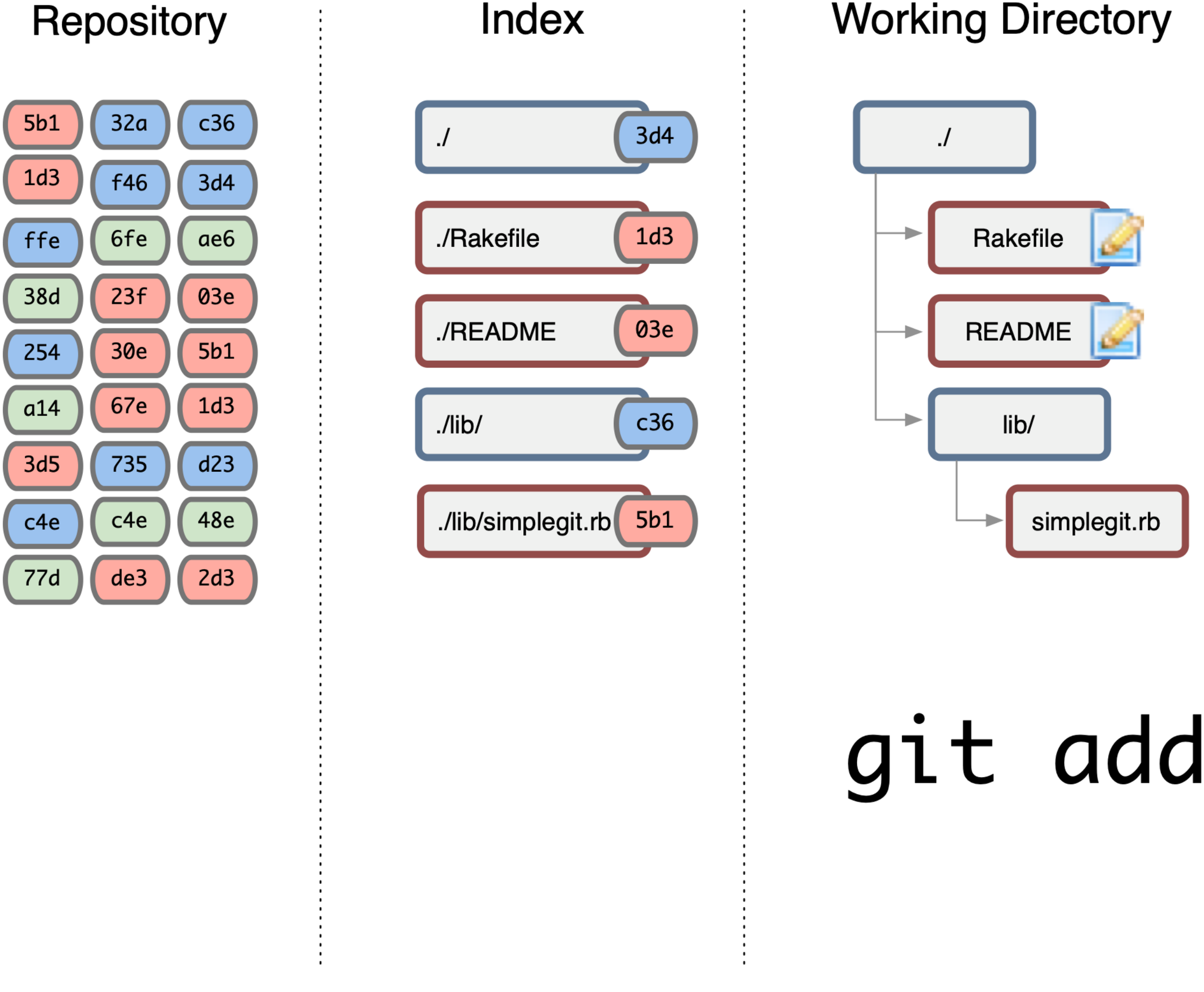
git checkout branch

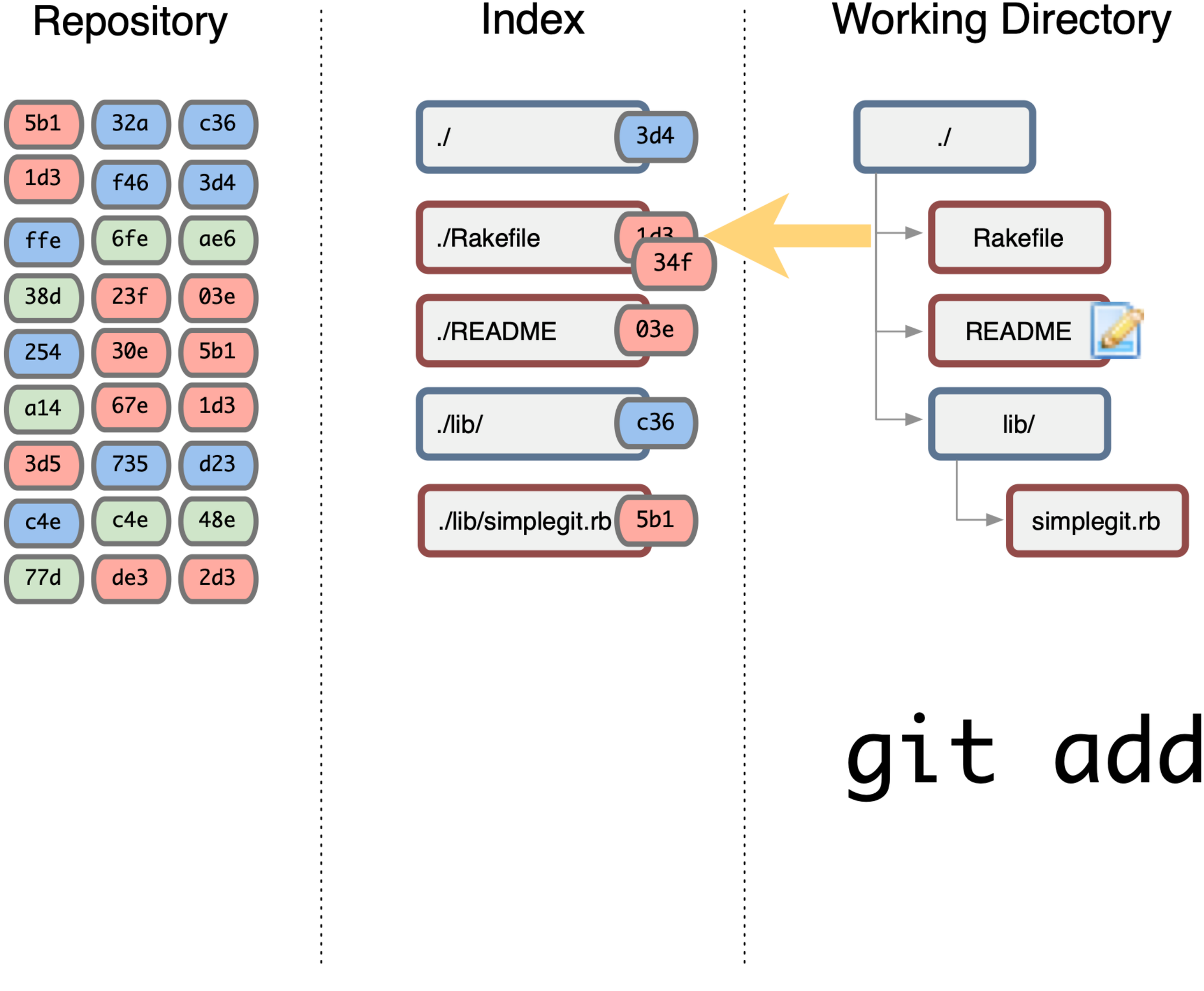

Repository
Index
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
./
3d4
./Rakefile
1d3
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1

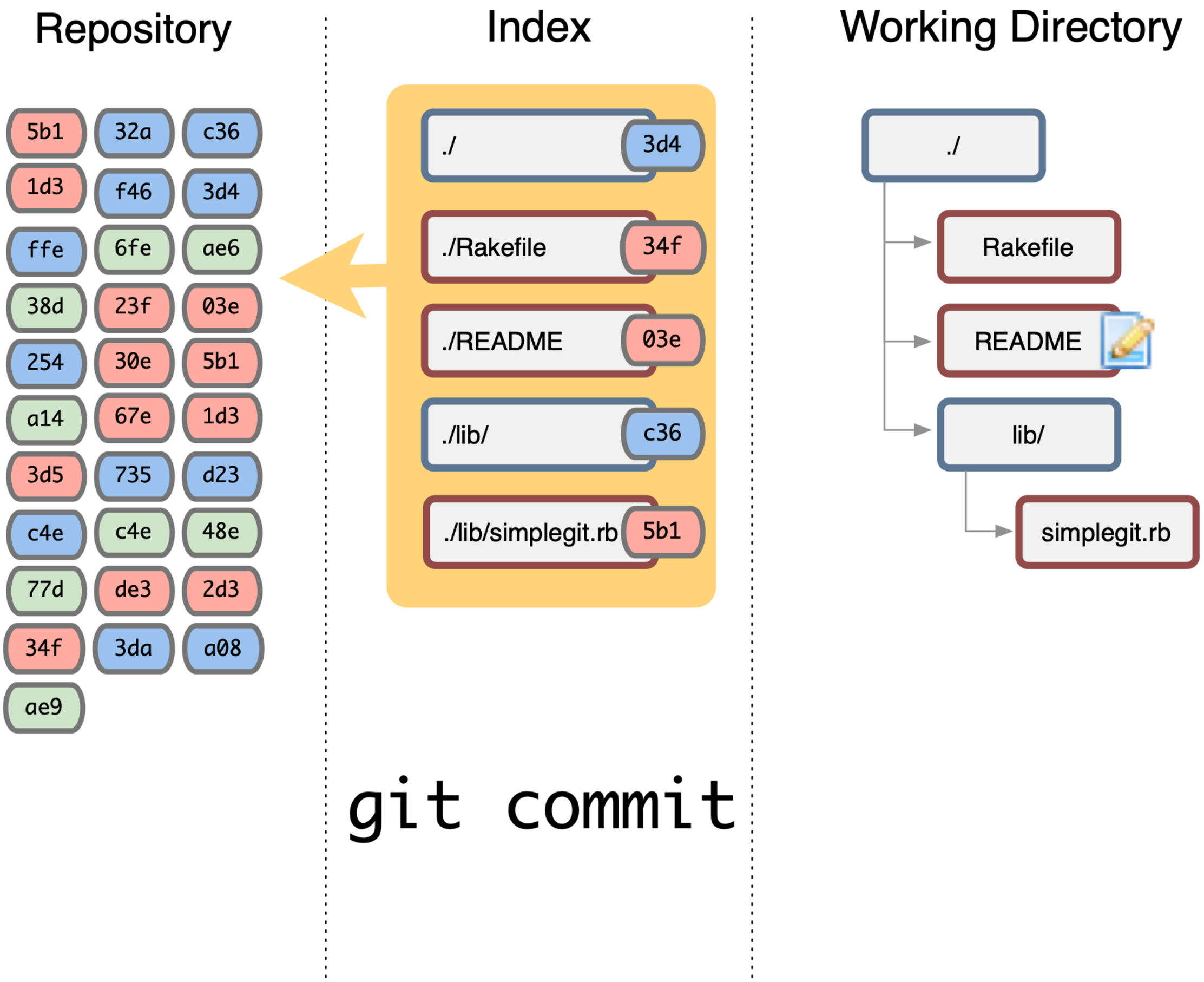

Repository
Index
Working Directory
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
./
3d4
./Rakefile
1d3
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1
./
Rakefile
README
lib/
simplegit.rb

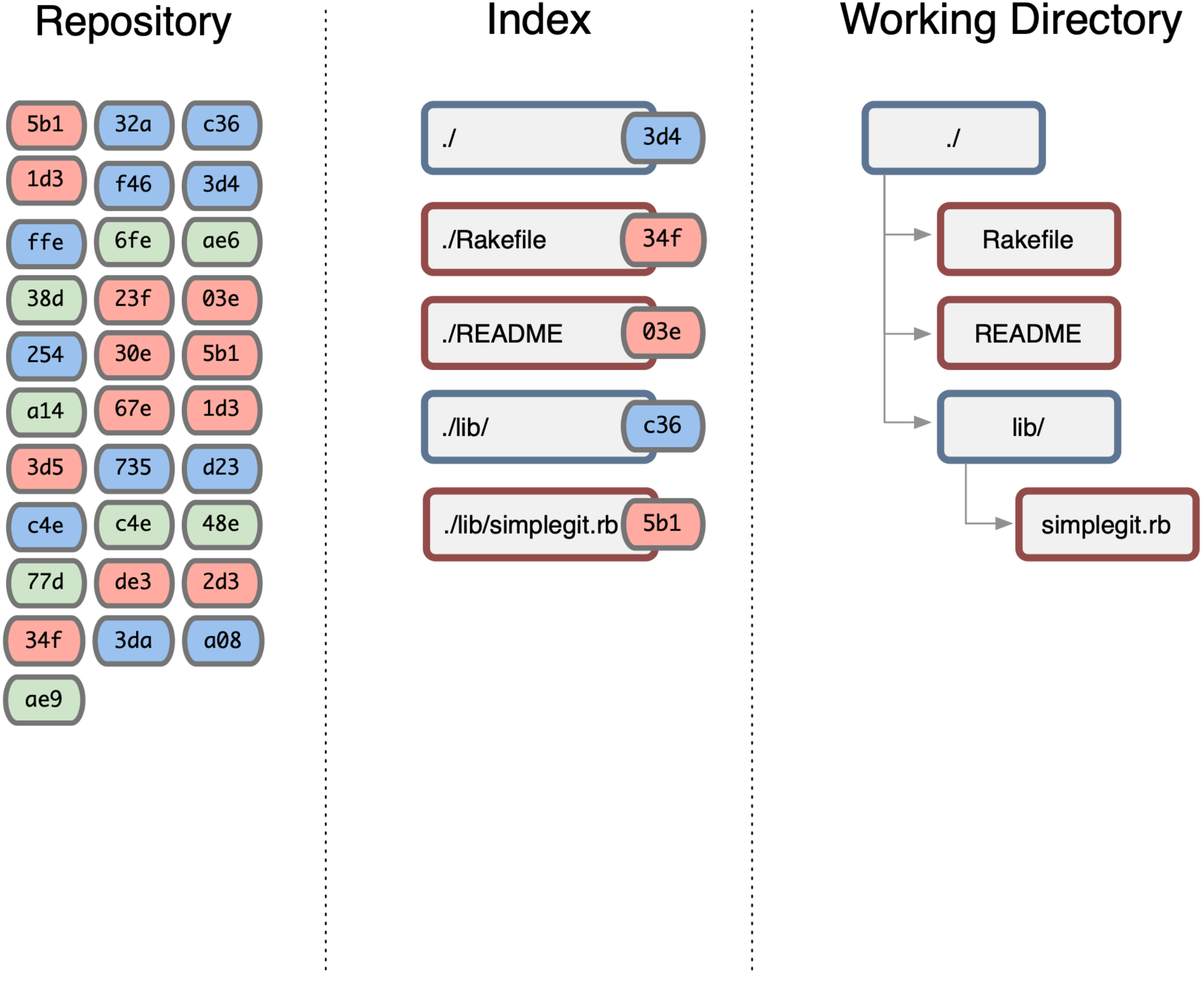

Repository
Index
Working Directory
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
./
3d4
./Rakefile
1d3
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1
./
Rakefile
README
lib/
simplegit.rb


Repository
Index
Working Directory
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
./
3d4
./Rakefile
1d3
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1
./
Rakefile
README
lib/
simplegit.rb


Repository
Index
Working Directory
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
./
3d4
./Rakefile
1d3
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1
./
Rakefile
README
lib/
simplegit.rb
git add


Repository
Index
Working Directory
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
./
3d4
./Rakefile
34f
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1
./
Rakefile
README
lib/
simplegit.rb
git add


Repository
Index
Working Directory
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
34f
3da
a08
ae9
./
3d4
./Rakefile
34f
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1
./
Rakefile
README
lib/
simplegit.rb
git commit


Repository
Index
Working Directory
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
34f
3da
a08
ae9
./
3d4
./Rakefile
34f
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1
./
Rakefile
README
lib/
simplegit.rb

OGGETTO TREE

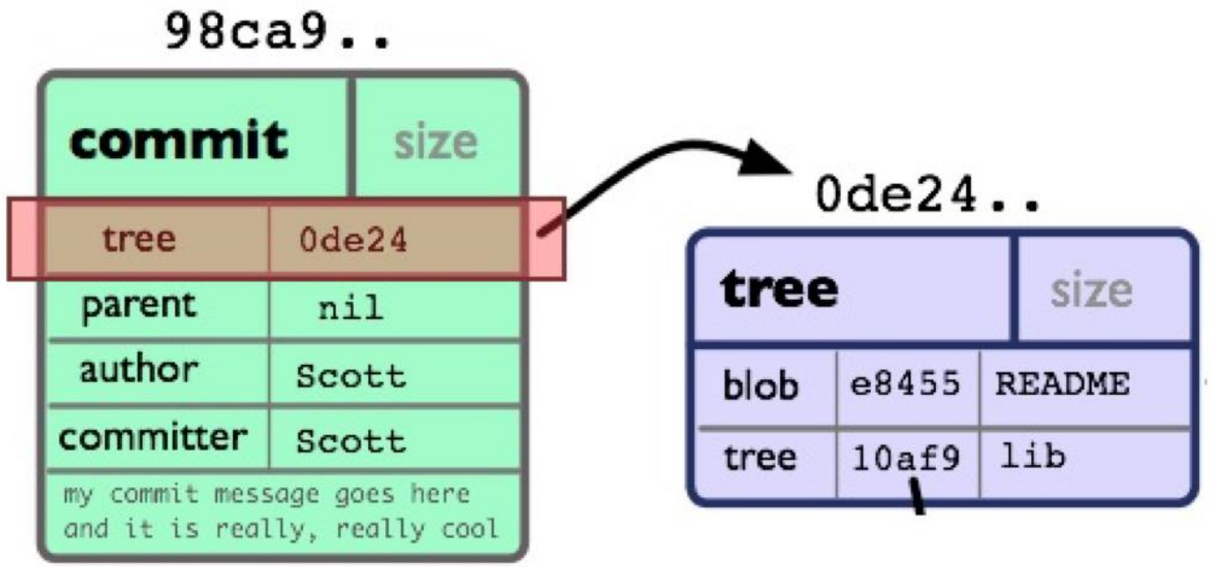
- Rappresenta una directory.
- Contiene una lista degli oggetti contenuti nella directory.
- Ogni entry contiene:
  - Dimensione dell’oggetto
  - Tipo dell’oggetto
  - Riferimento (checksum) all’oggetto
  - Nome dell’oggetto (nome del file o della directory)

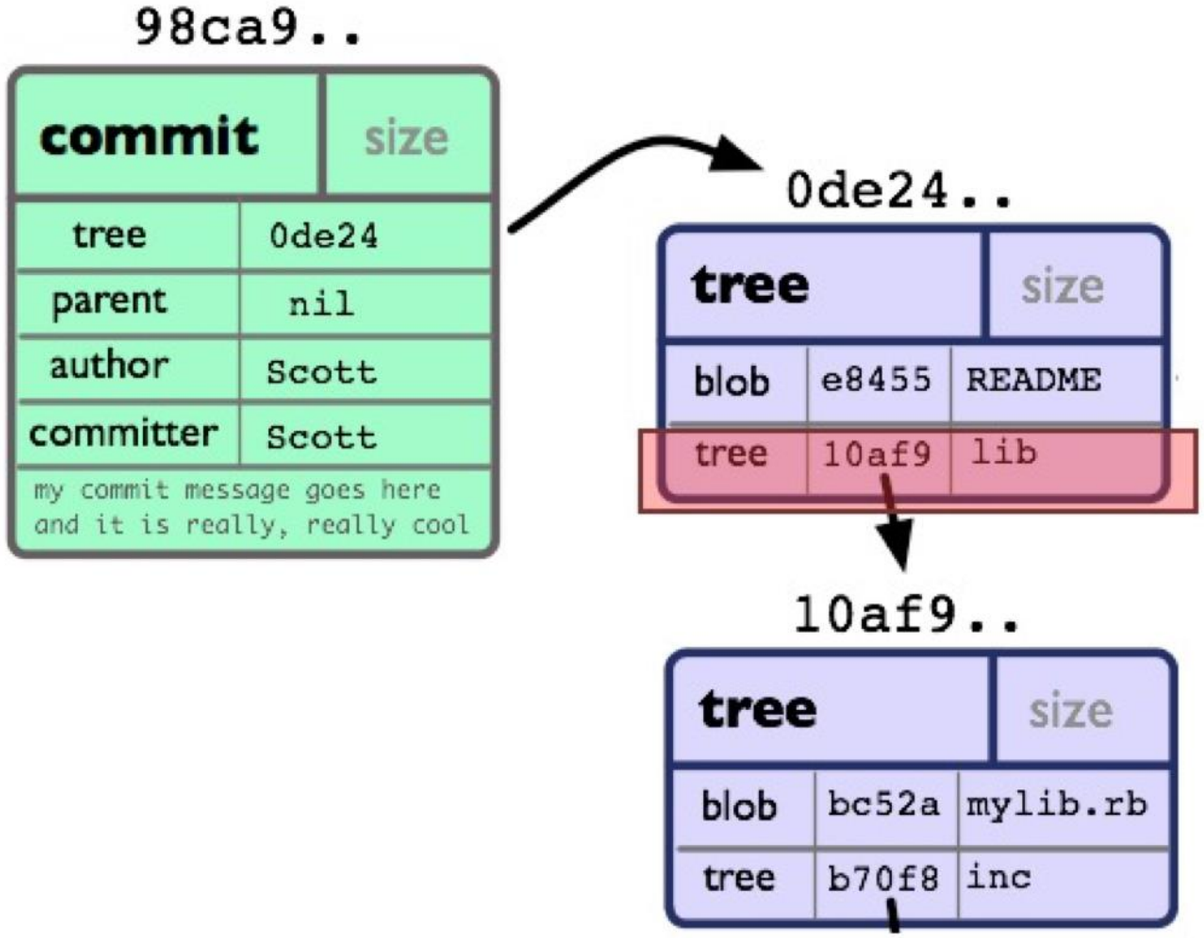
98ca9..

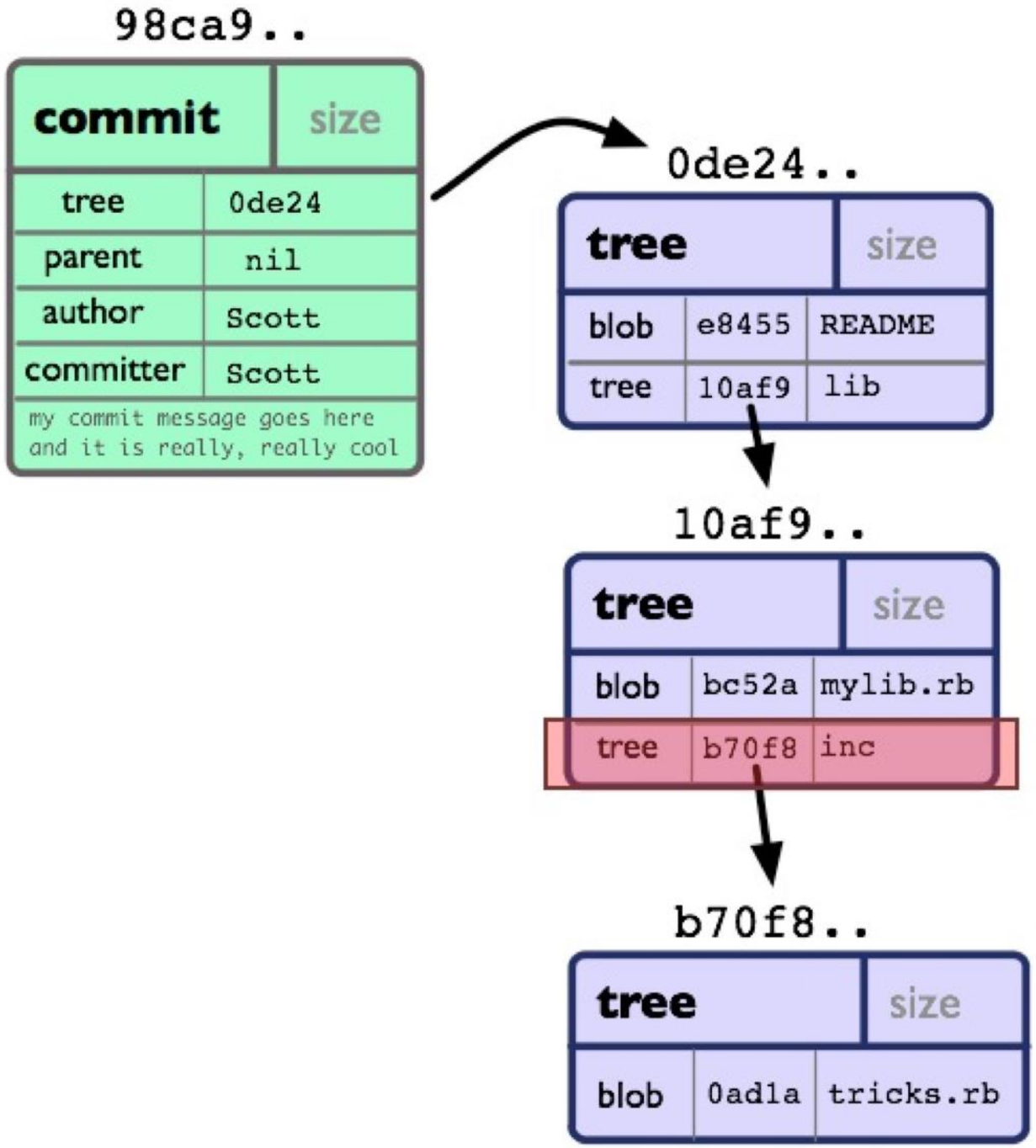
commit	size
tree	0de24
parent	nil
author	Scott
committer	Scott

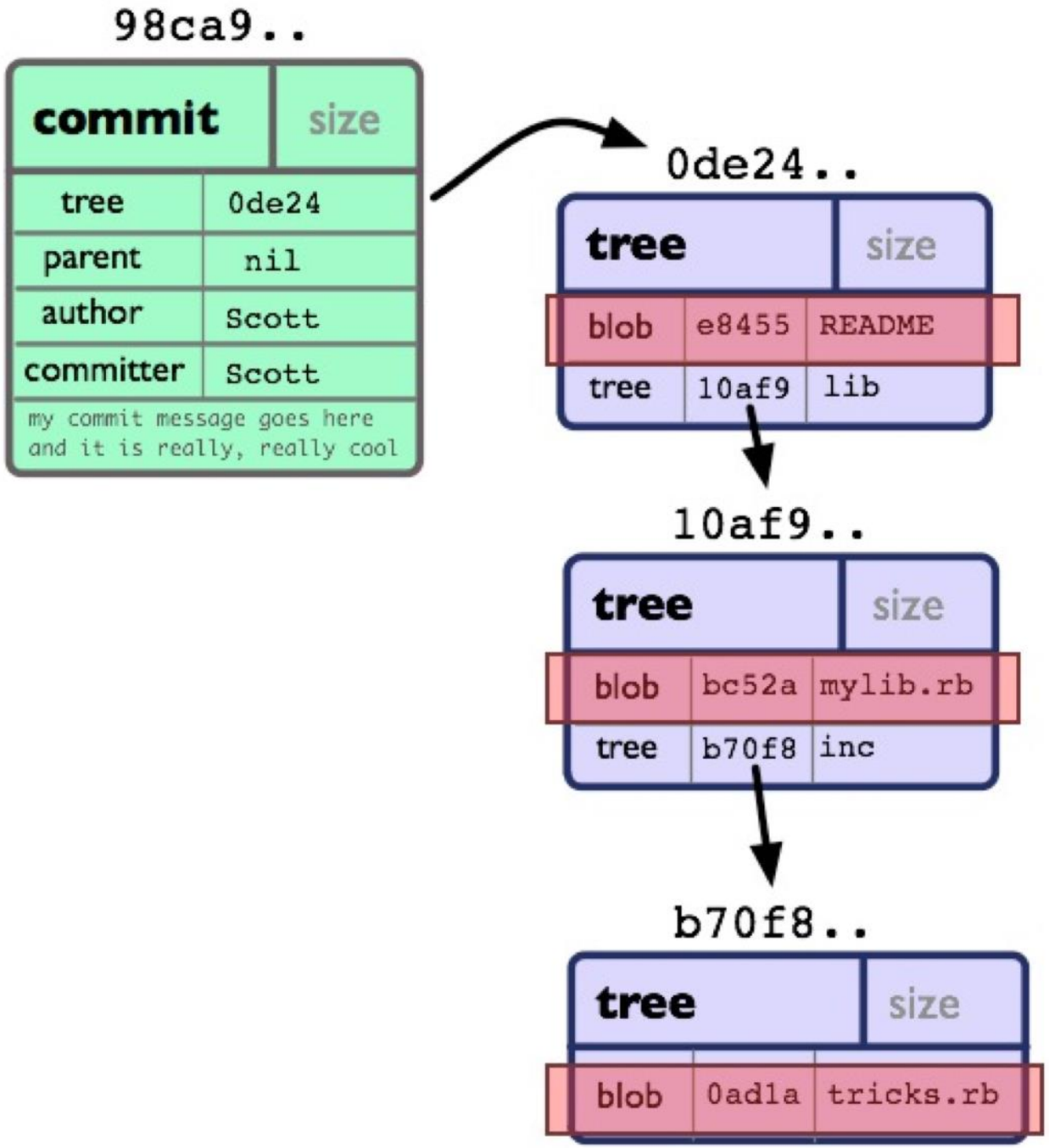
my commit message goes here
and it is really, really cool


98ca9..
commit
size
tree
0de24
parent
nil
author
Scott
committer
Scott
my commit message goes here
and it is really, really cool


98ca9..
commit
size
tree
0de24
parent
nil
author
Scott
committer
Scott
my commit message goes here
and it is really, really cool
0de24..
tree
size
blob
e8455
README
tree
10af9
lib

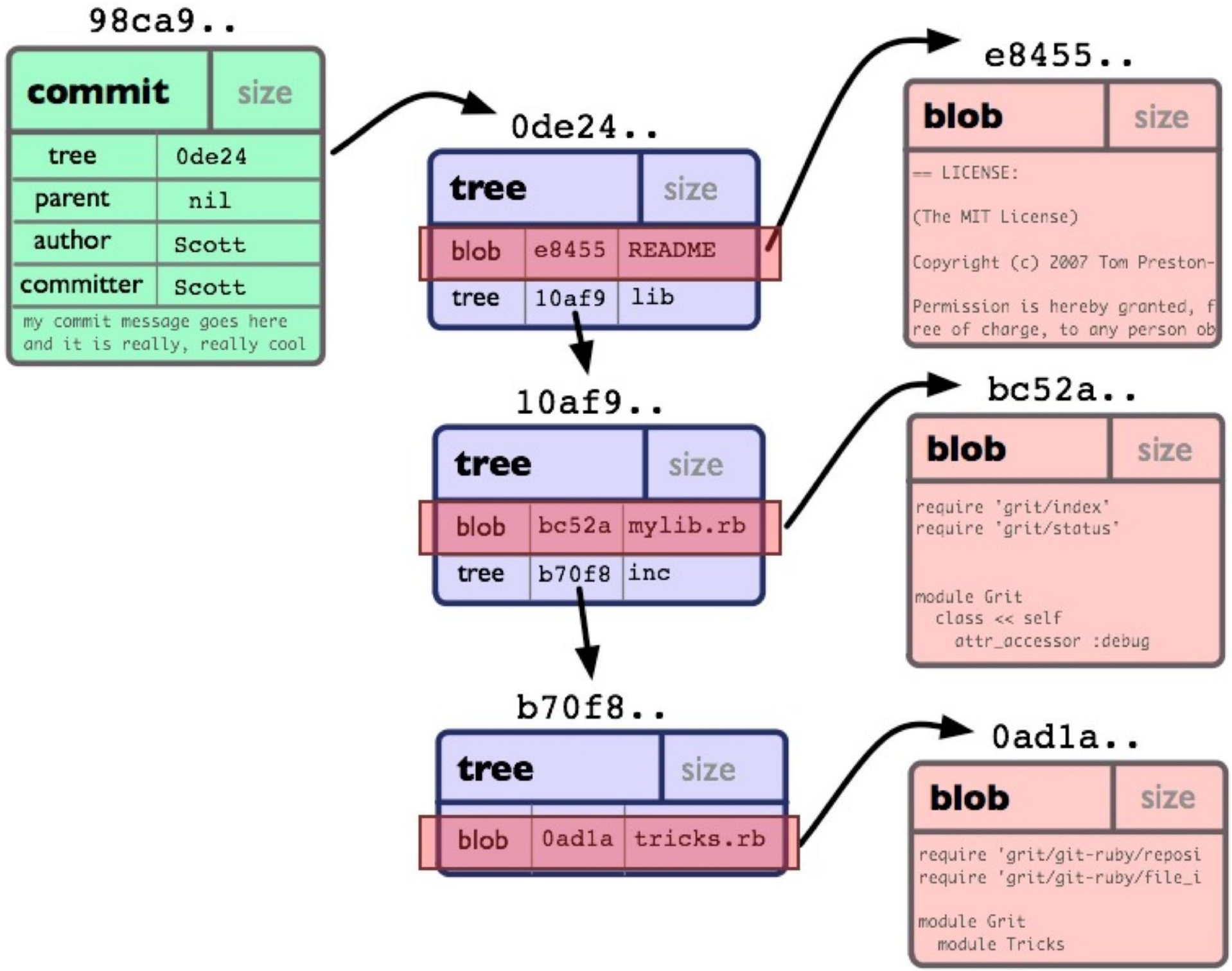

98ca9..
commit
size
tree
0de24
parent
nil
author
Scott
committer
Scott
my commit message goes here
and it is really, really cool
0de24..
tree
size
blob
e8455
README
tree
10af9
lib
10af9..
tree
size
blob
bc52a
mylib.rb
tree
b70f8
inc

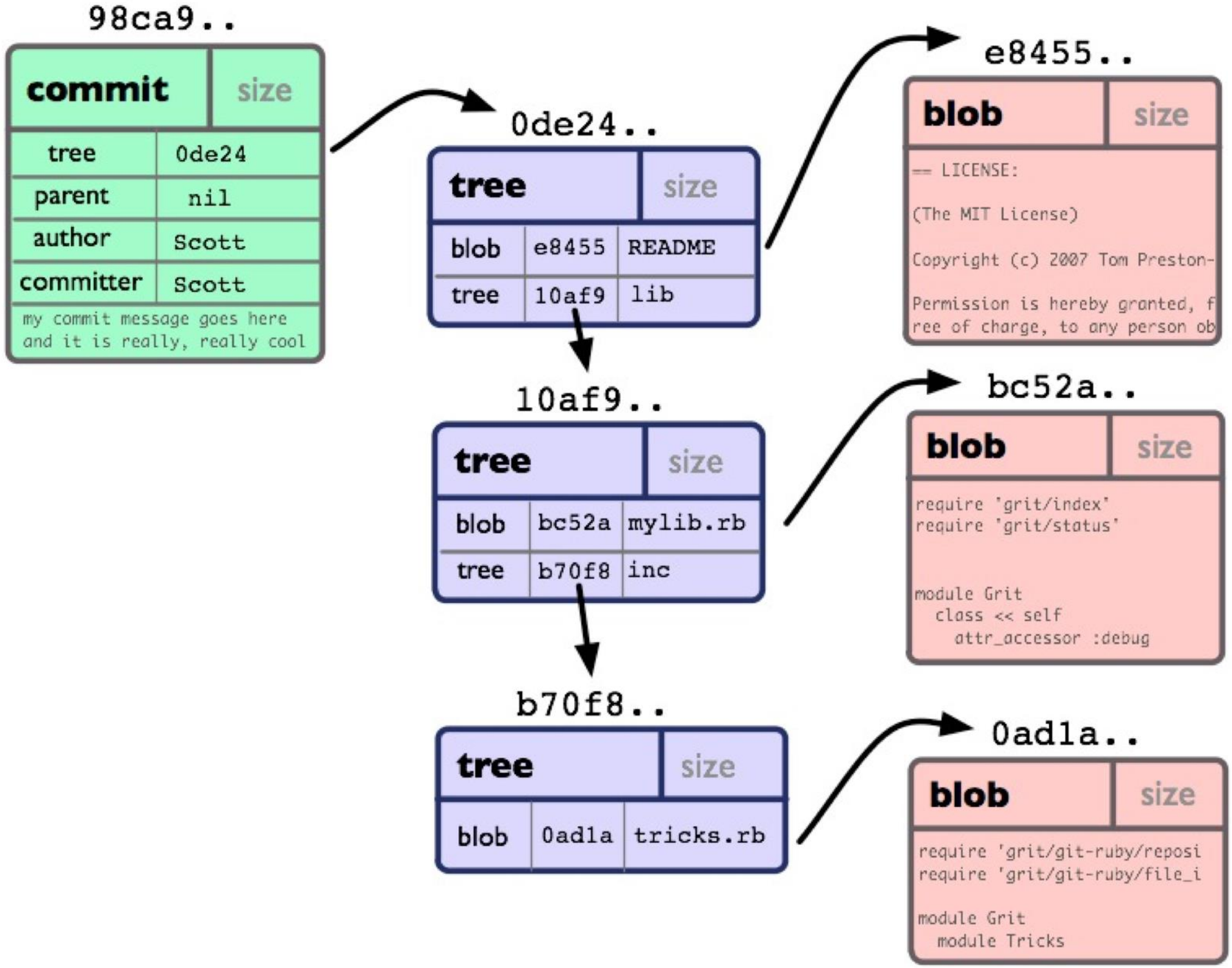

98ca9..
commit
size
tree
0de24
parent
nil
author
Scott
committer
Scott
my commit message goes here
and it is really, really cool
0de24..
tree
size
blob
e8455
README
tree
10af9
lib
10af9..
tree
size
blob
bc52a
mylib.rb
tree
b70f8
inc
b70f8..
tree
size
blob
0ad1a
tricks.rb

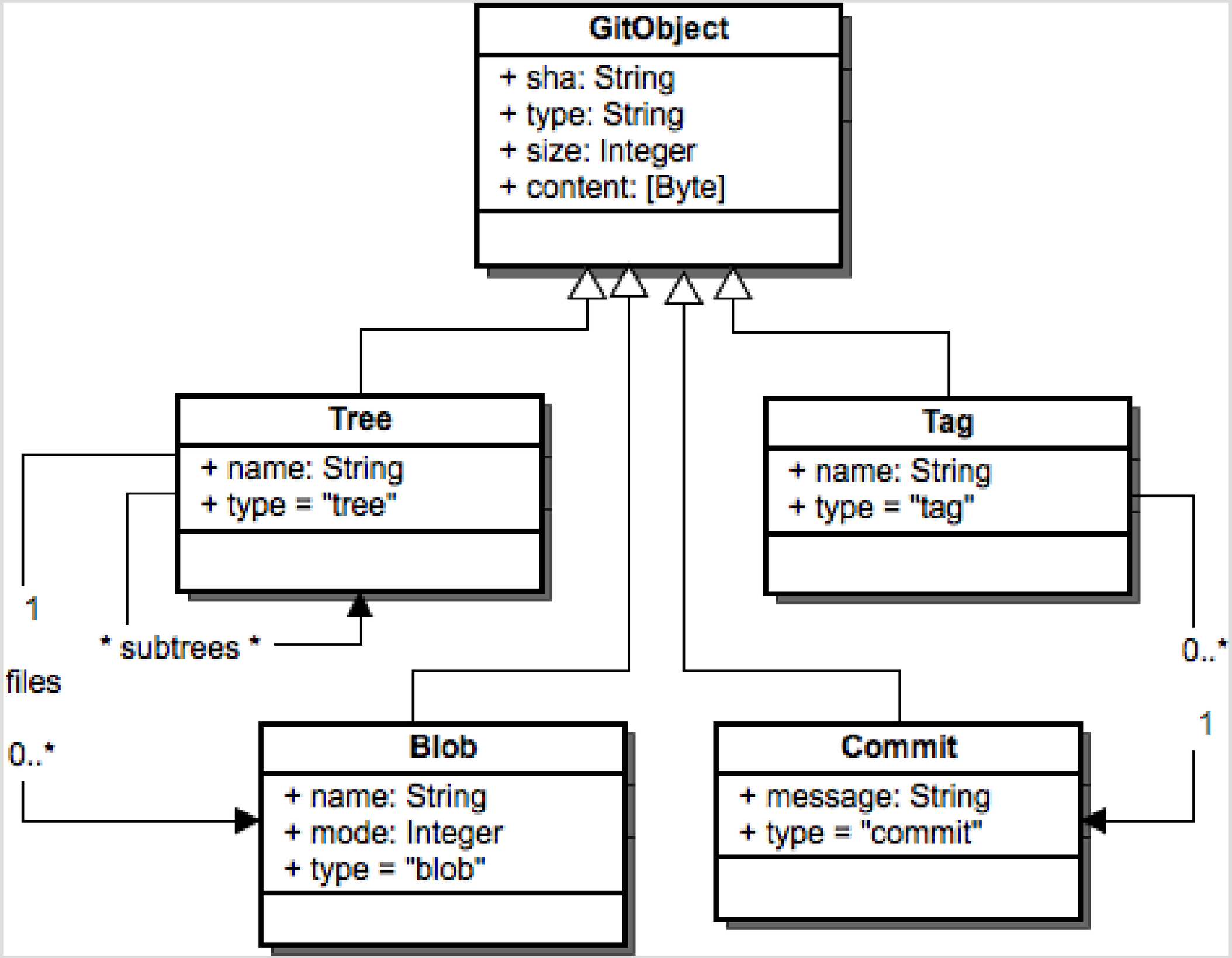

98ca9..
commit
size
tree
0de24
parent
nil
author
Scott
committer
Scott
my commit message goes here
and it is really, really cool
0de24..
tree
size
blob
e8455
README
tree
10af9
lib
10af9..
tree
size
blob
bc52a
mylib.rb
tree
b70f8
inc
b70f8..
tree
size
blob
0ad1a
tricks.rb

OGGETTO BLOB

- Rappresenta un file versionato (binario).
- Contiene esclusivamente il contenuto dei file versionati.


98ca9..
commit
size
tree
0de24
parent
nil
author
Scott
committer
Scott
my commit message goes here
and it is really, really cool
0de24..
tree
size
blob
e8455
README
tree
10af9
lib
e8455..
blob
size
== LICENSE:
(The MIT License)
Copyright (c) 2007 Tom Preston-
Permission is hereby granted, f
ree of charge, to any person ob
10af9..
tree
size
blob
bc52a
mylib.rb
tree
b70f8
inc
bc52a..
blob
size
require 'grit/index'
require 'grit/status'
module Grit
class << self
attr_accessor :debug
b70f8..
tree
size
blob
0ad1a
tricks.rb
0ad1a..
blob
size
require 'grit/git-ruby/reposi
require 'grit/git-ruby/file_i
module Grit
module Tricks


98ca9..
commit
size
tree
0de24
parent
nil
author
Scott
committer
Scott
my commit message goes here
and it is really, really cool
0de24..
tree
size
blob
e8455
README
tree
10af9
lib
e8455..
blob
size
== LICENSE:
(The MIT License)
Copyright (c) 2007 Tom Preston-
Permission is hereby granted, f
ree of charge, to any person ob
10af9..
tree
size
blob
bc52a
mylib.rb
tree
b70f8
inc
bc52a..
blob
size
require 'grit/index'
require 'grit/status'
module Grit
class << self
attr_accessor :debug
b70f8..
tree
size
blob
0ad1a
tricks.rb
0ad1a..
blob
size
require 'grit/git-ruby/reposi
require 'grit/git-ruby/file_i
module Grit
module Tricks


GitObject
+ sha: String
+ type: String
+ size: Integer
+ content: [Byte]
Tree
+ name: String
+ type = "tree"
Tag
+ name: String
+ type = "tag"
1
* subtrees *
files
0..*
0..*
1
Blob
+ name: String
+ mode: Integer
+ type = "blob"
Commit
+ message: String
+ type = "commit"

CAT-FILE

- Per controllare il contenuto dei vari oggetti possiamo usare il comando visto precedentemente

```
$ git cat-file –t <hash>


$ git cat-file –p  <hash>
```

- Per controllare cosa contengono i riferimenti, possiamo usare il cat classico del sistema operativo
  - I file che rappresentano riferimenti non sono compressi

BRANCHING

BRANCHING

Il “branching” serve per gestire un’evoluzione non lineare di un software. Esempi:

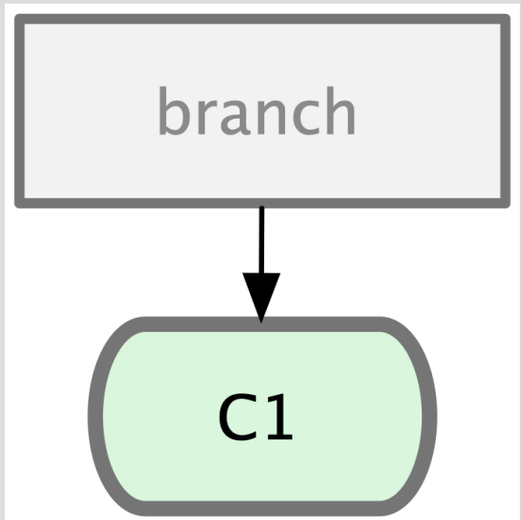
- gestire contemporaneamente più linee alternative di sviluppo
- gestire l’implementazione di nuove funzionalità
- bug fixing
- versioni alternative

Un branch è una linea di evoluzione del software

- ha i suoi commit
- può essere relazionato ad altri branch
- ha un suo stato attuale

BRANCHING

In Git un branch è un riferimento (puntatore) ad un commit

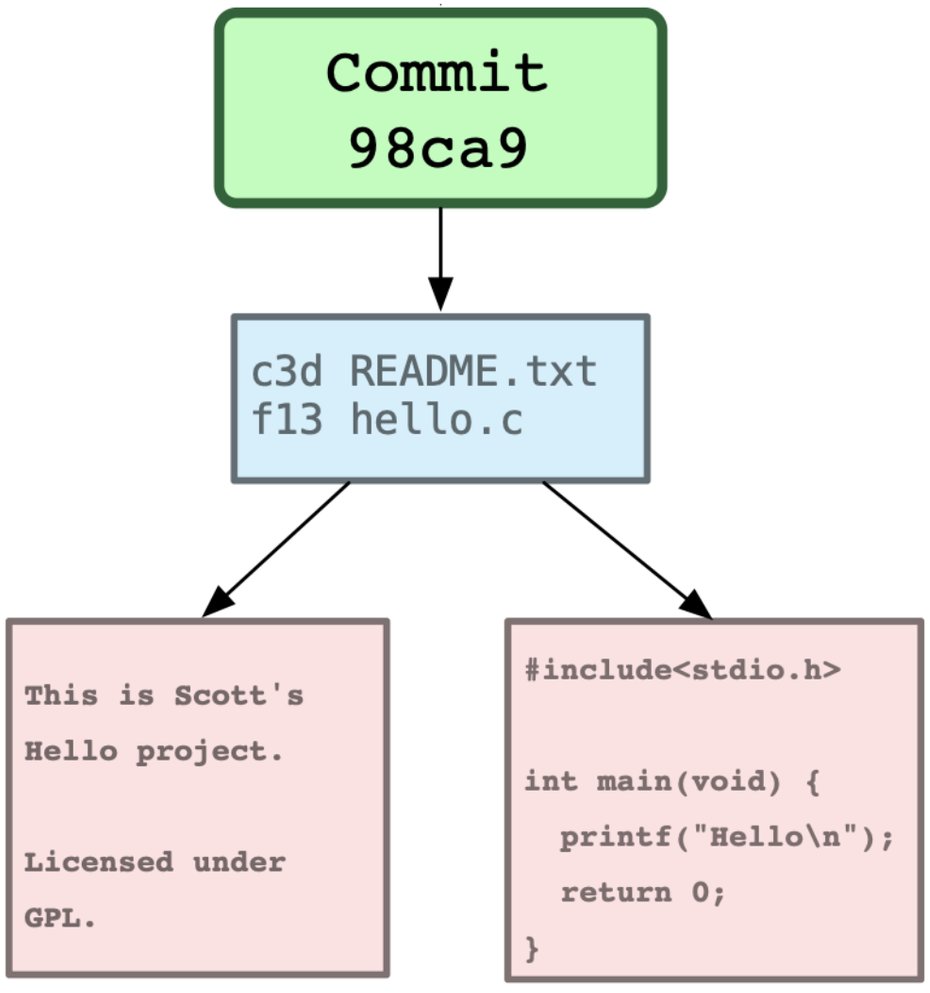


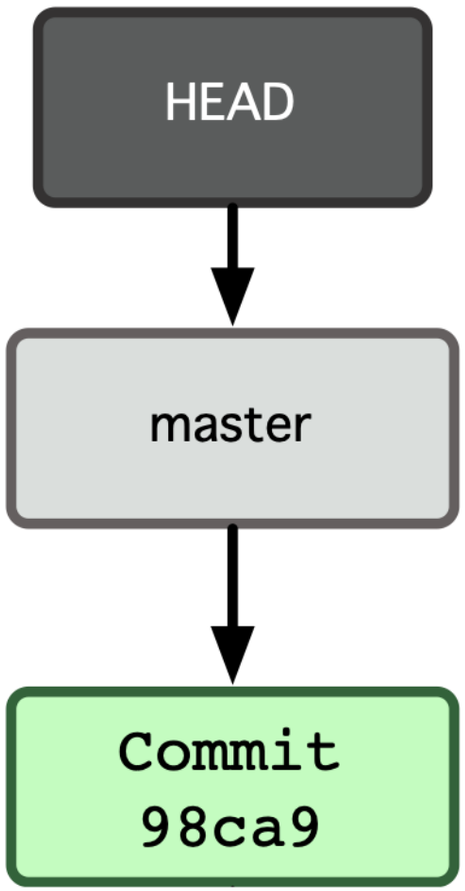
Per ottenere la lista dei branch:

```
$ git branch -a
```

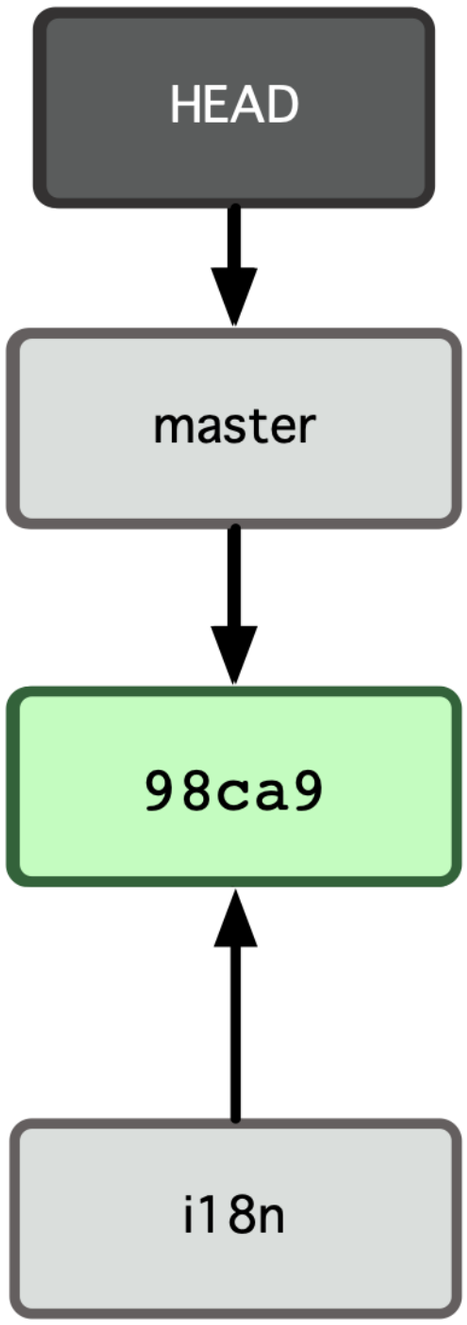
Per passare ad un altro branch:

```
$ git checkout <nome-branch>
```

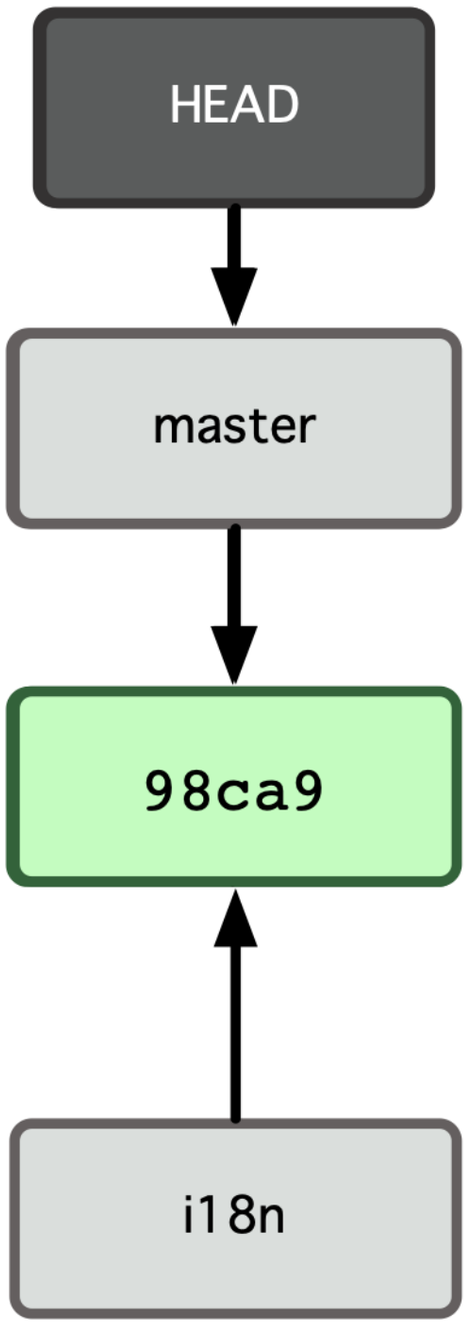

Commit
98ca9
c3d README.txt
f13 hello.c
This is Scott's
Hello project.
Licensed under
GPL.
#include<stdio.h>
int main(void) {
printf("Hello\n");
return 0;
}


HEAD
master
Commit
98ca9

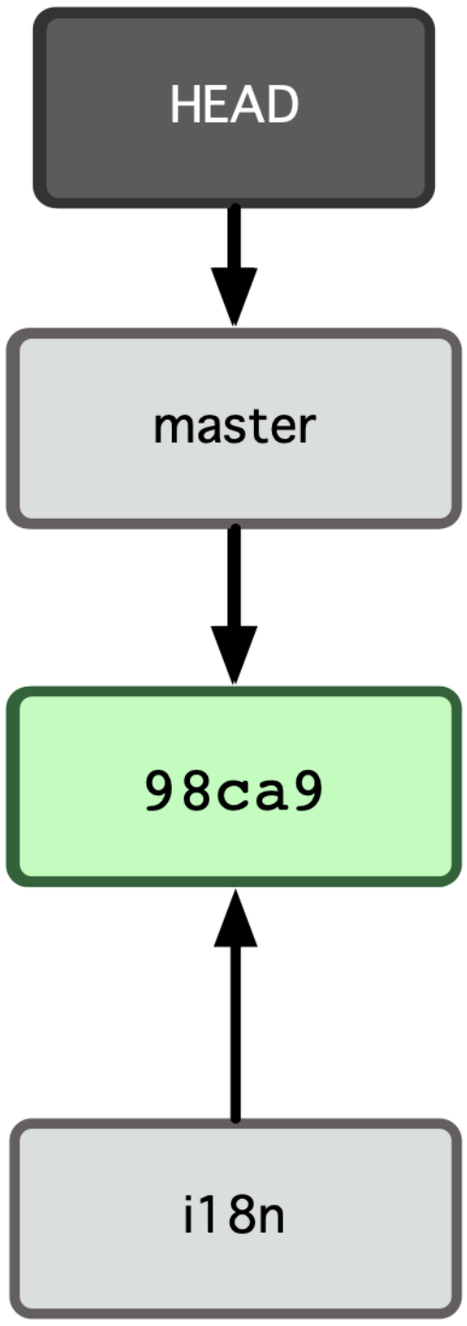
```
git branch i18n
```



git branch



git branch

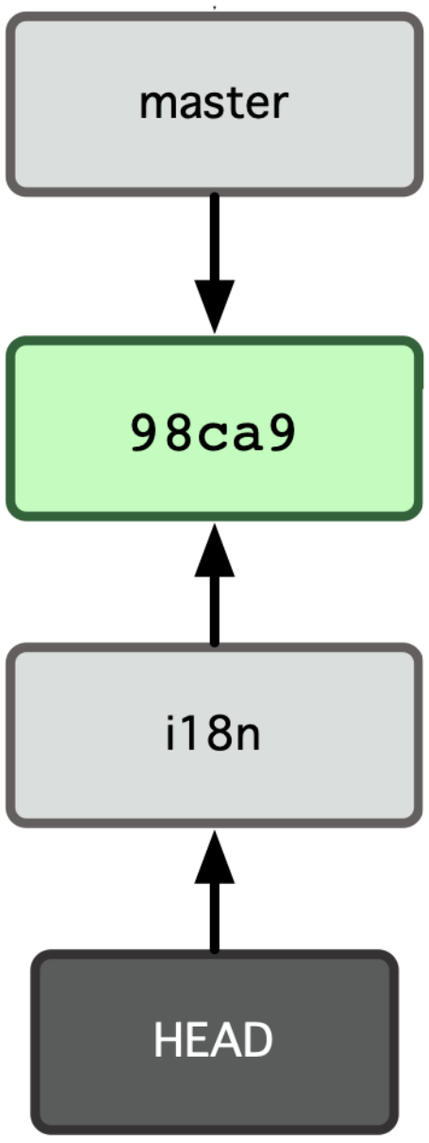



```
$ git branch
* master
  i18n
```

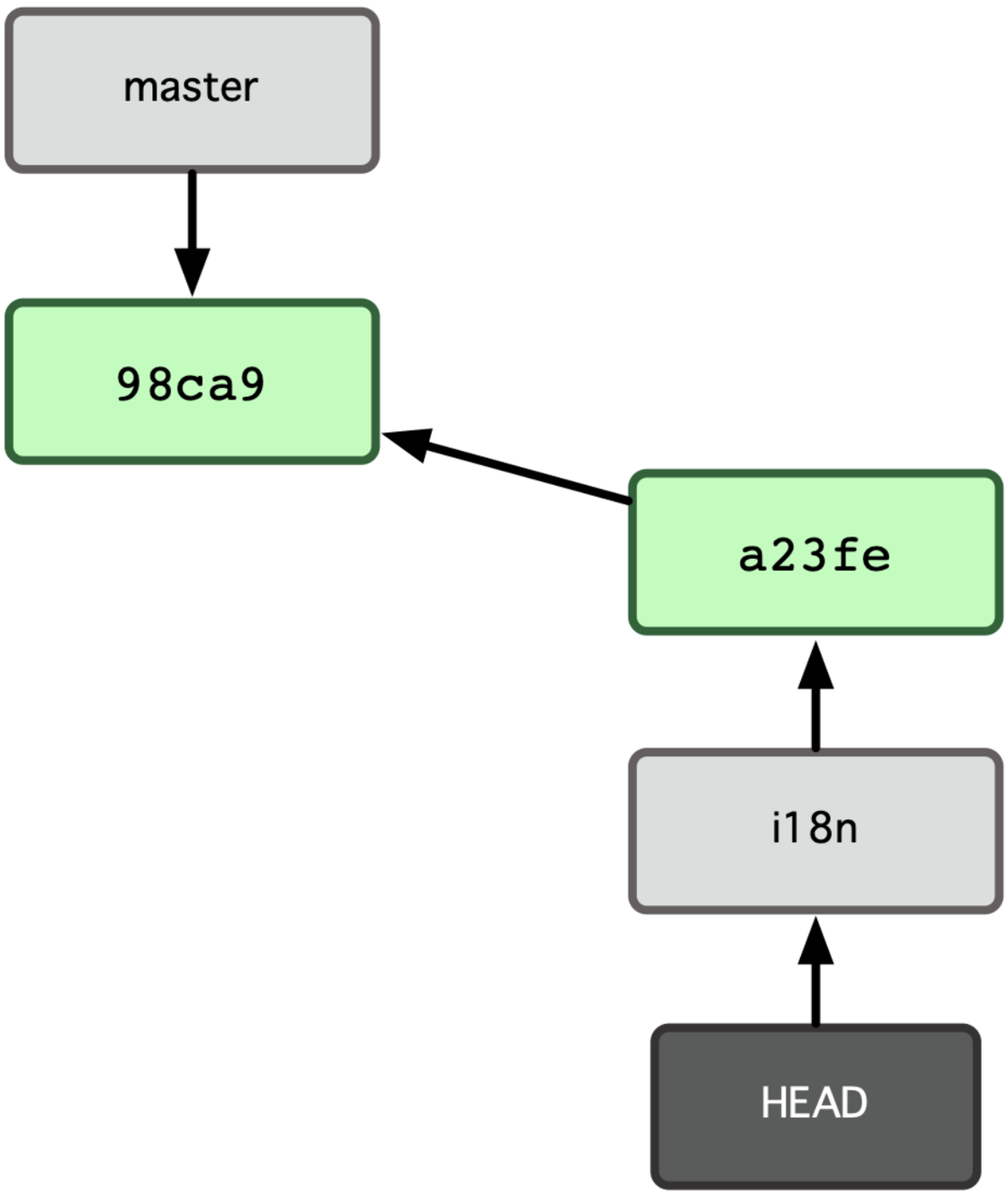
```
$ find .git/refs
.git/refs
.git/refs/heads
.git/refs/heads/master
.git/refs/heads/i18n
```

```
$ cat .git/refs/heads/master
98ca909dc9e38af91565082bdf93577ff555489e
```

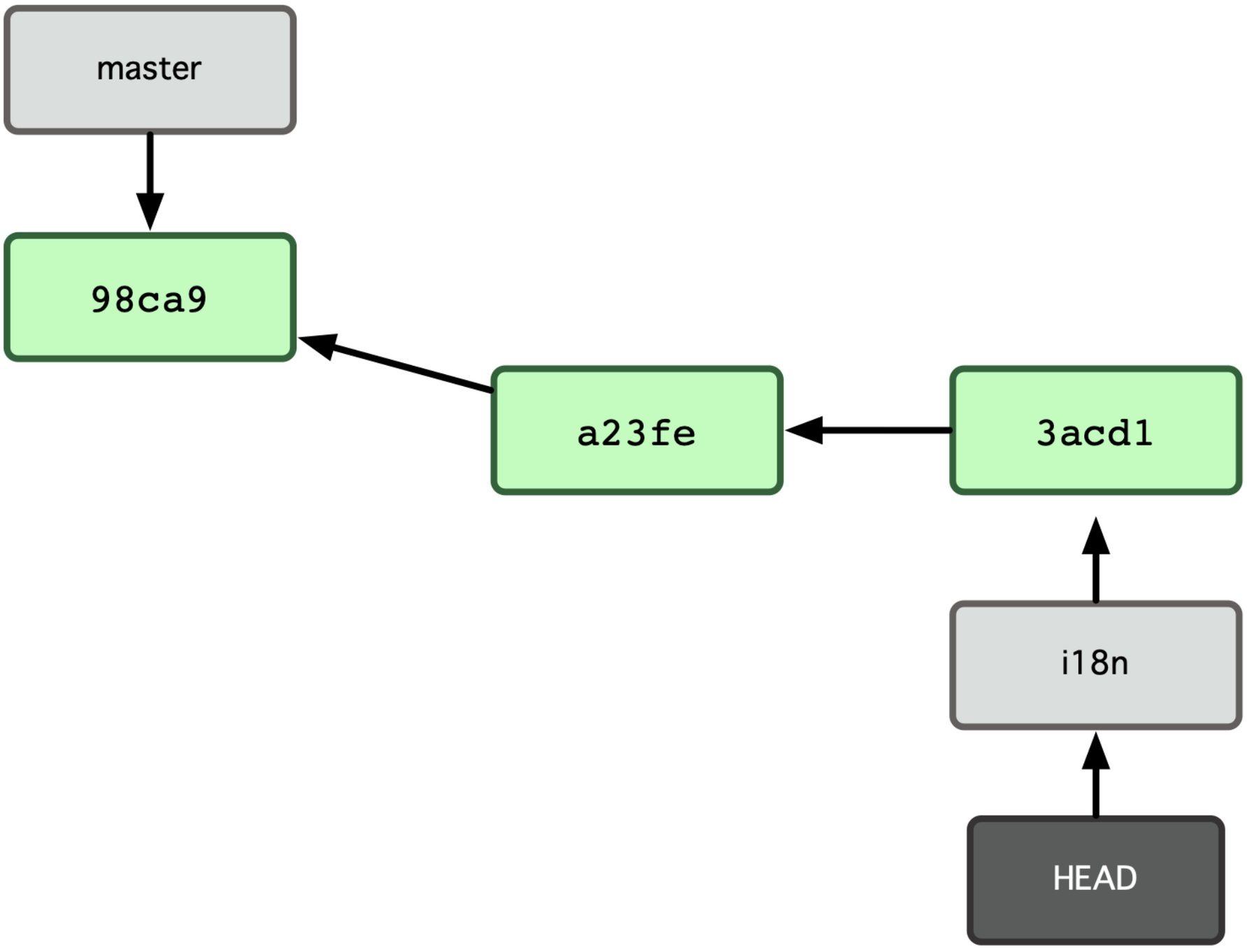
```
git checkout i18n
```



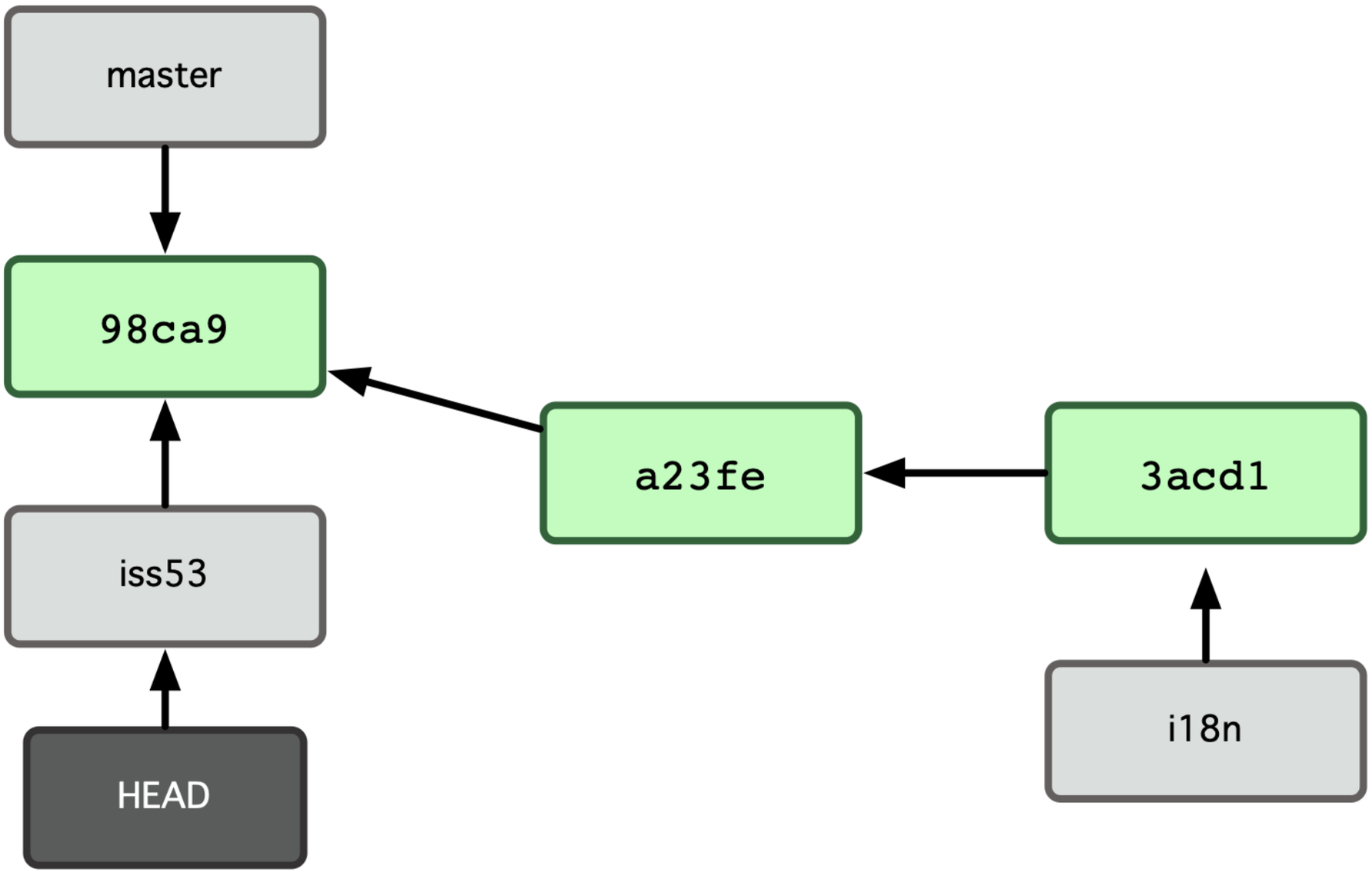
```
git commit
```



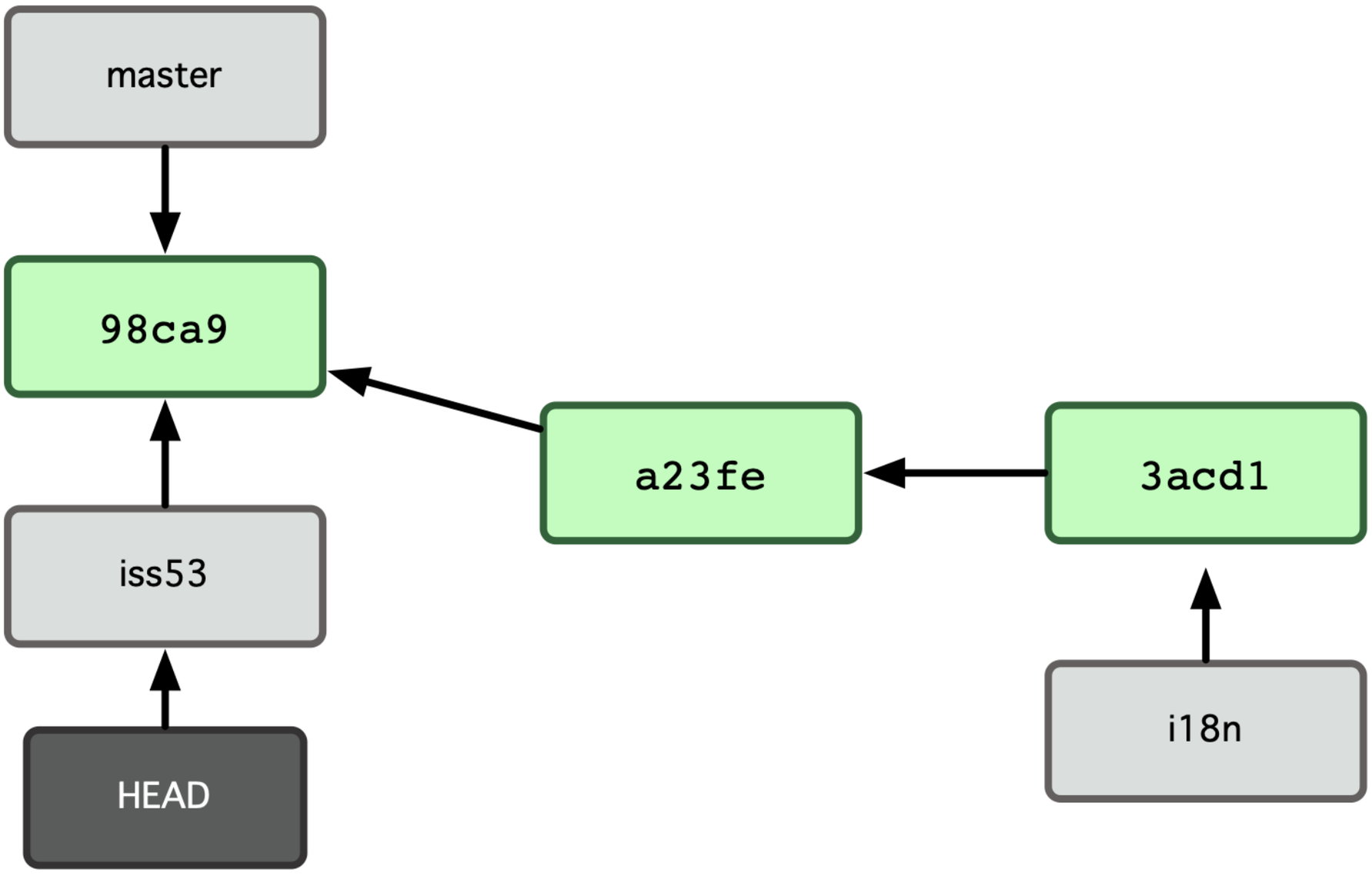
```
git commit
```



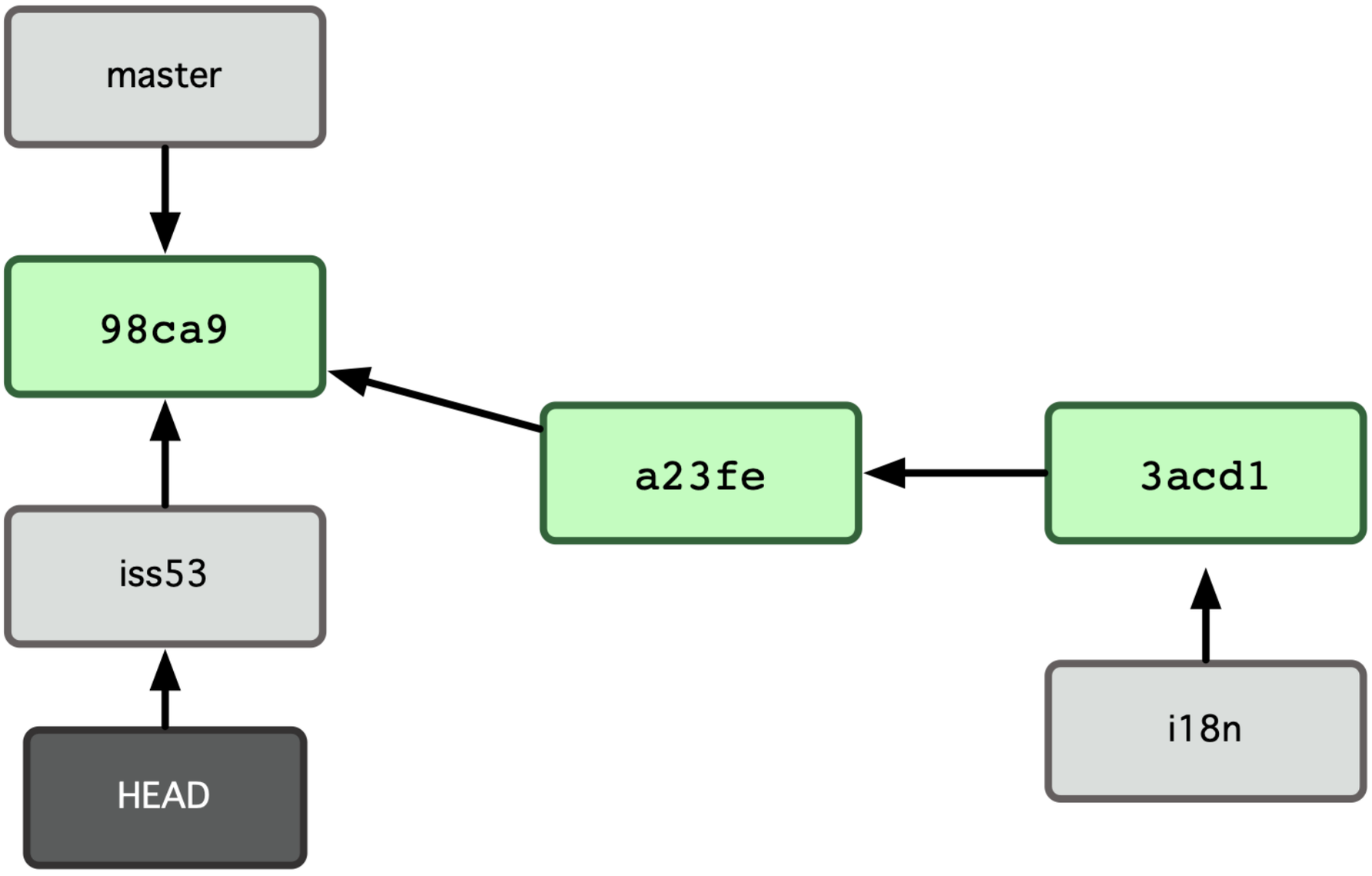
```
git checkout -b iss53 master
```



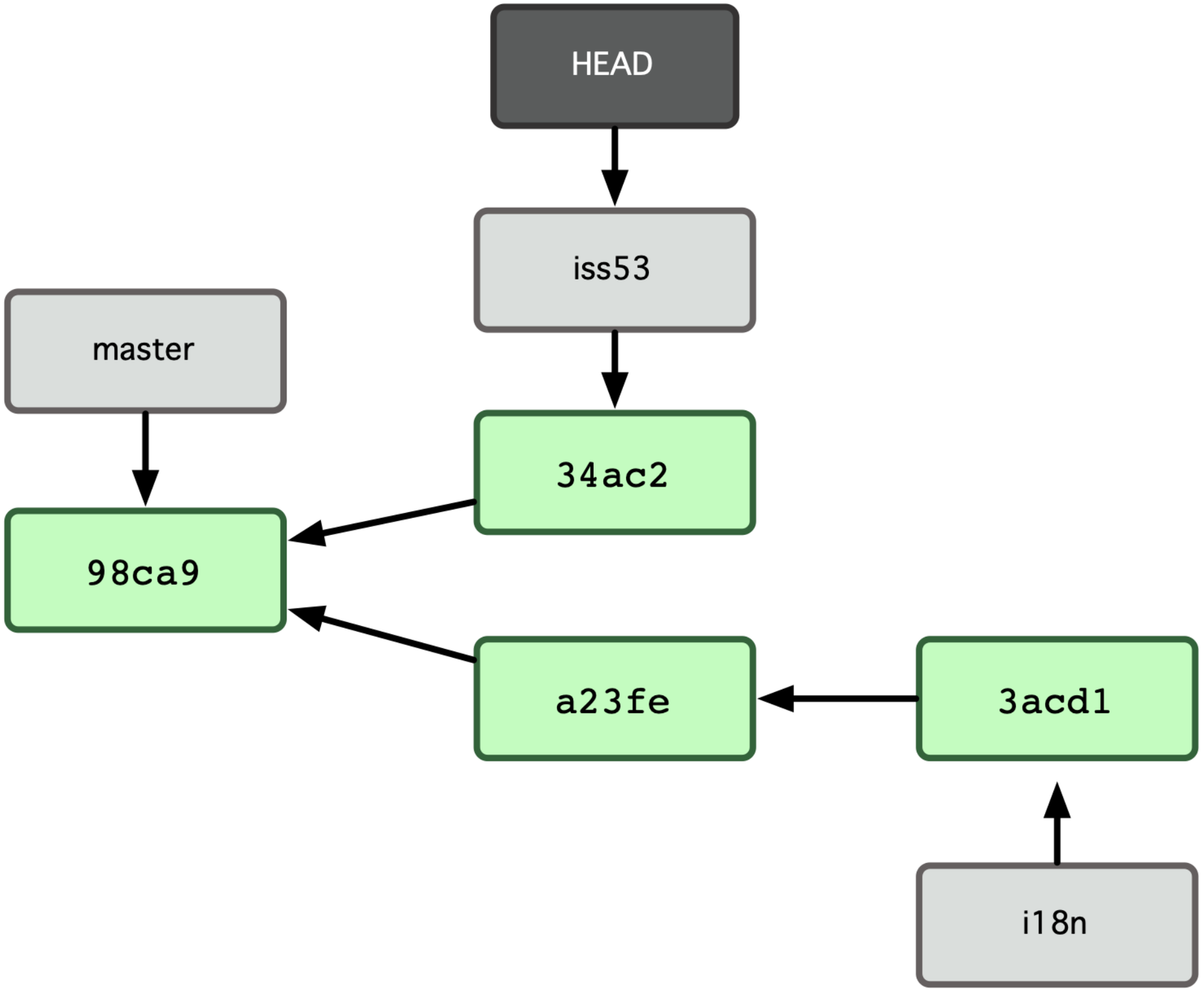
```
git checkout master;
git checkout -b iss53
```



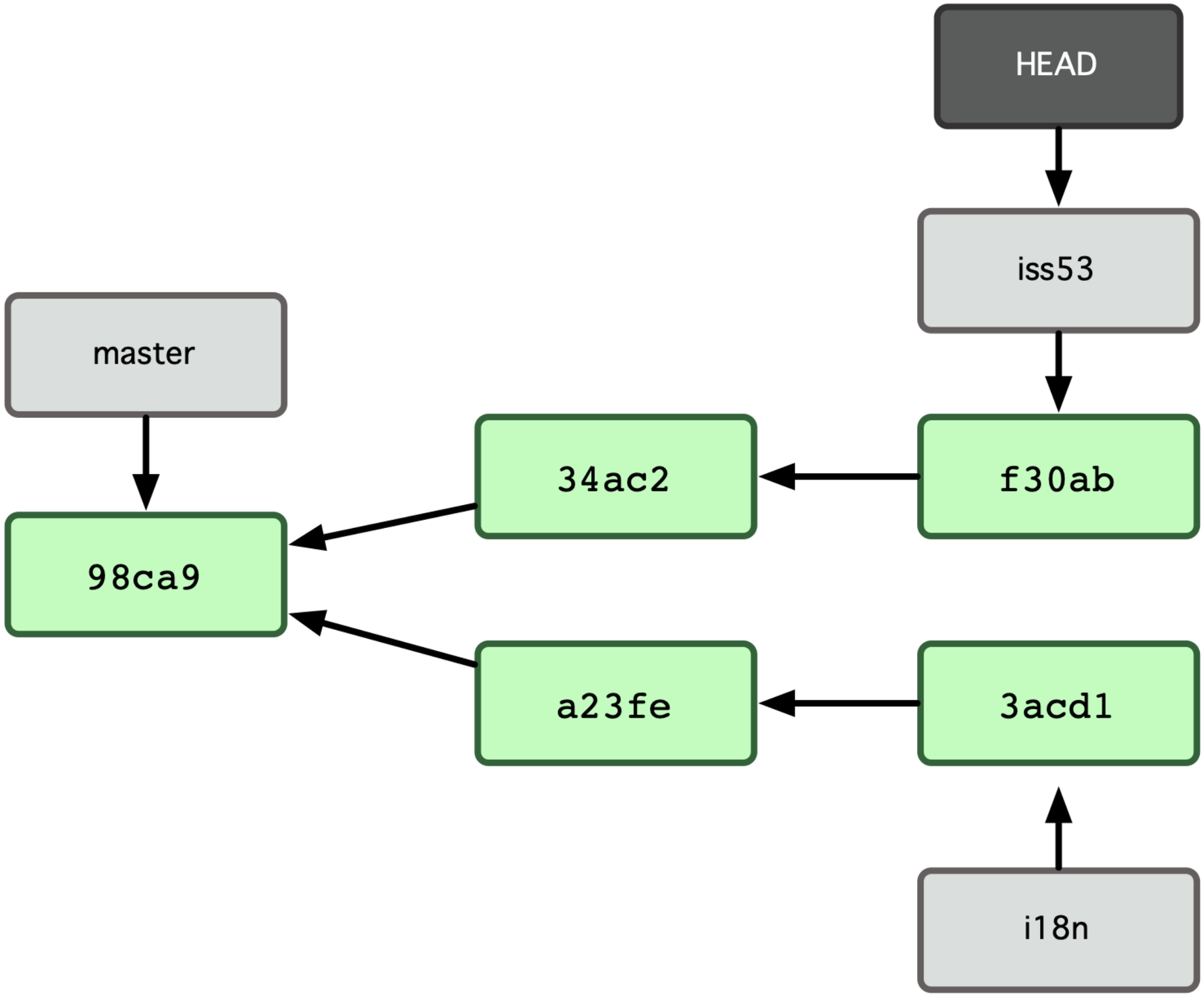
```
git checkout master;
git branch iss53;
git checkout iss53
```



git commit



git commit

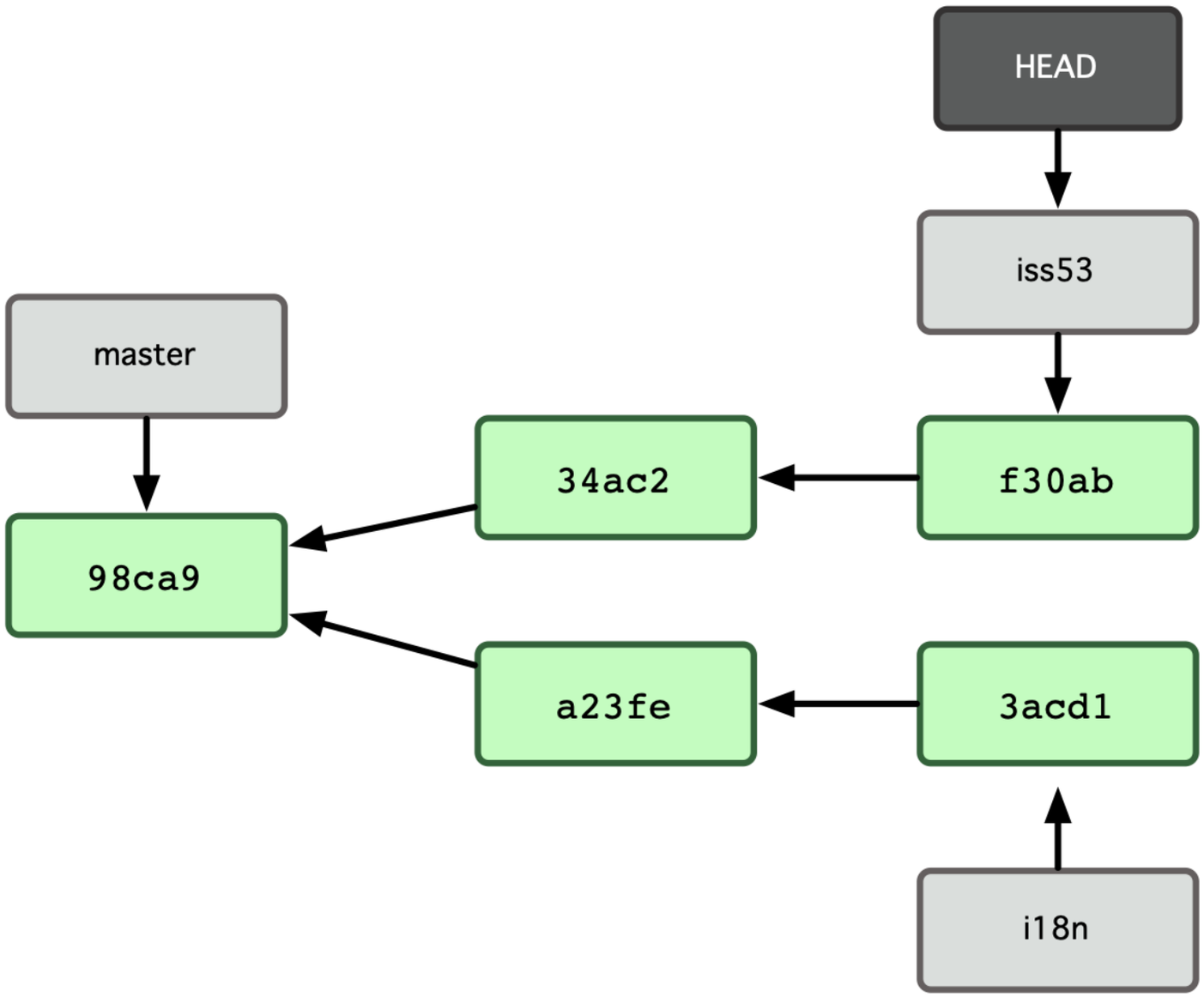


MERGING

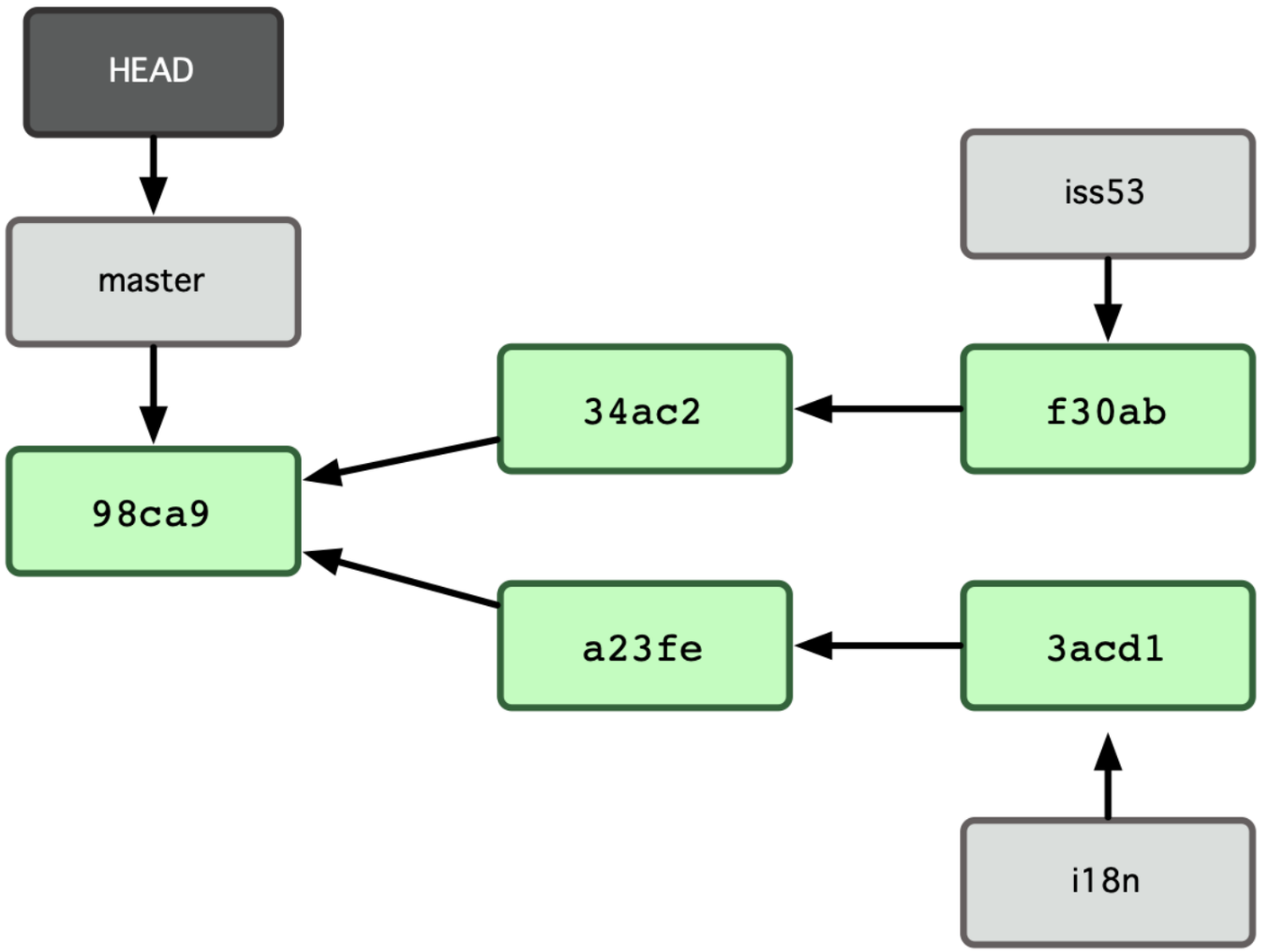
- Due branch possono essere “fusi” usando il comando **merge**
  - Eseguito a partire da un branch “target”
  - Unisce le modifiche presenti nel branch “source”
  - Il risultato è un branch “target” che contiene le modifiche apportate ad entrambi i branch

- Le operazioni di merge possono essere di due tipo
  - **Fast-forward:** I due branch vivono nella stessa linea temporale.
  - **Non Fast-Forward:** I due branch non vivono nella stessa linea temporale

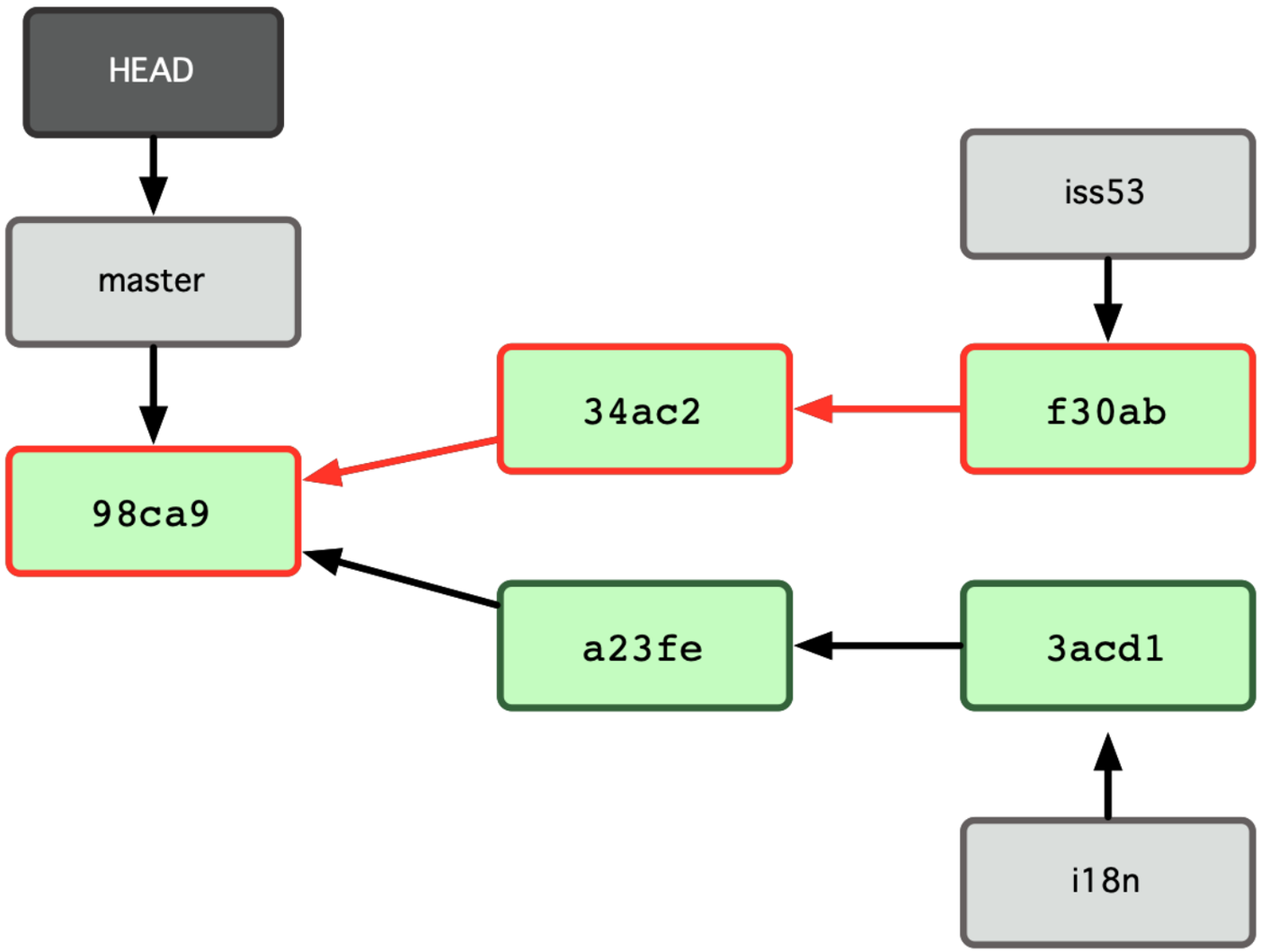
- Non Fast-Forward merge posso provocare dei conflitti che devono essere risolti manualmente.


HEAD
iss53
master
34ac2
f30ab
98ca9
a23fe
3acd1
i18n

```
git checkout master
```

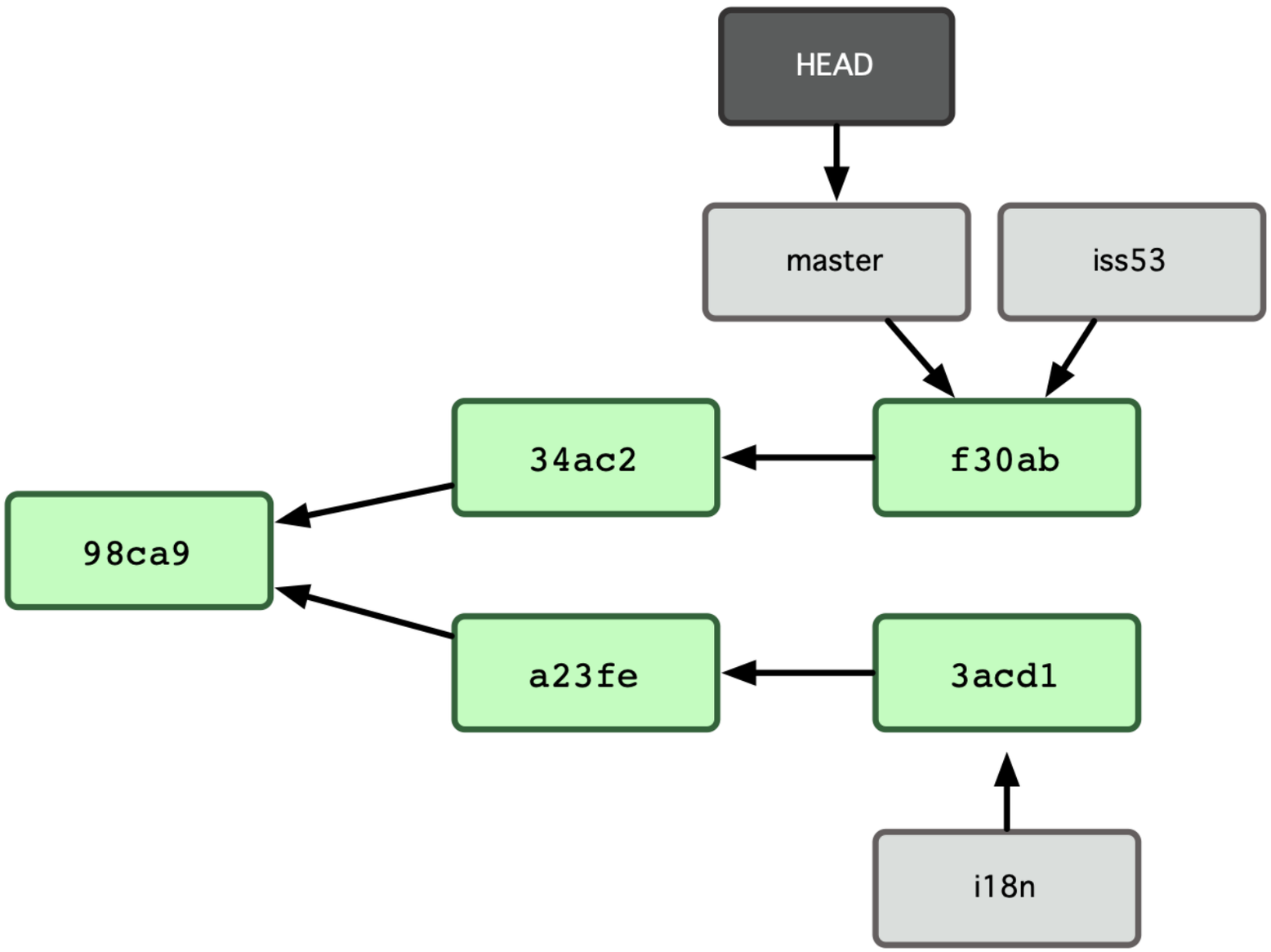


```
git merge iss53
```



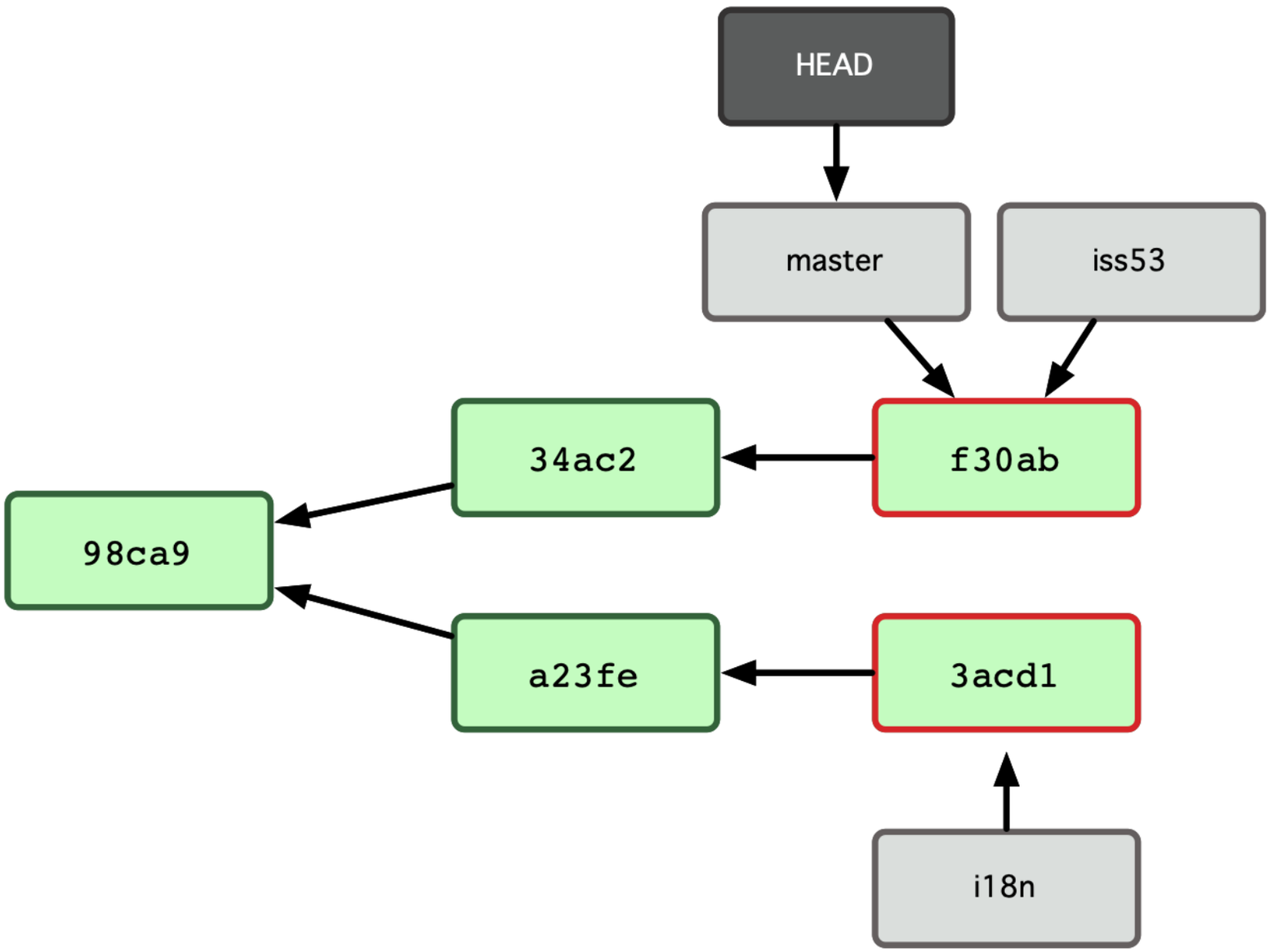
```
git merge iss53
```

fast-forward merge

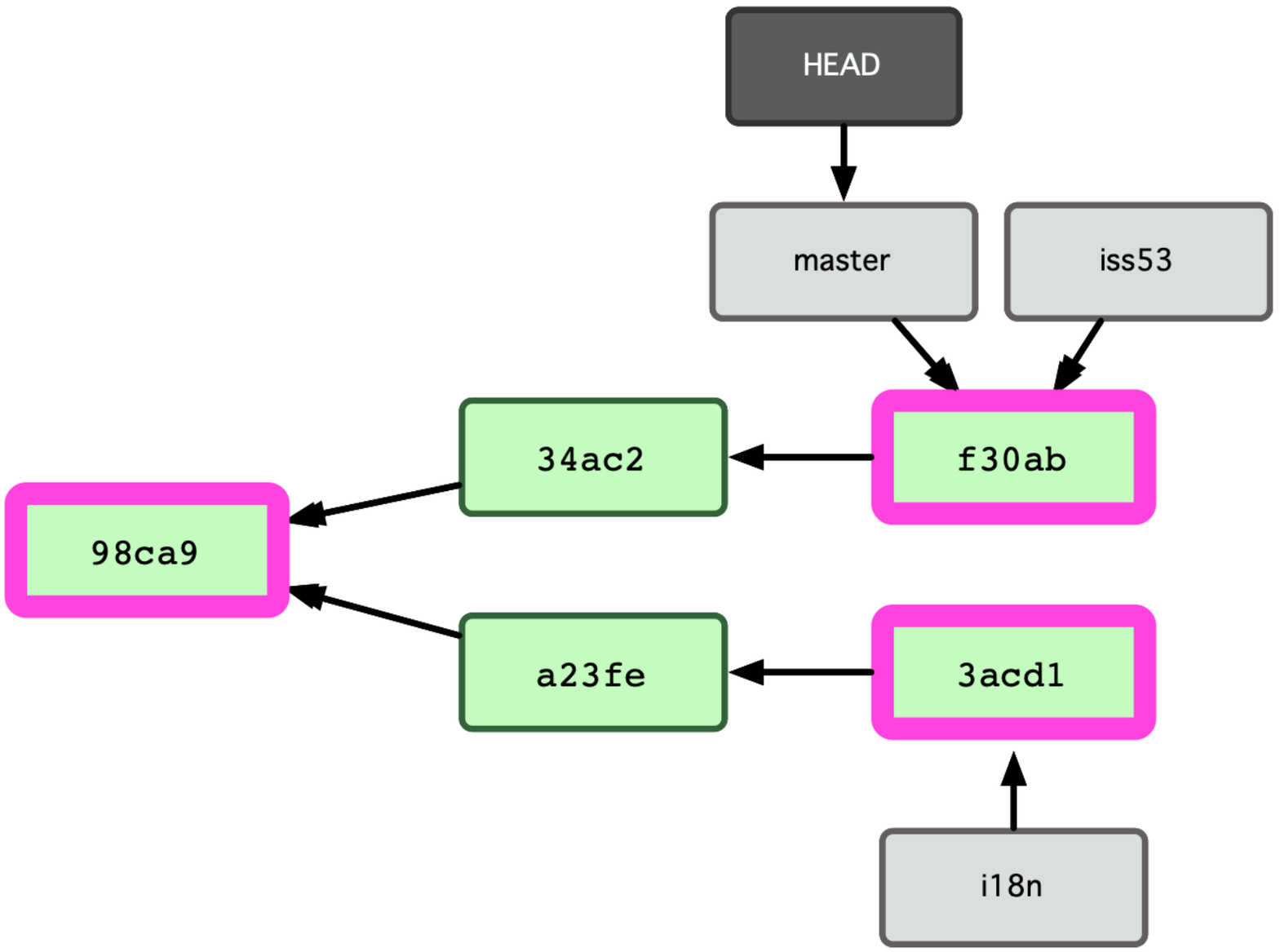


```
git merge i18n
```

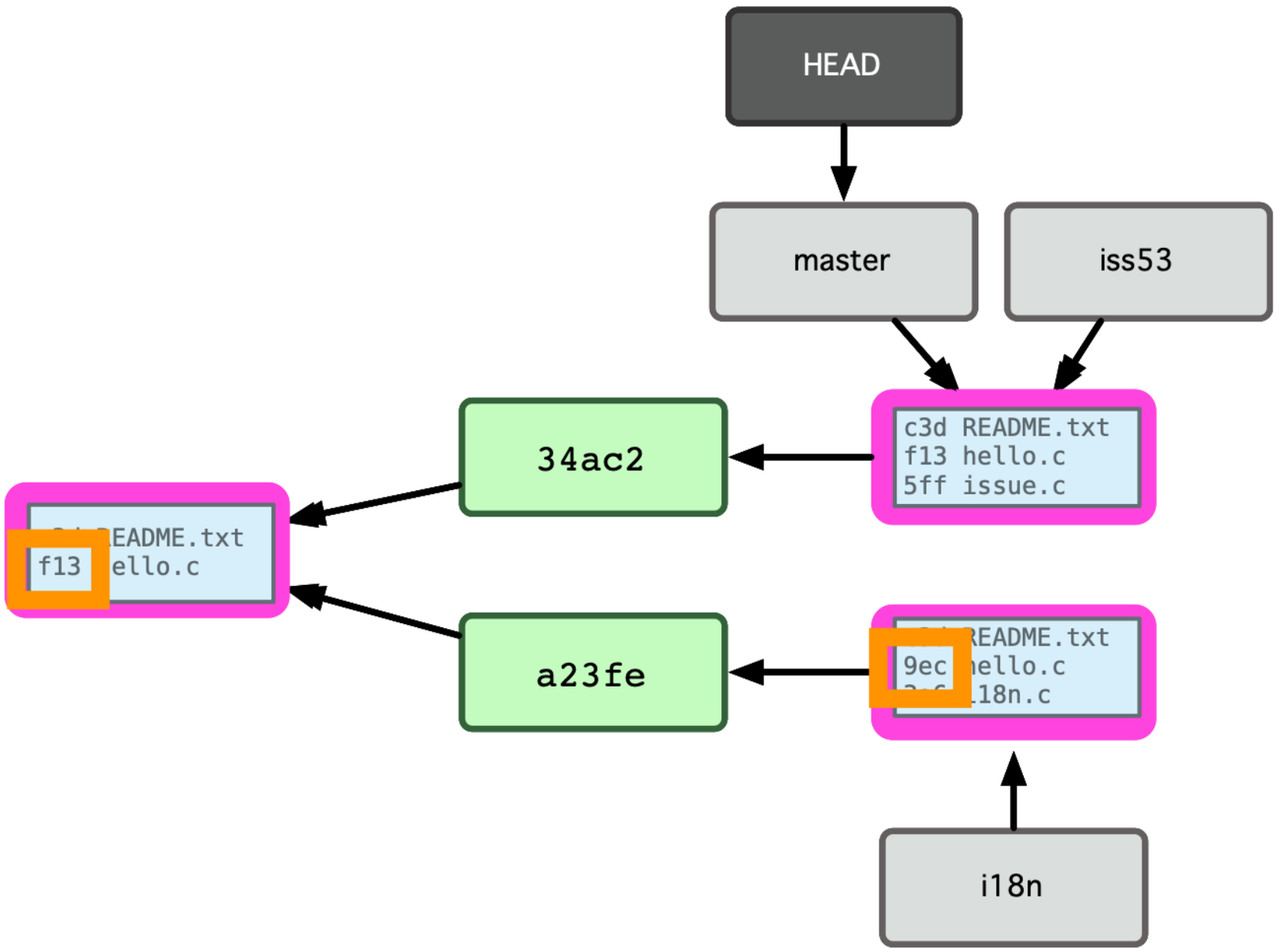
NON fast-forward merge



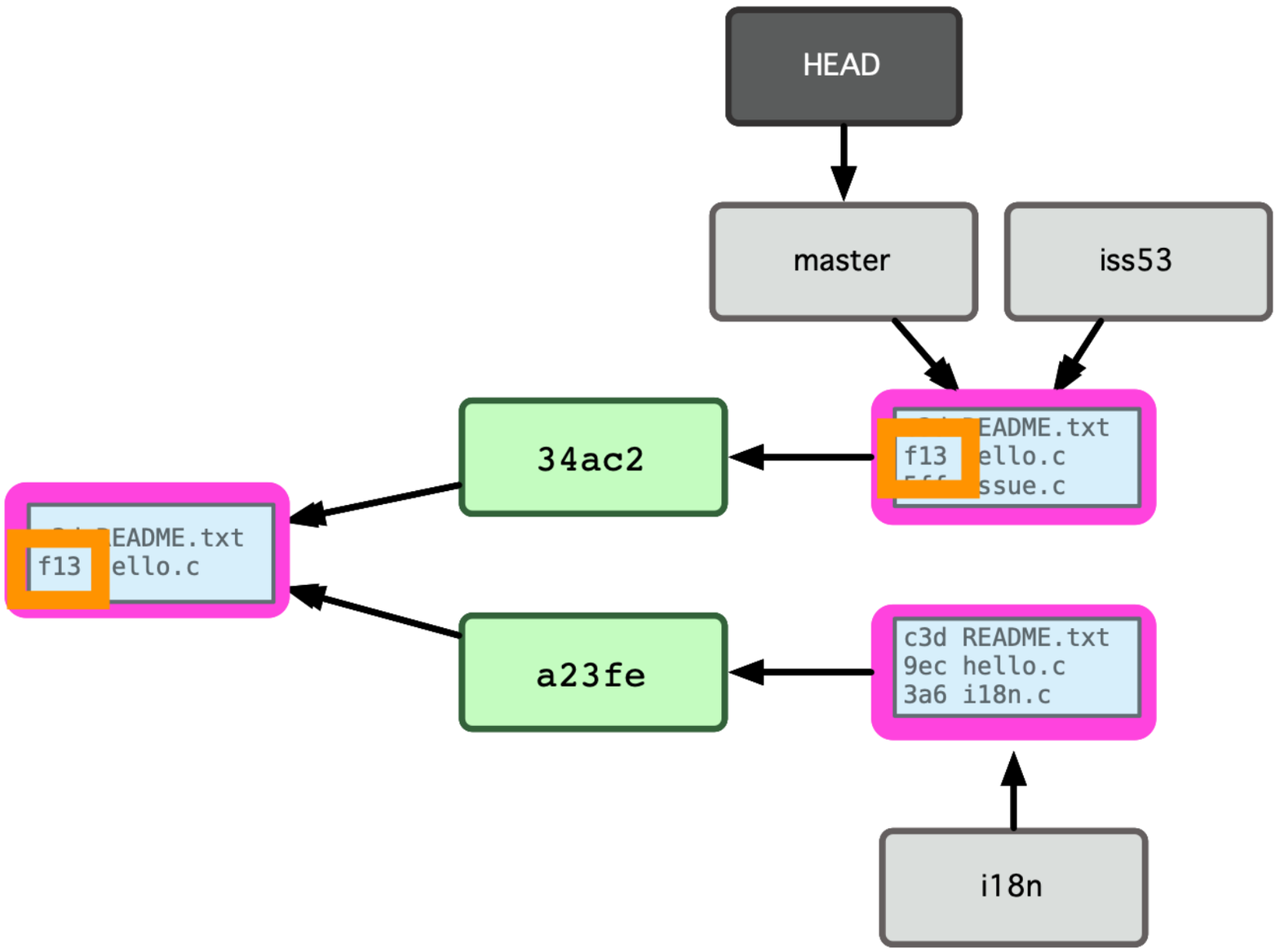
```
git merge i18n
```



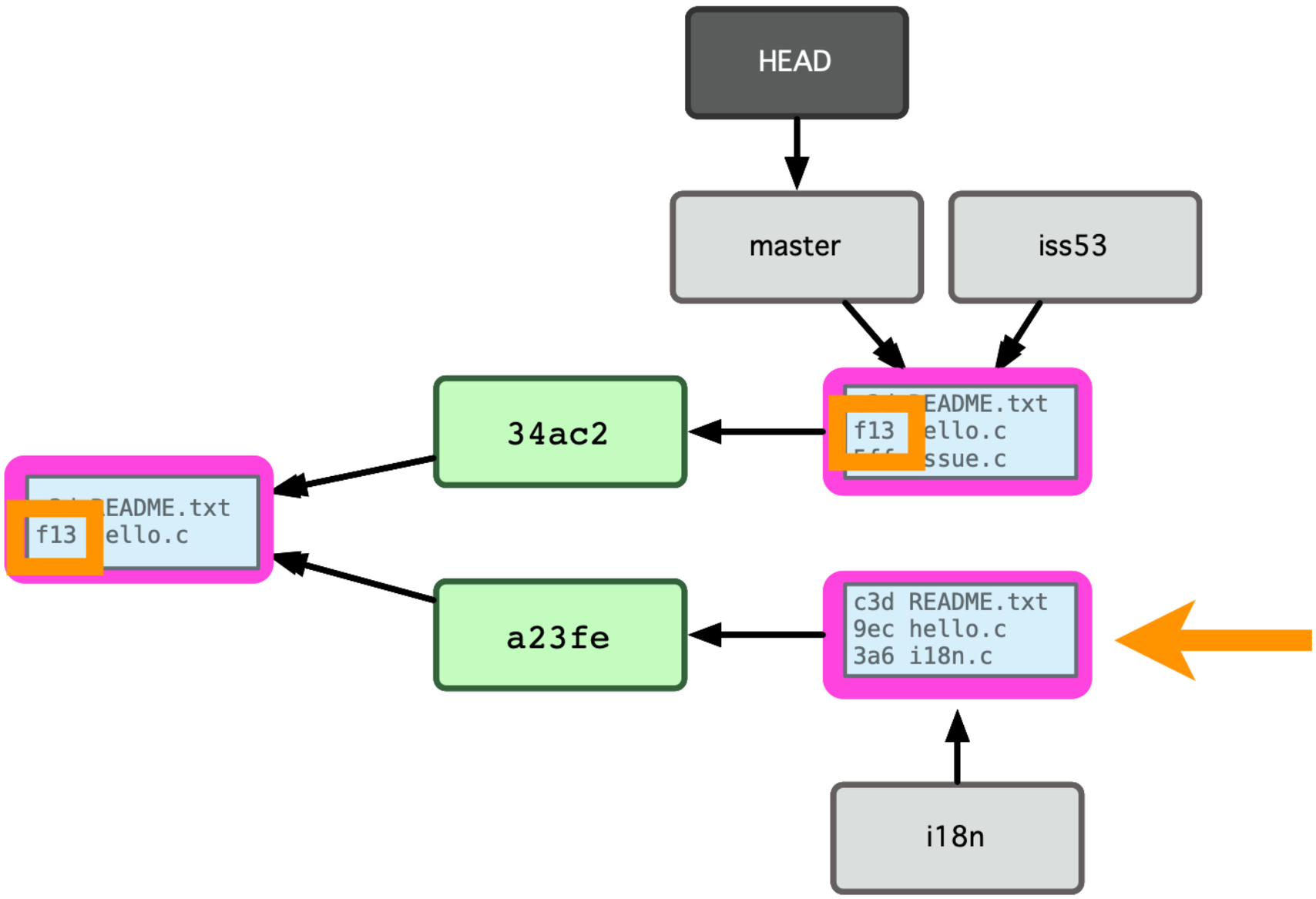
git merge i18n



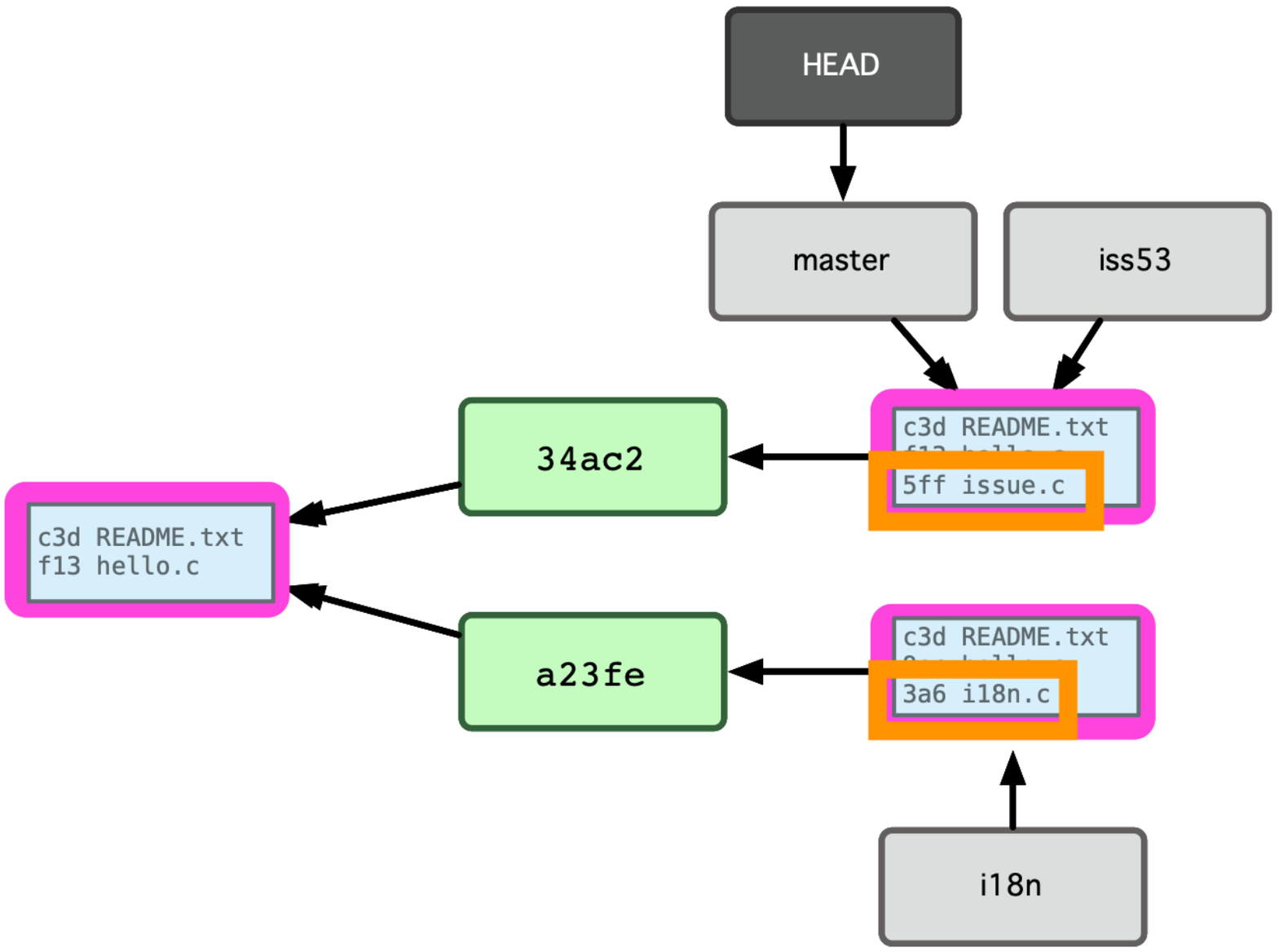
git merge i18n



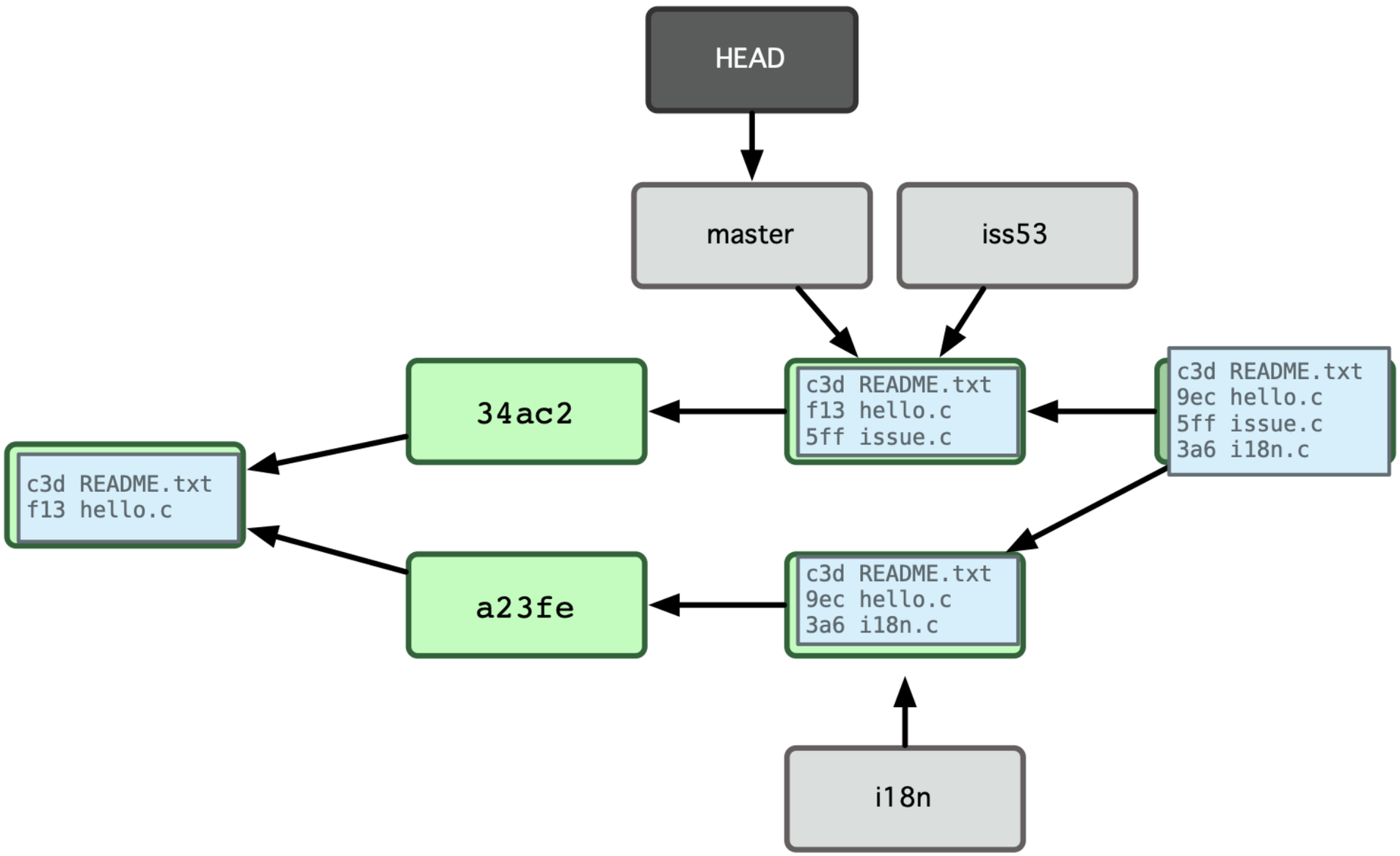
git merge i18n



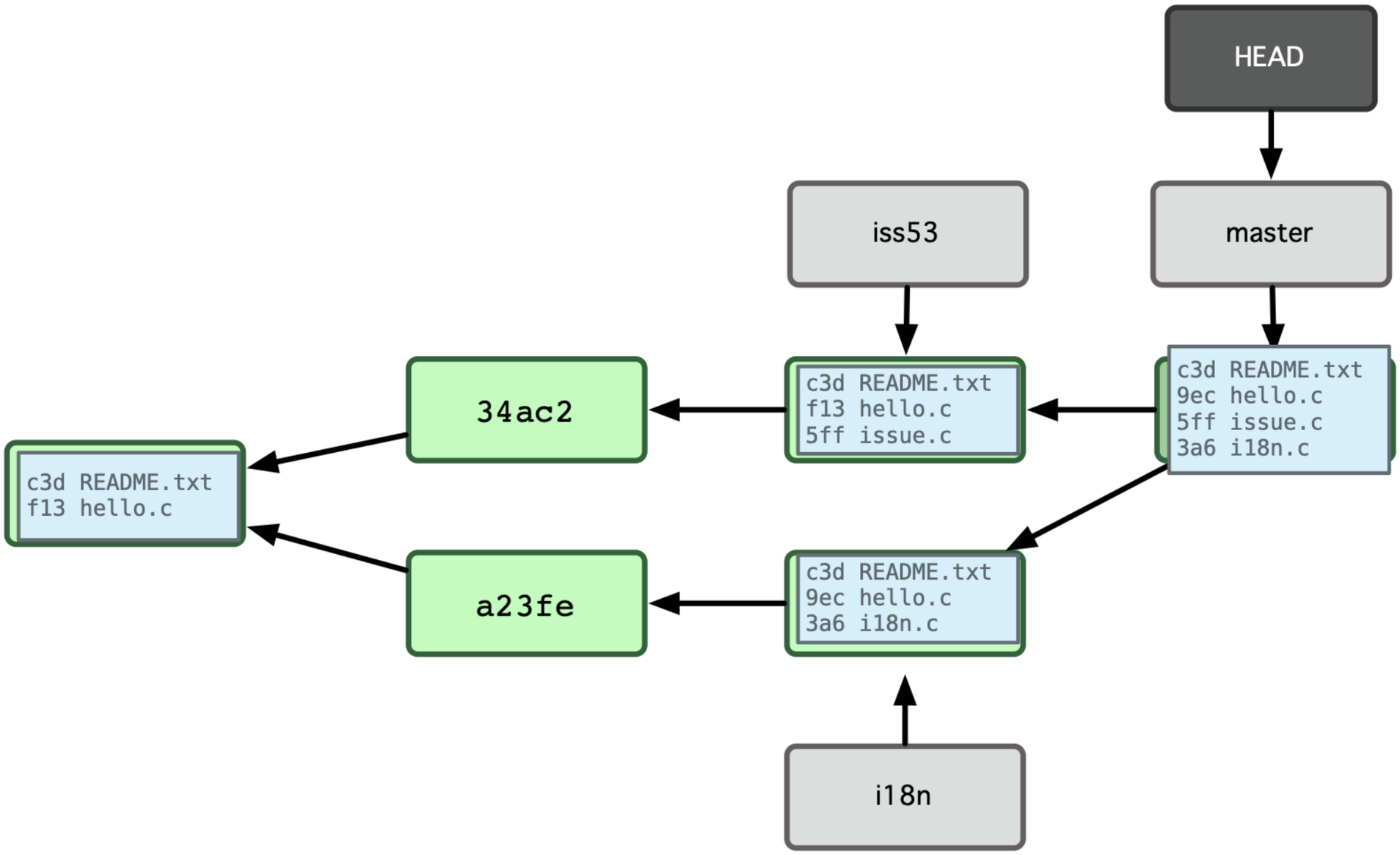
```
git merge i18n
```

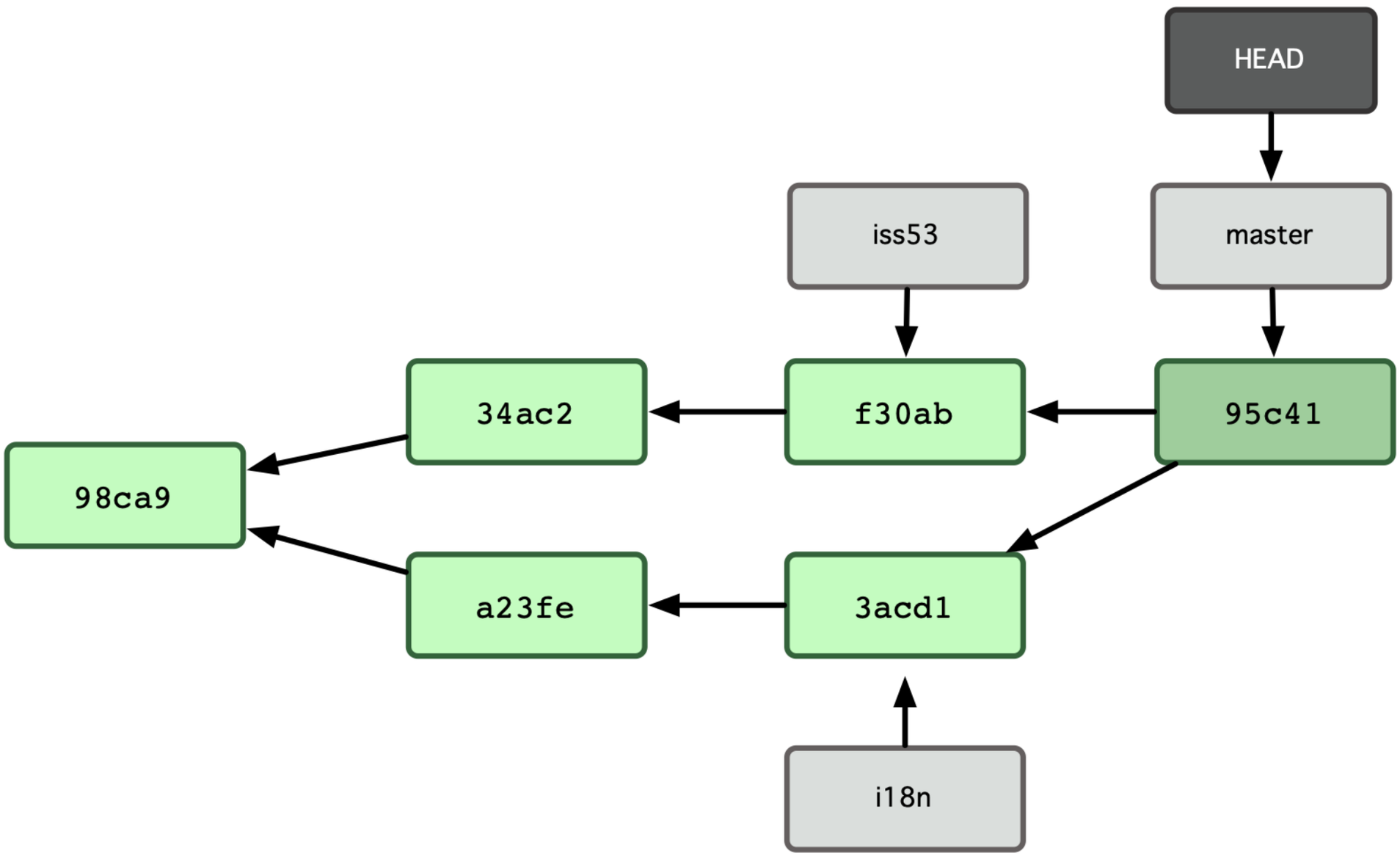


git merge i18n



git merge i18n




HEAD
iss53
master
34ac2
f30ab
95c41
98ca9
a23fe
3acd1
i18n

MERGING

- Nella fase di riconciliazione di due branch possono verificarsi dei conflitti
  - Una stessa riga in uno stesso file è stata modificata in entrambi i branch
  - Git non sa quale sia la versione più recente (perché entrambe lo sono!)

- **ATTENZIONE**: Il problema può verificarsi anche quando se si collabora sullo stesso commit.

```
$ 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 merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.


$ git status
index.html: needs merge
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in
working directory)
#
#  unmerged:   index.html
#
```

```
<<<<<<< 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
```

```
$ git add index.html
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to uns
#
#       modified:   index.html
#
$ git commit
```

DETACHED HEAD


FBI
Google
How to re-attach a detached head
Google Search
I'm Feeling Lucky
FBI
Google
How to re-attach a detached head Git
Google Search
I'm Feeling Lucky

BRANCH HEAD

- L’ultimo commit di un branch è identificato da **`HEAD`**
  - Ogni branch ha la sua **`HEAD`**
  - Da non confondere con il branch **`MASTER`**

- **`HEAD`** è il commit che viene caricate quando facciamo checkout di un branch
  - È il commit che dovremmo modificare quando sviluppiamo con git

- I riferimenti alle **`HEAD`** dei vari branch sono salvate in file
  - **`/.git/logs/HEAD`** la head del branch corrente
  - **`/.git/logs/refs/heads`** tutte le head

DETACHED HEAD

- Lo sviluppo del codice dovrebbe procedere utilizzando sempre la **`HEAD`** del branch
  - Git si aspetta di ricevere modifiche riguardanti questo commit

- Se facciamo checkout di commit passati, git ci avvertirà che ci troviamo in uno stato **`detached head`**

- Nello stato **`detached head`** git non è in grado di associare eventuali cambiamenti ad una specifica linea temporale.

MODIFICHE DA DETACHED HEAD

- Modificare una **`detached head`** è un po' come riscrivere il

- Se le modifiche non sono rese stabili (**`commit`**) e facciamo checkout di un altro commit, le nostre modifiche sono perse

- Se le modifiche non sono rese stabili (**`commit`**), il commit viene creato ma non è associato ad alcun branch
  - Un file che rappresenta il commit è aggiunto alla cartella **`./git`**
  - Il commit è «perso» nella storia dello sviluppo

- Possiamo creare un branch apposito che punta al commit

```
$ git branch <new-branch-name> <commit-id>
```

DETACHED HEAD -- CONSIGLI PRATICI

- In generale dovremmo evitare di utilizzare **`detached heads`**
  - Il rischio di perdere i nostri cambiamenti è elevato
  - Commit nascosti rendono complesso ricostruire la storia dello sviluppo

- Se intendiamo costruire uno stato precedente del codice, dovremmo creare anticipatamente un nuovo branch

```
$ git checkout -b <new-branch-name> <commit-id>
```

- Se siamo in una **`detached heads`** e intendiamo modificare il codice dobbiamo creare un branch per il commit corrente

```
$ git branch <new-branch-name> <commit-id>
```

REMOTES

REMOTES

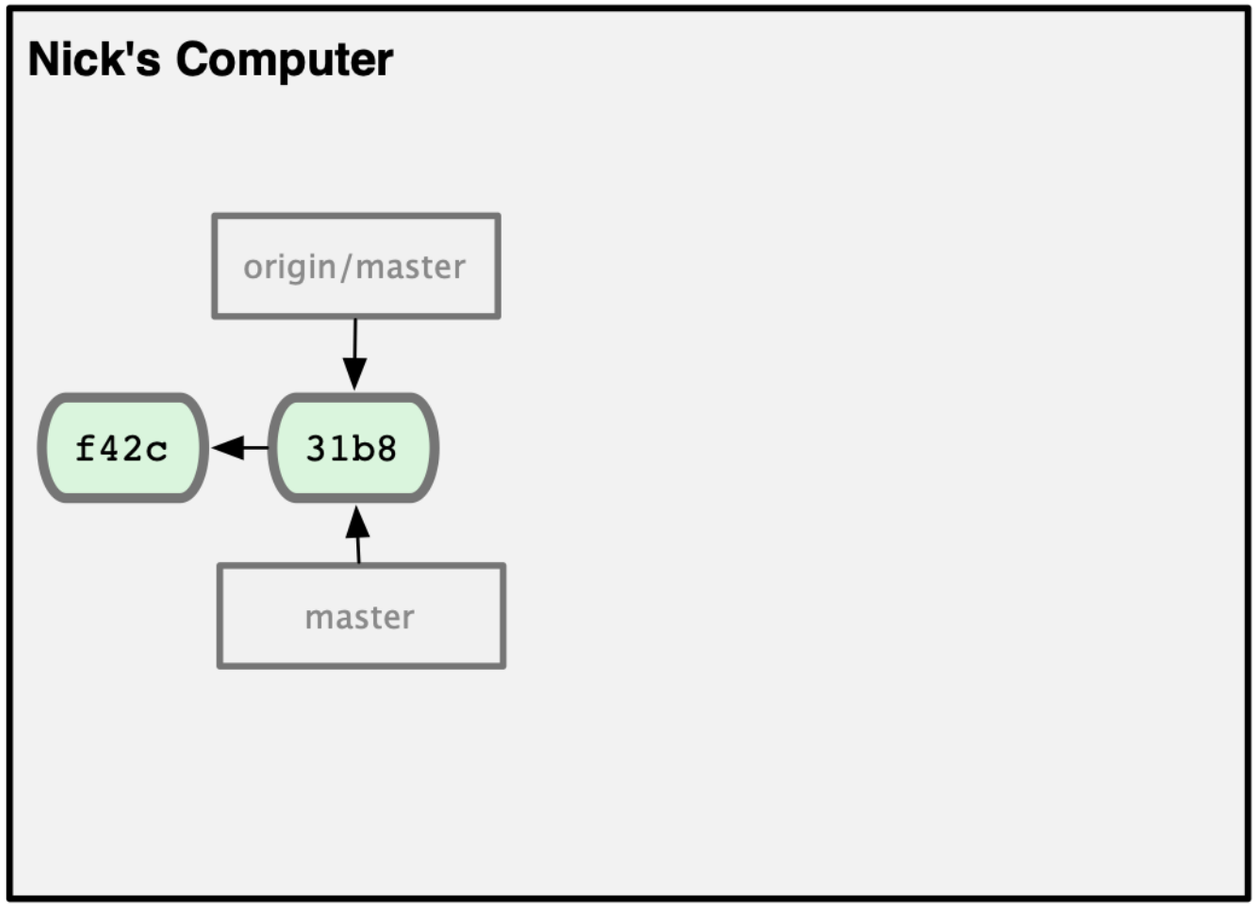
- Un remote rappresenta un repository esterno
  - Git è un VCS distribuito

- È possibile definire collegamenti tra branch locali e remoti
  - Creiamo un collegamento fra le copie locali e la repository remota

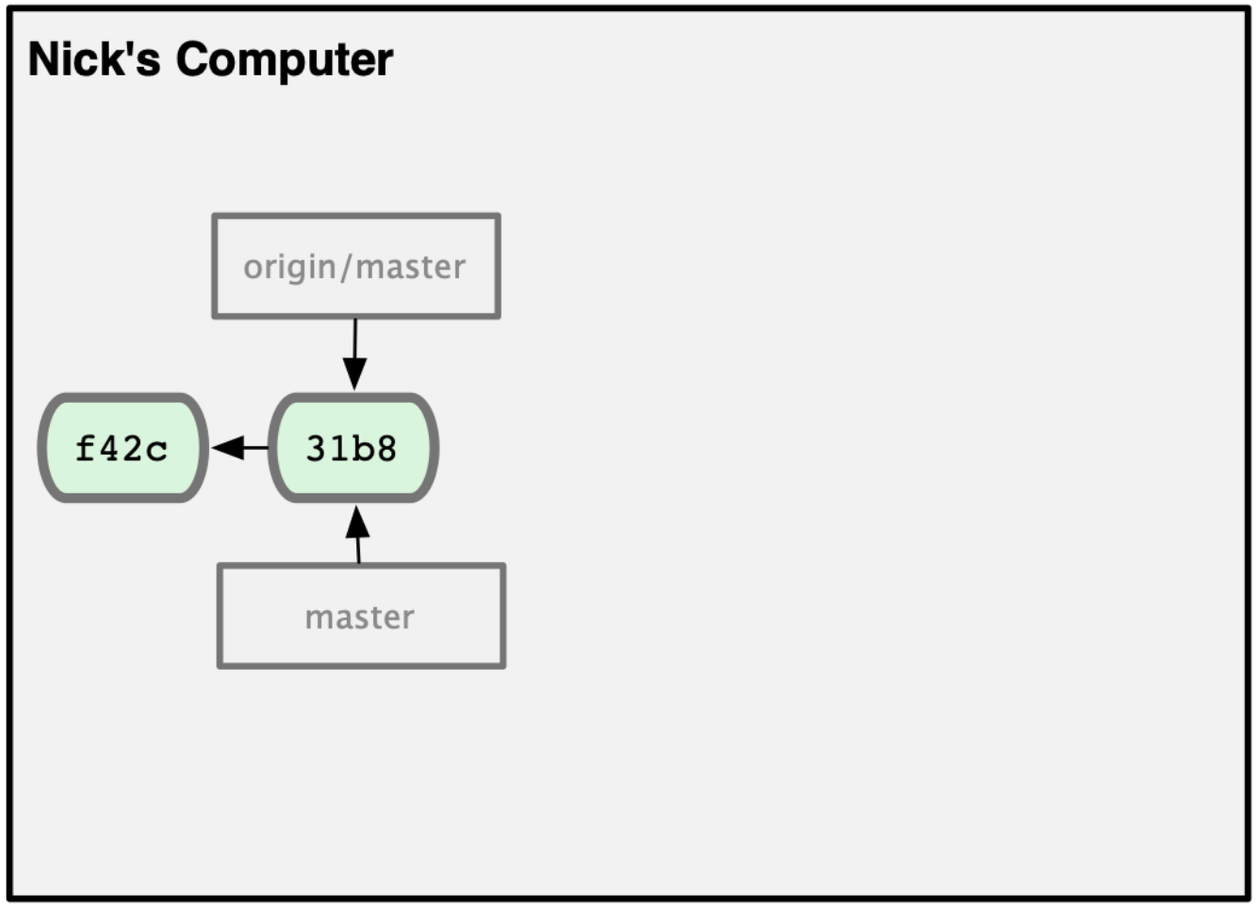
- I branch possono poi essere sincronizzati
  - Con l'operazione di **`push`**

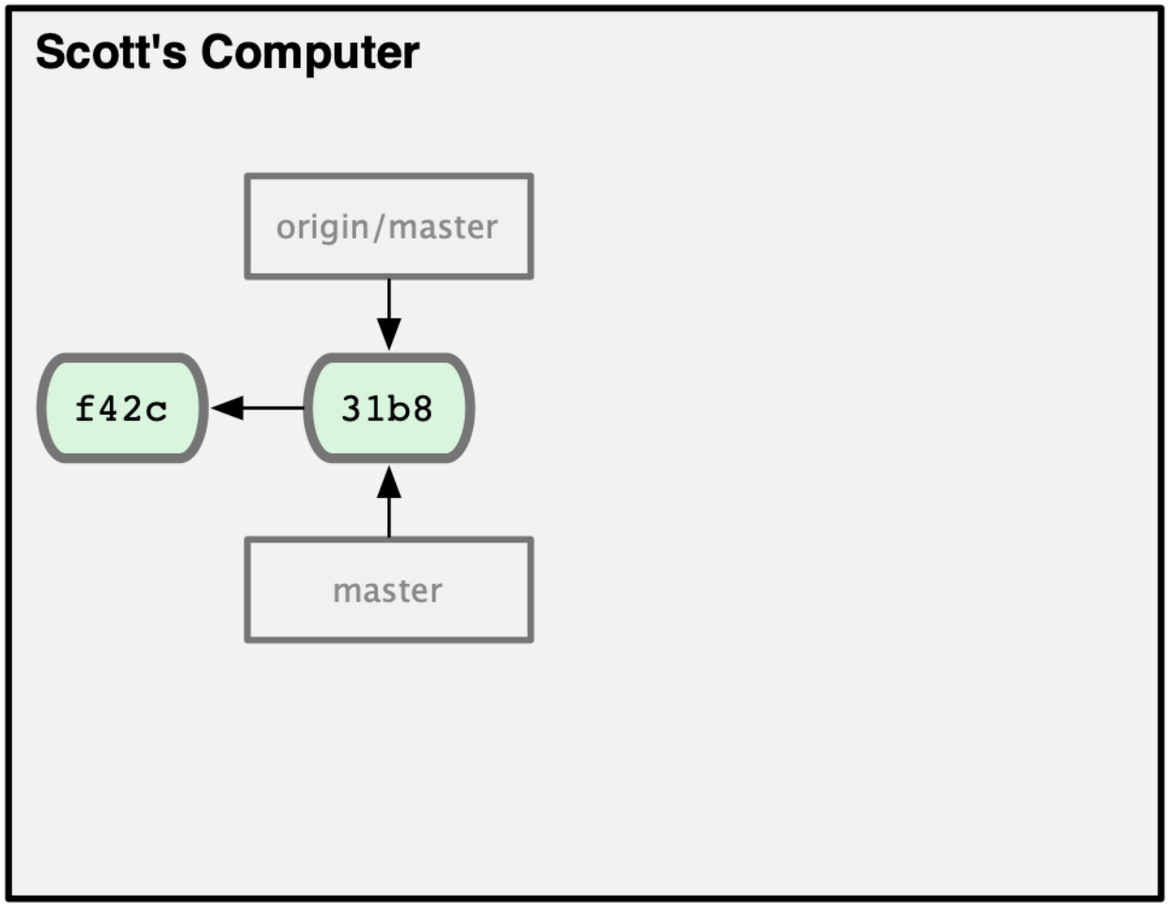

git.ourcompany.com
master
f42c
31b8


```
git clone nick@git.ourcompany.com:project.git
```



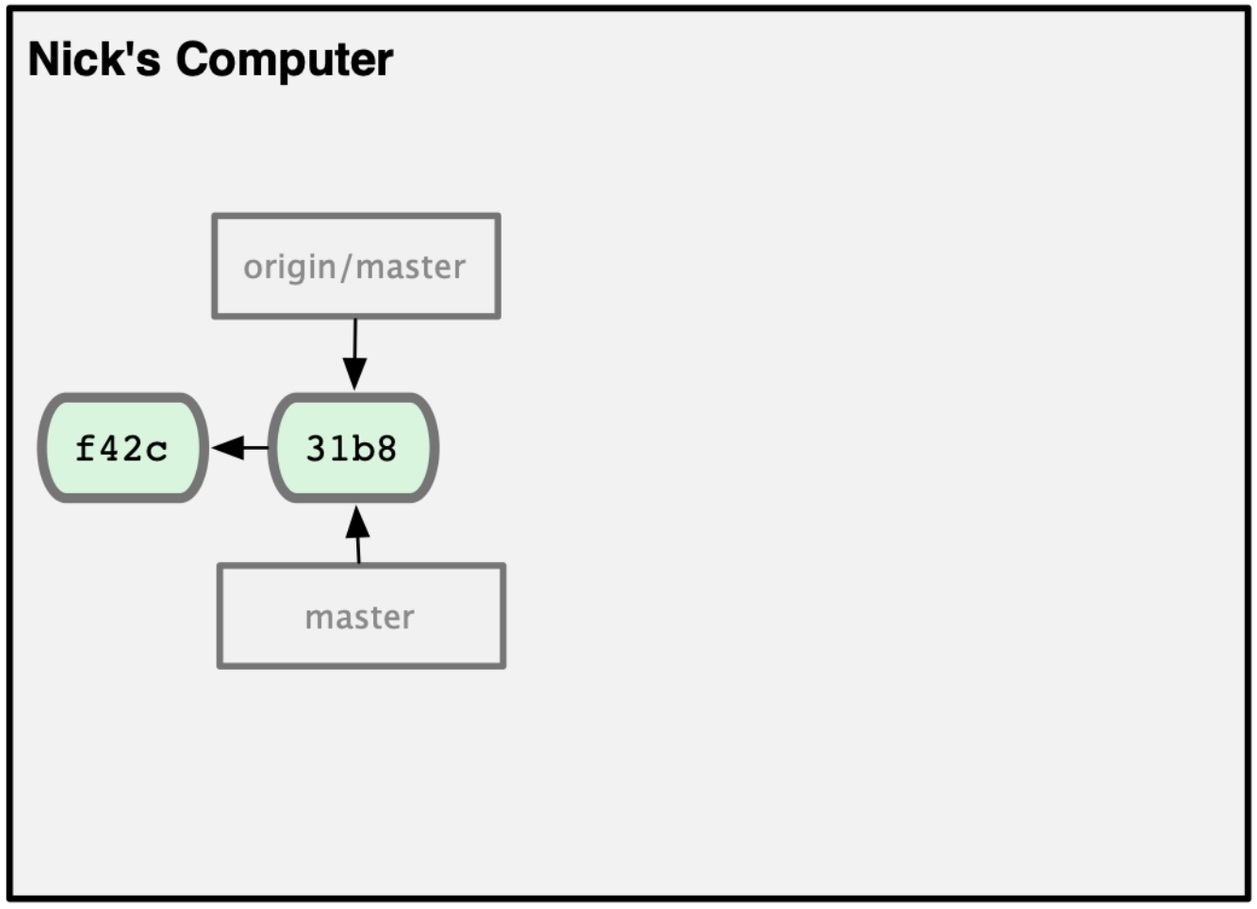

```
git clone scott@git.ourcompany.com:project.git
```

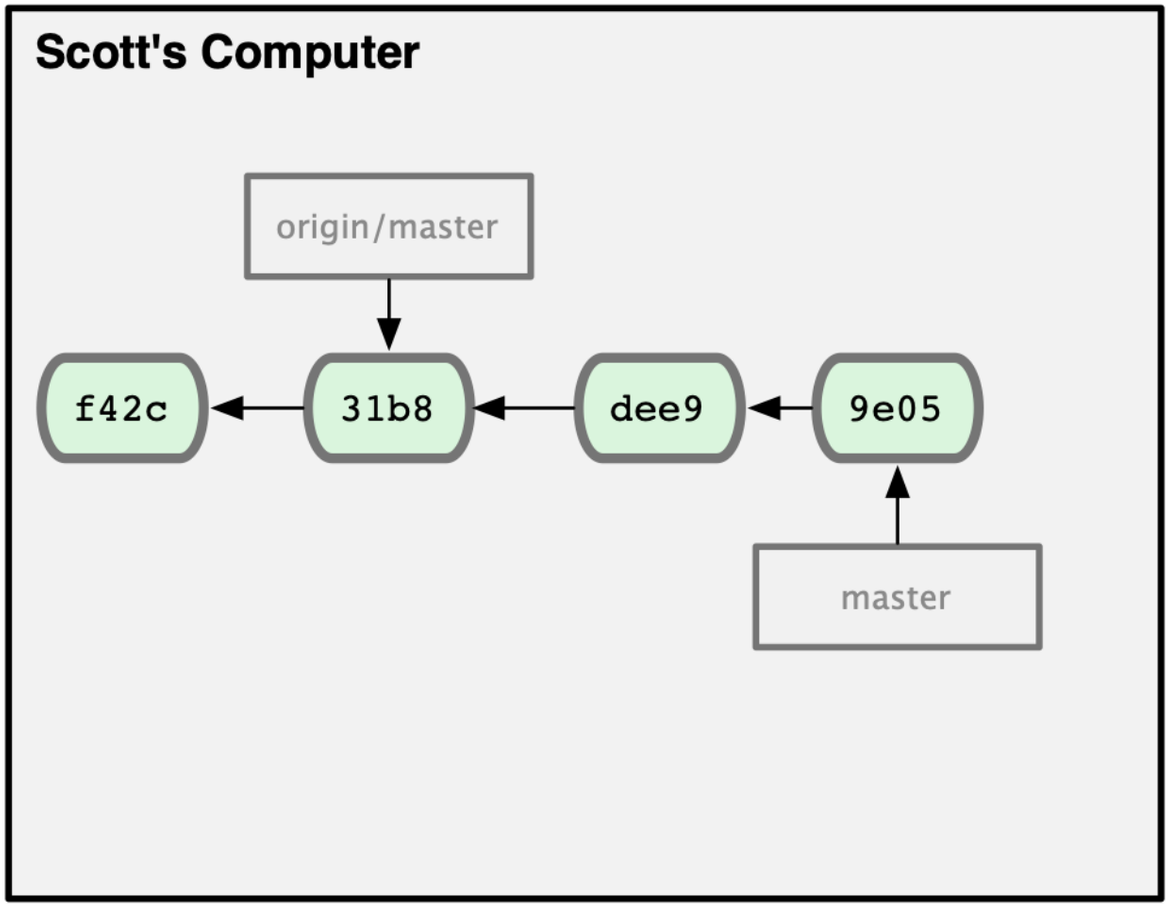




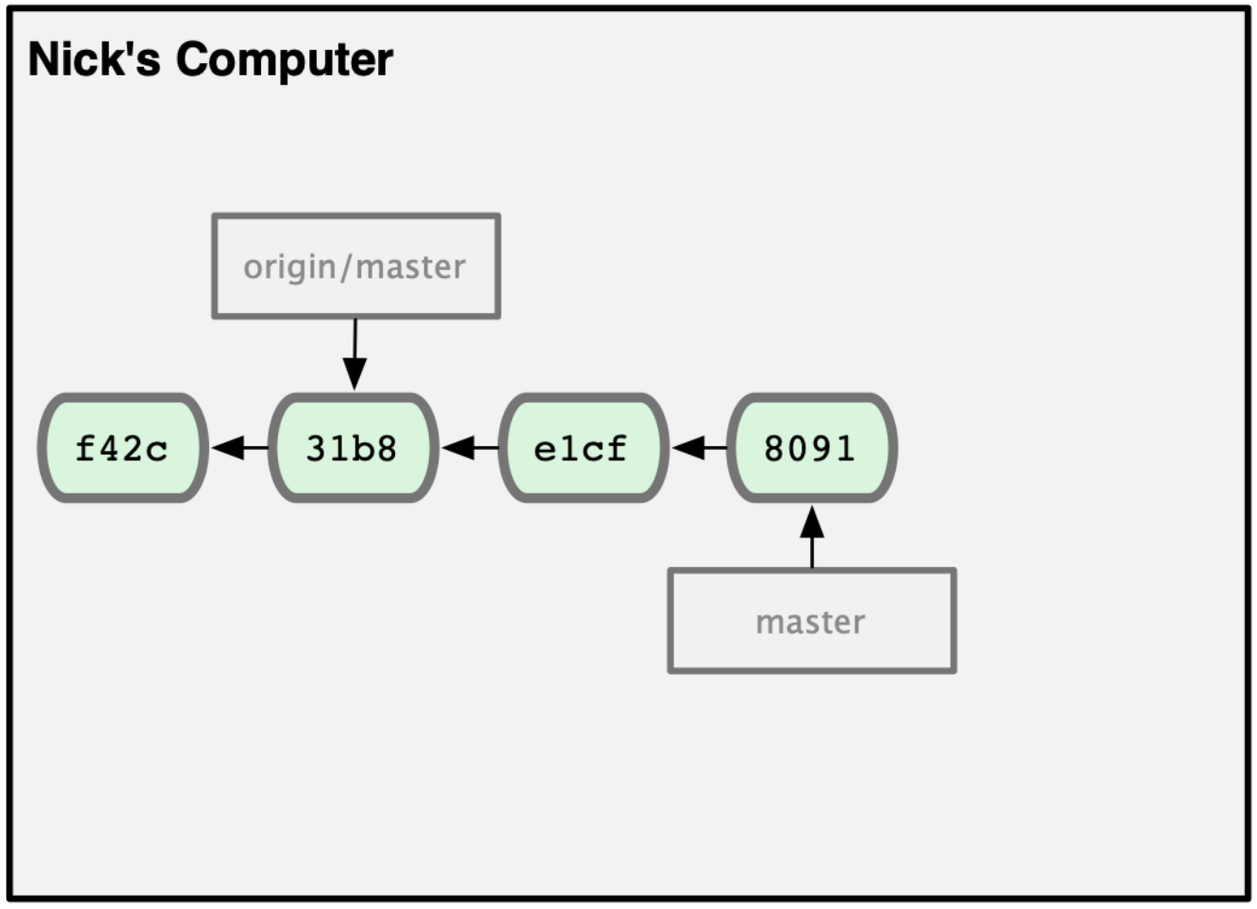

git.ourcompany.com
master
f42c
31b8

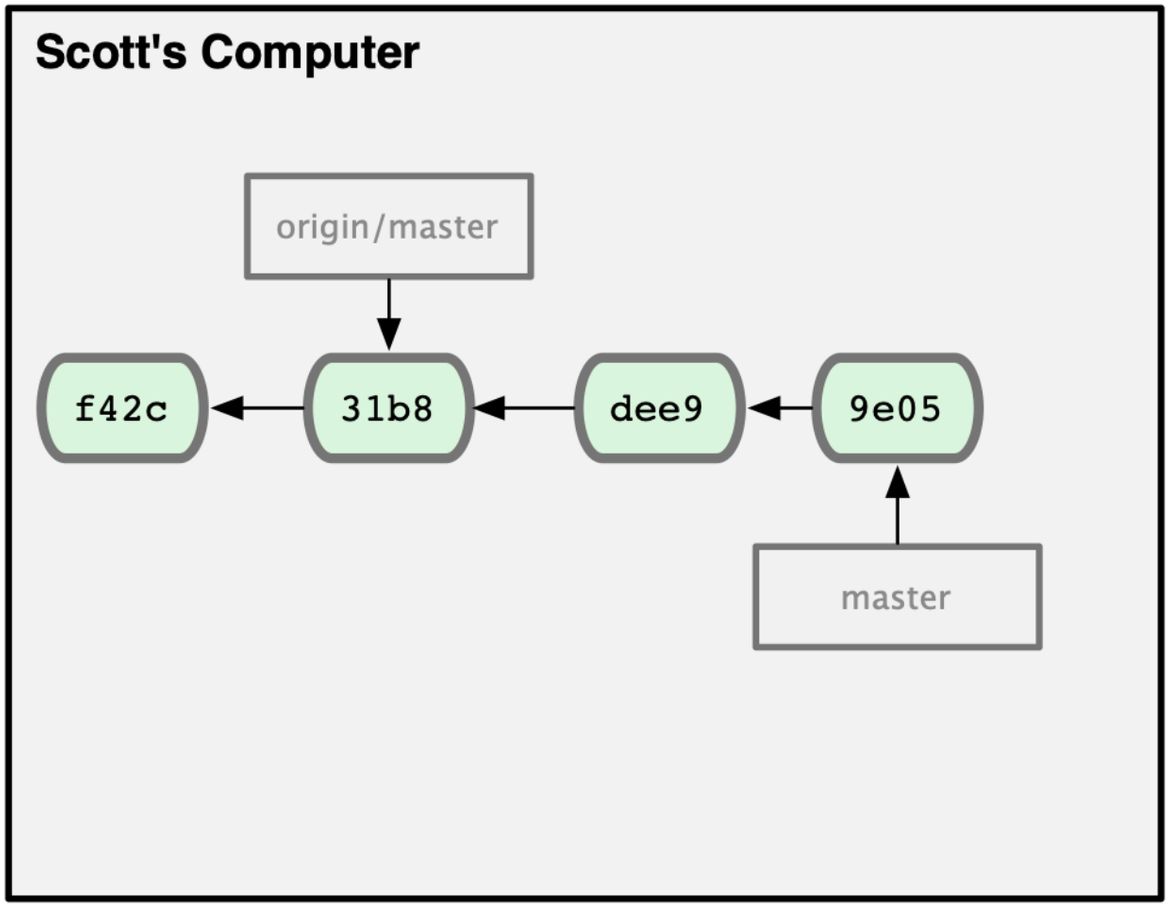
`git commit`


Nick's Computer
origin/master
f42c
31b8
master

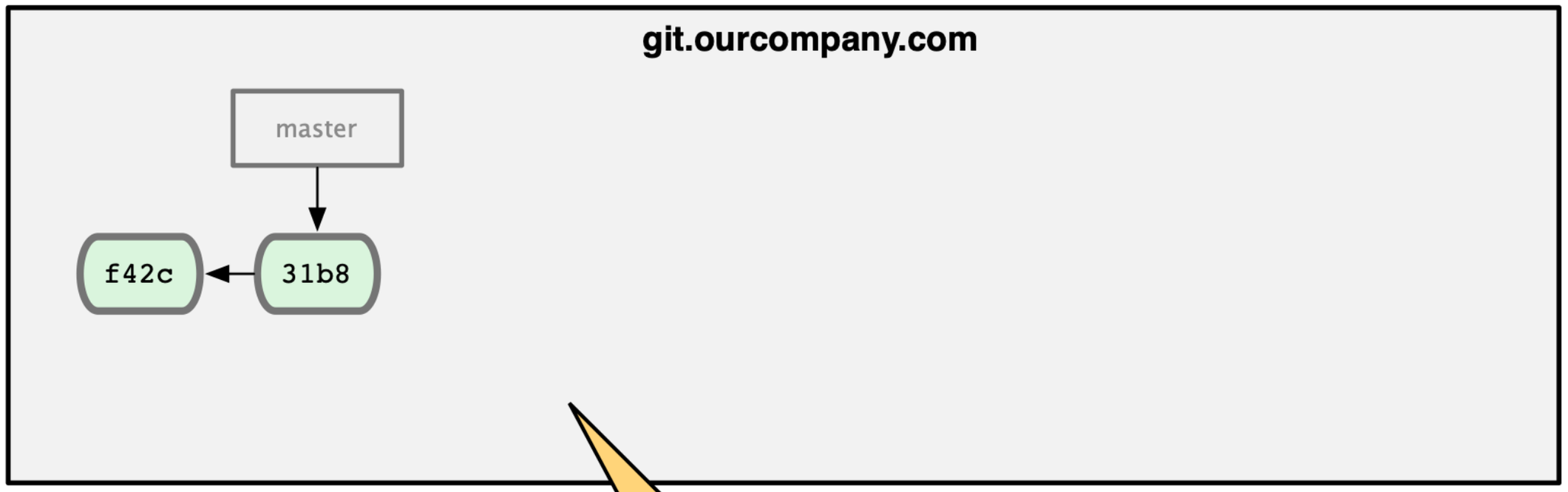

Scott's Computer
origin/master
f42c
31b8
dee9
9e05
master


```
git push origin master
```

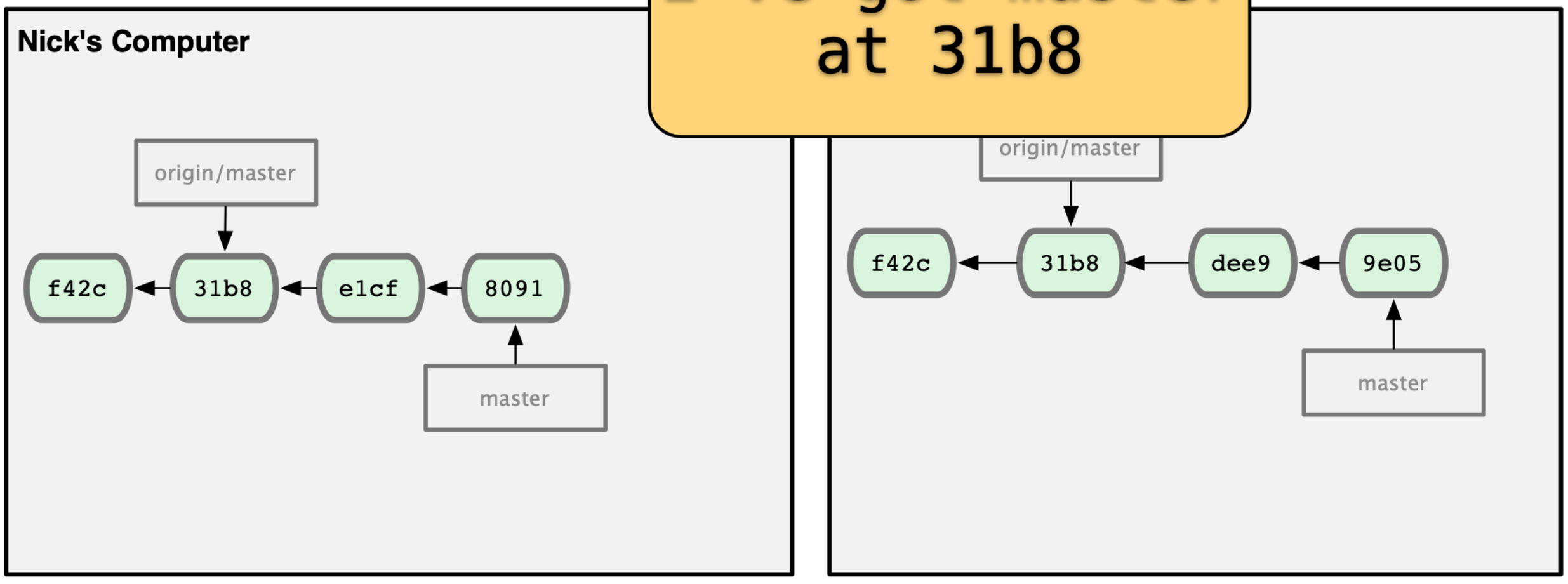





git.ourcompany.com
master
f42c
31b8
i want to push
some new stuff
git p h origin master
Nick's Computer
origin/master
f42c
31b8
e1cf
8091
master
Scott's Computer
origin/master
f42c
31b8
dee9
9e05
master

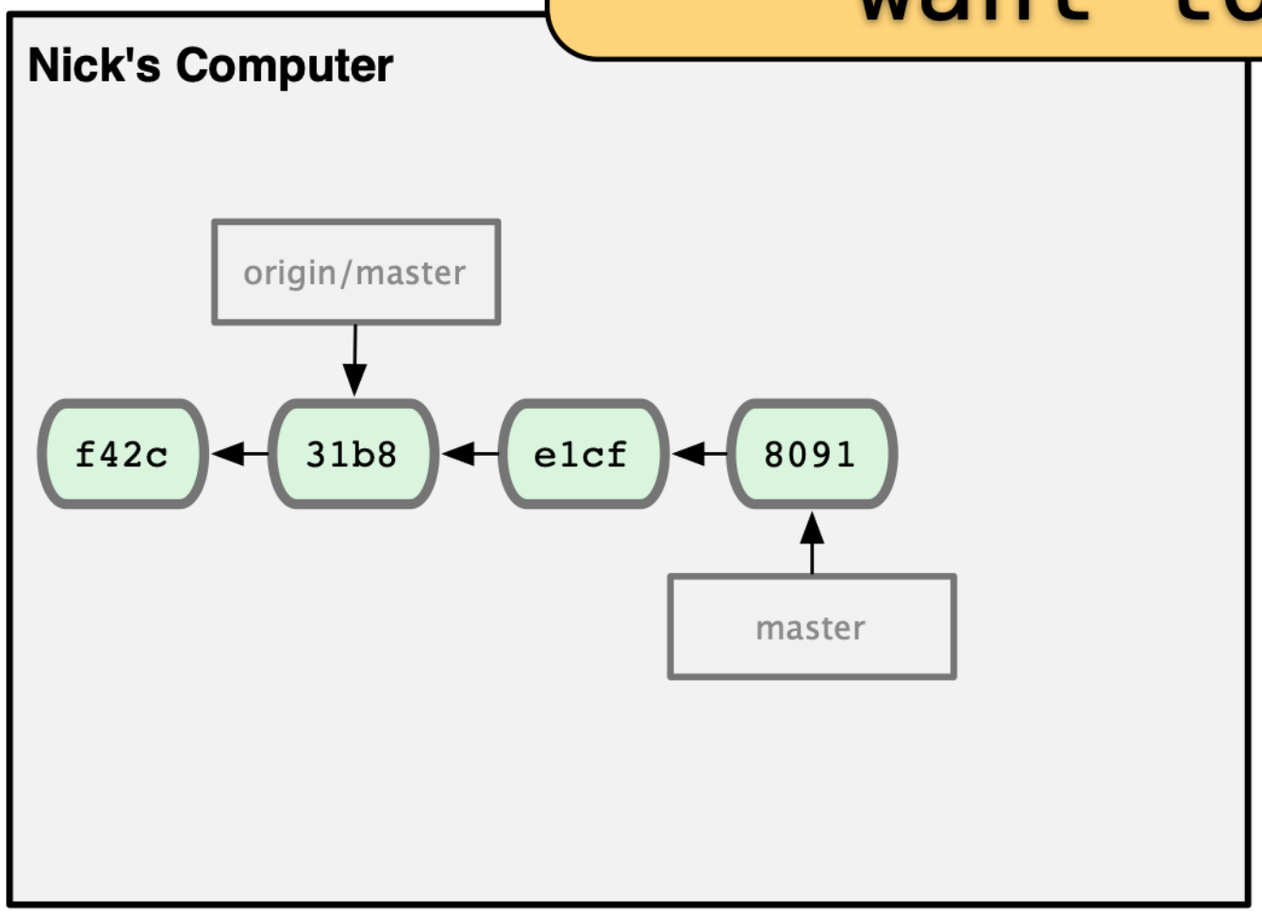


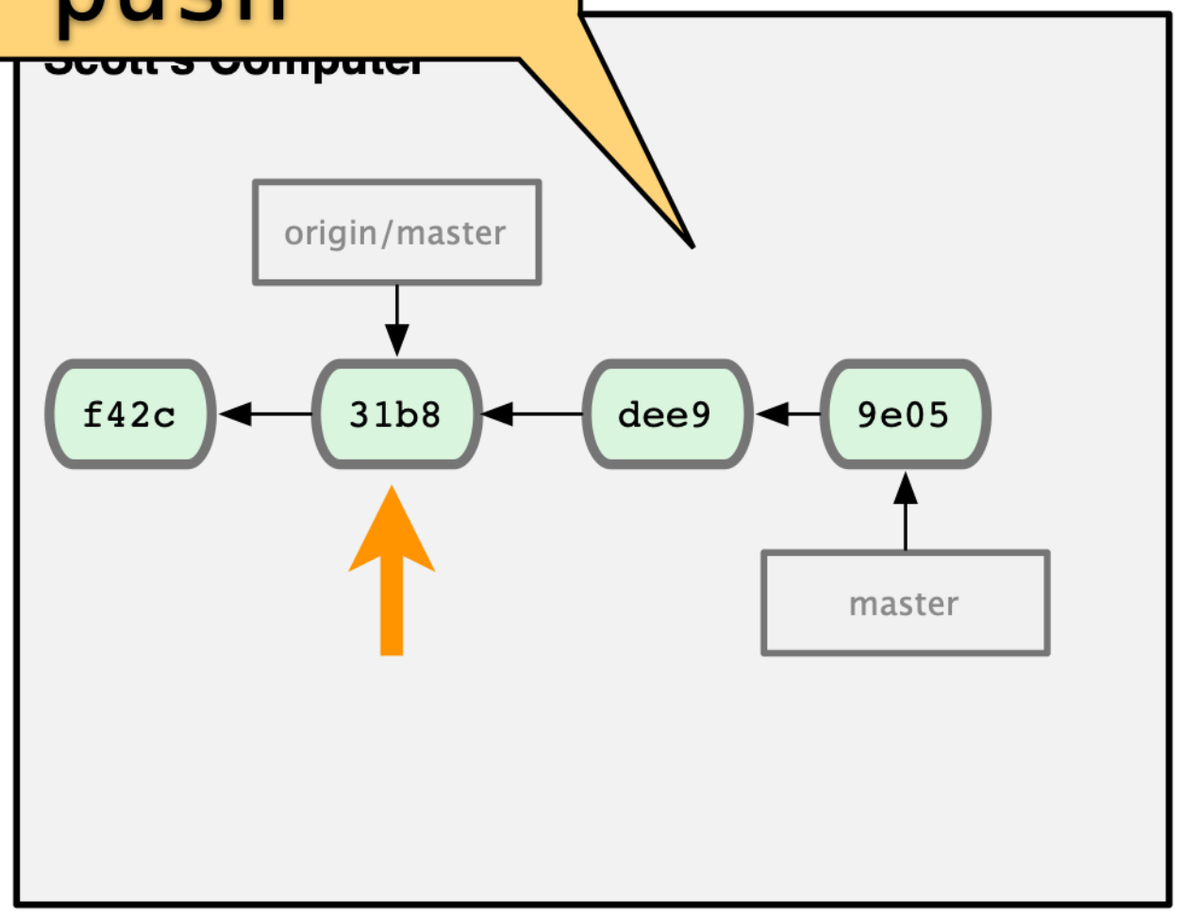
i've got master at 31b8



cool, i see that in the history of what i want to push

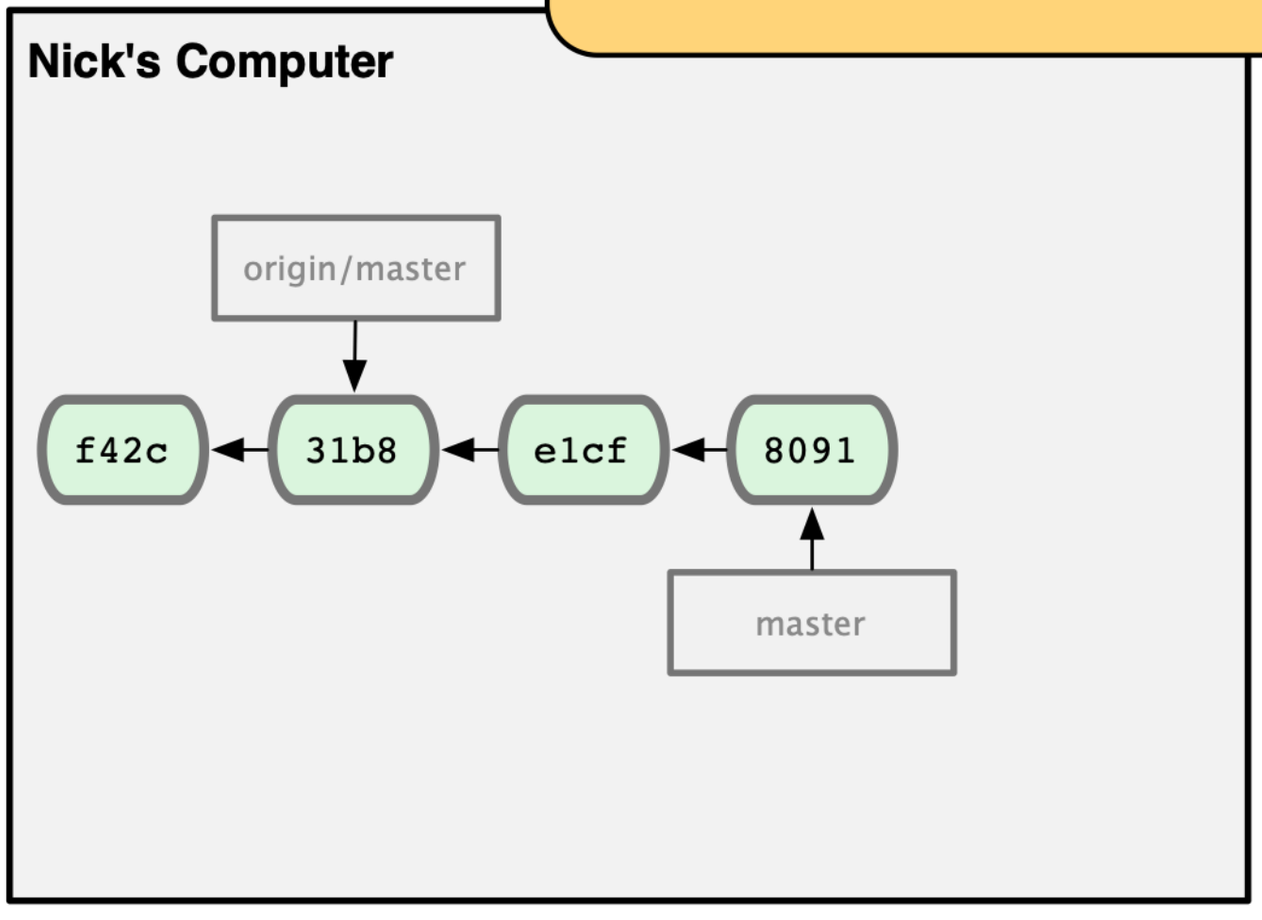
igin master

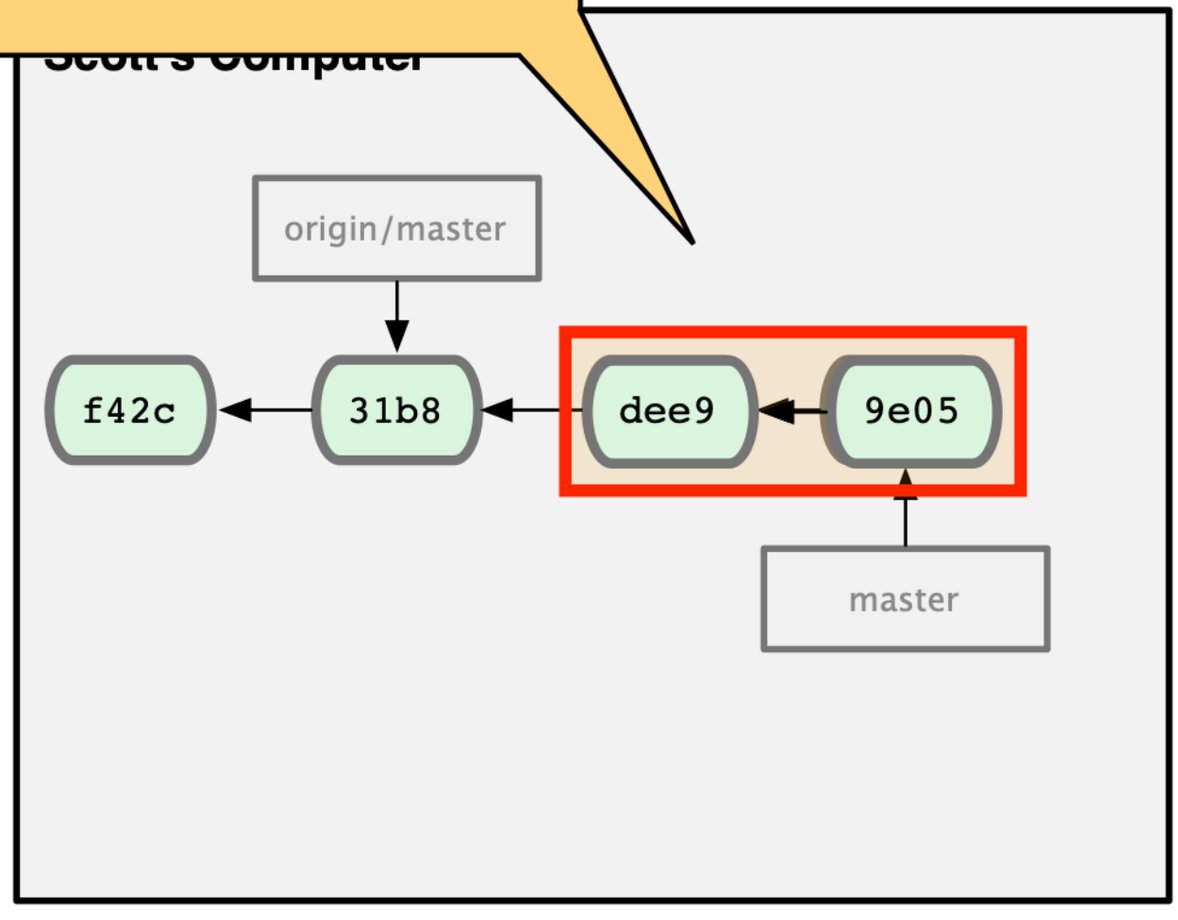


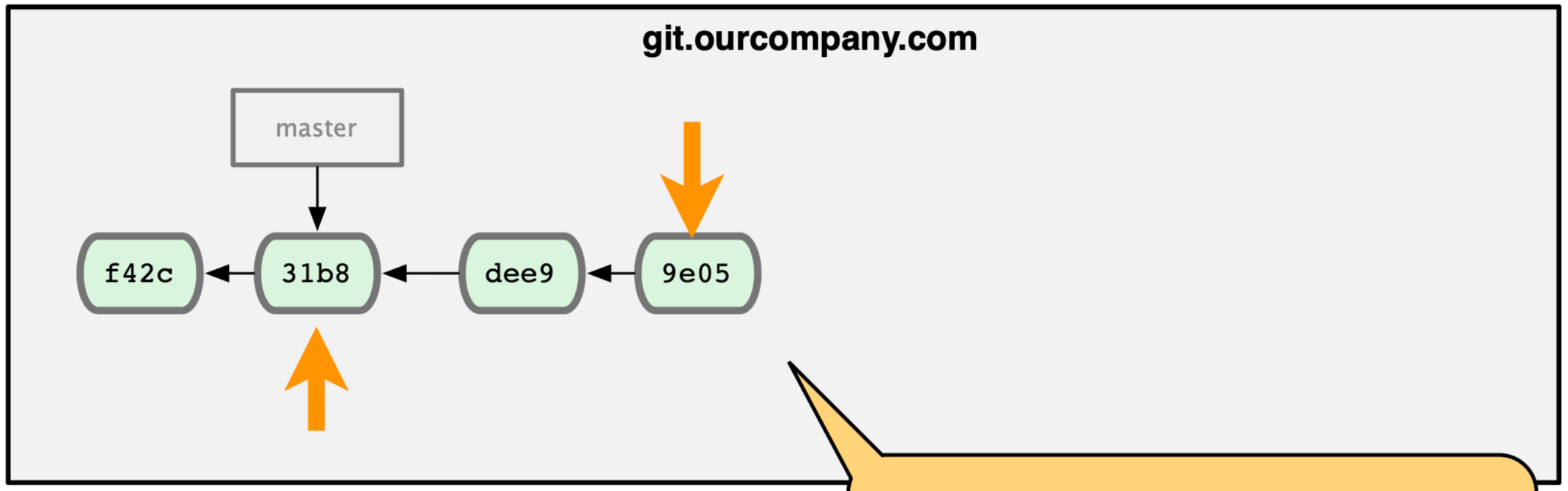


here's the difference

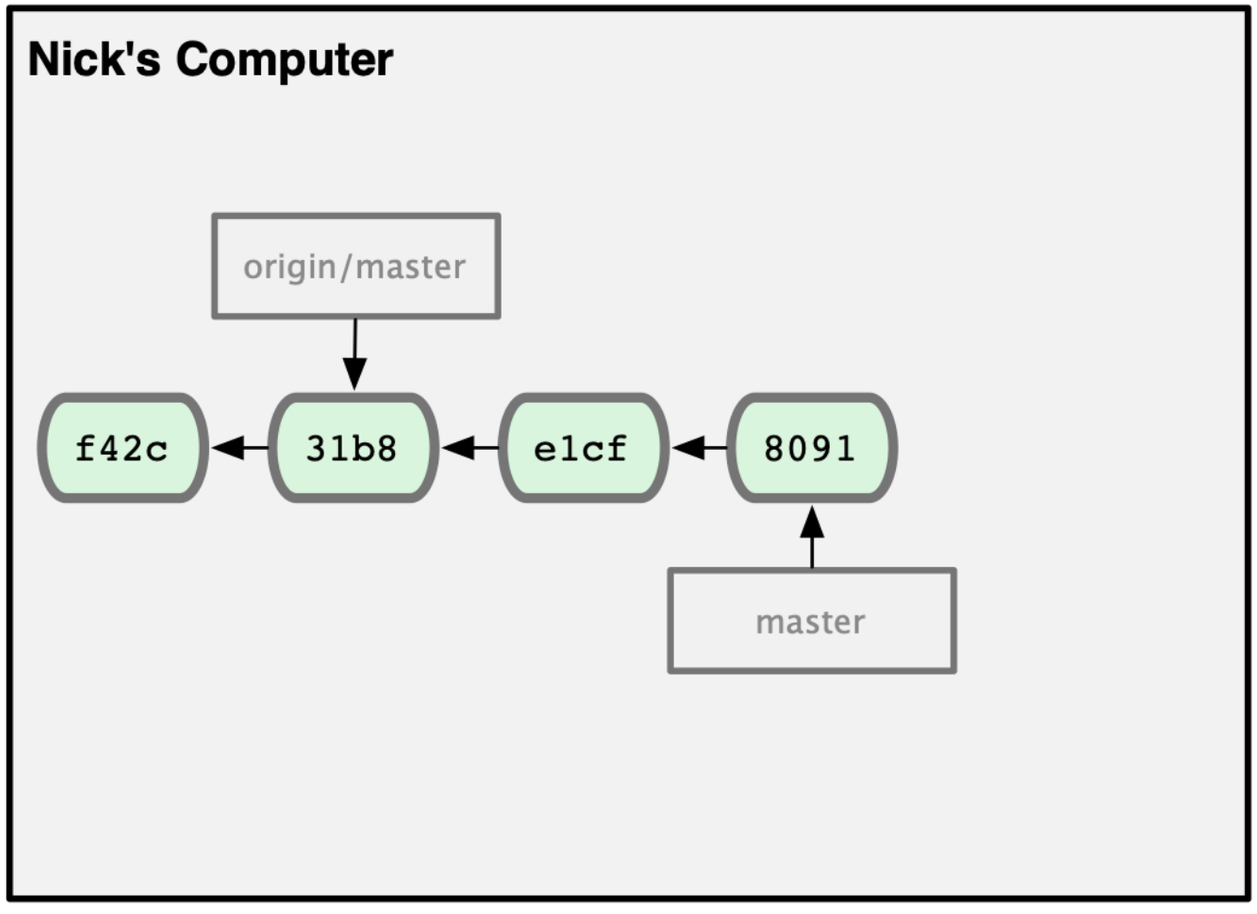
igin master

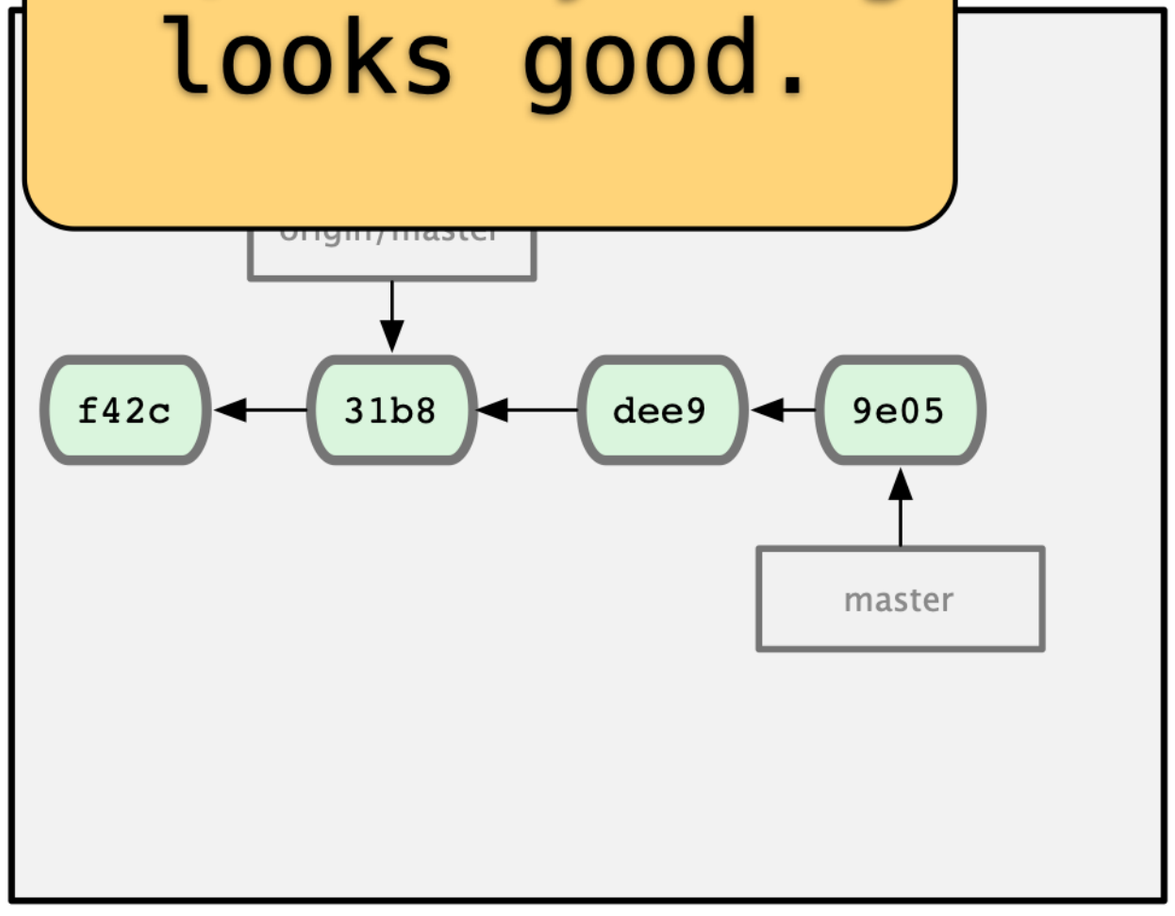


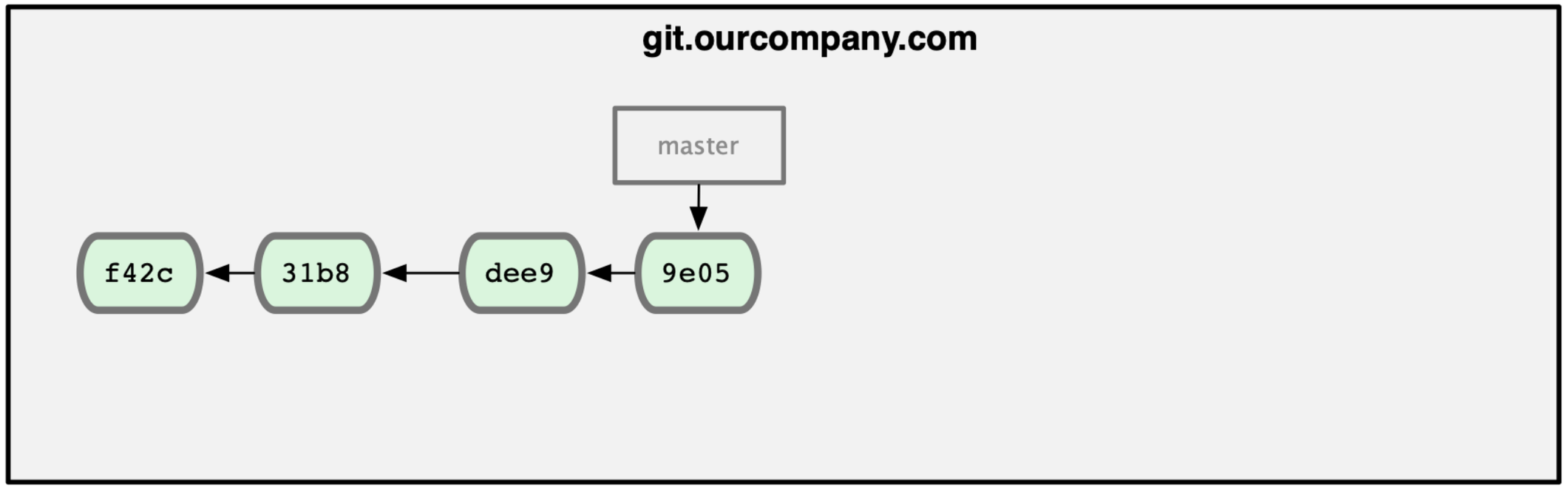


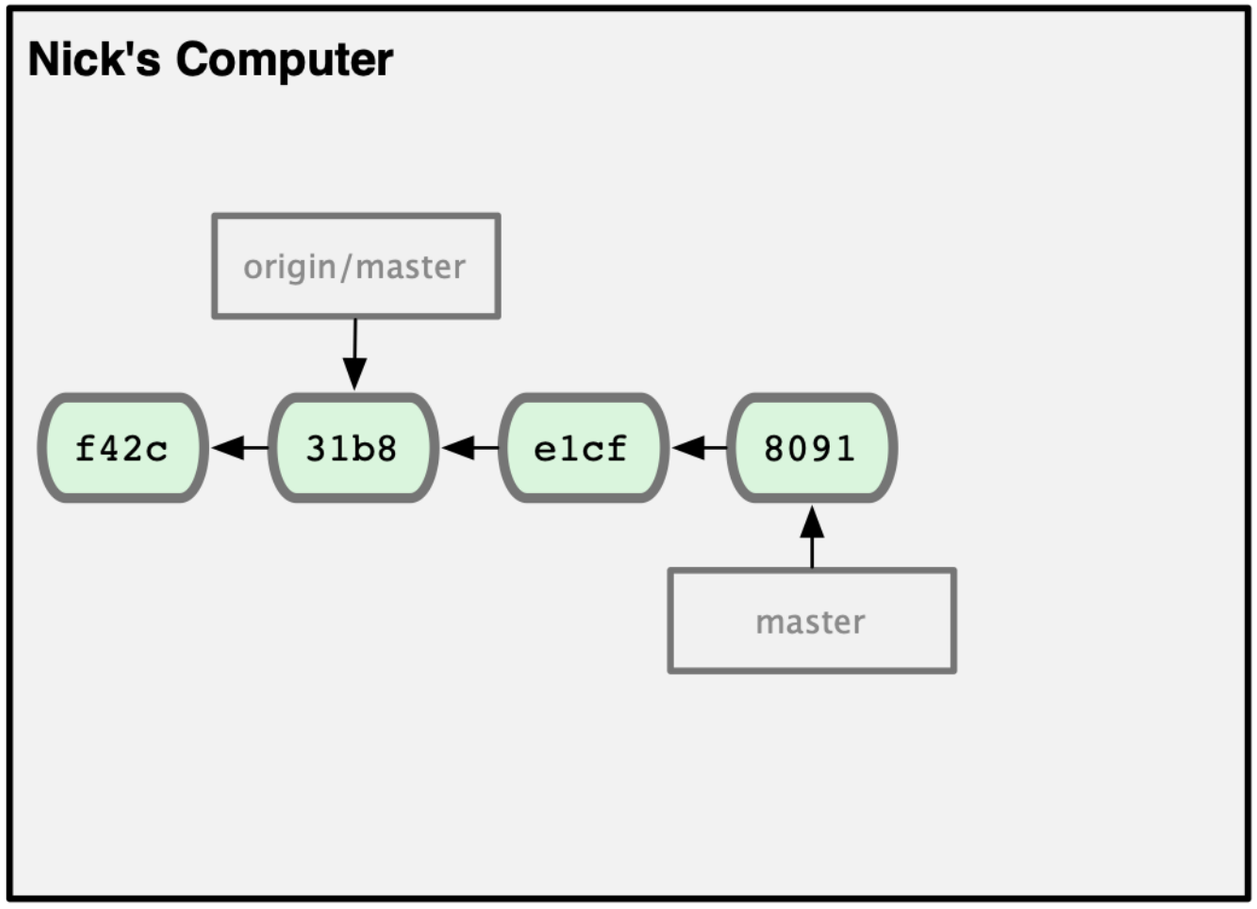


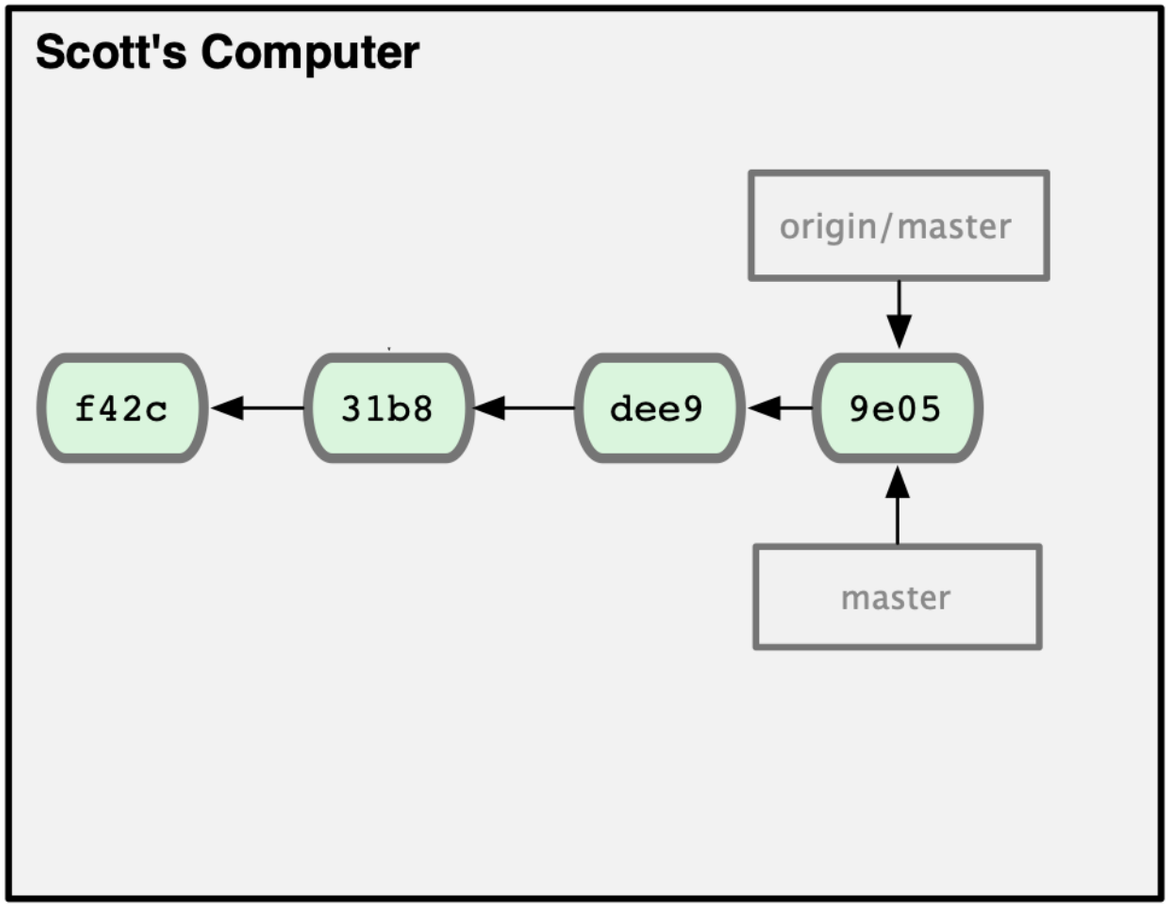
ok, everything looks good.



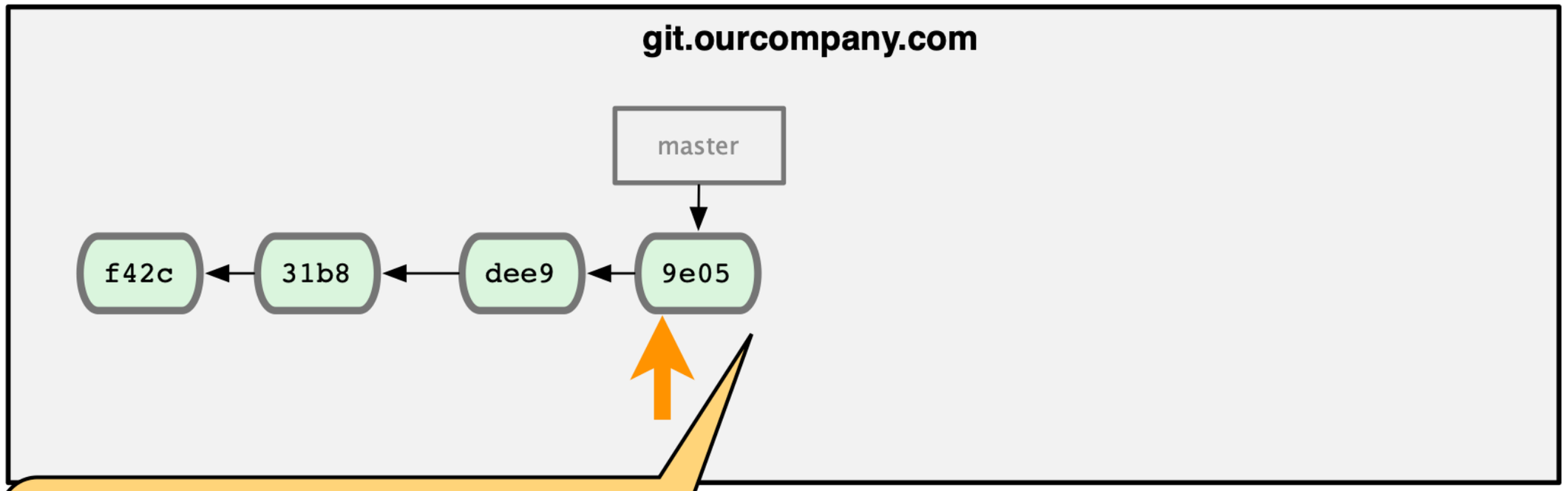



git.ourcompany.com
master
f42c
31b8
dee9
9e05

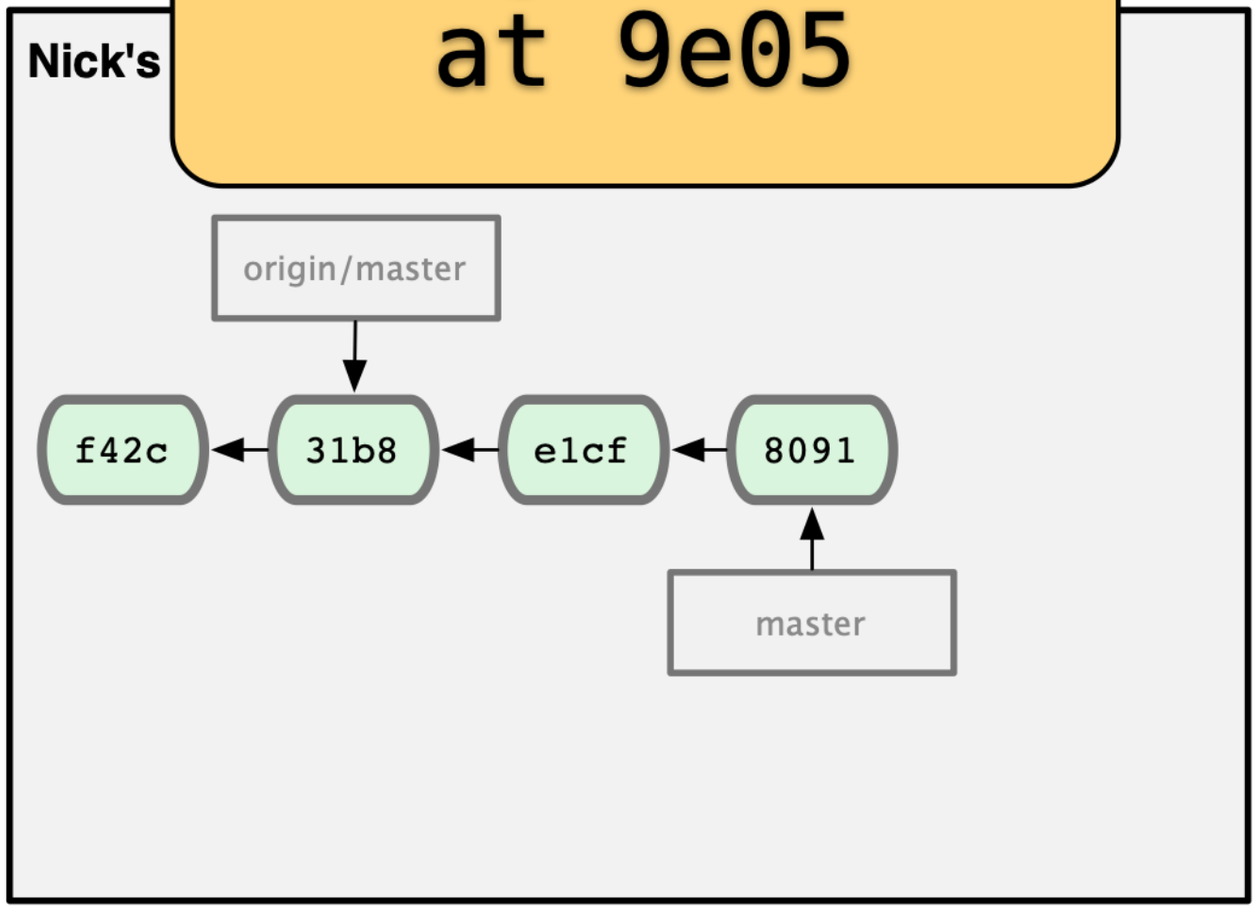

Nick's Computer
origin/master
f42c
31b8
e1cf
8091
master

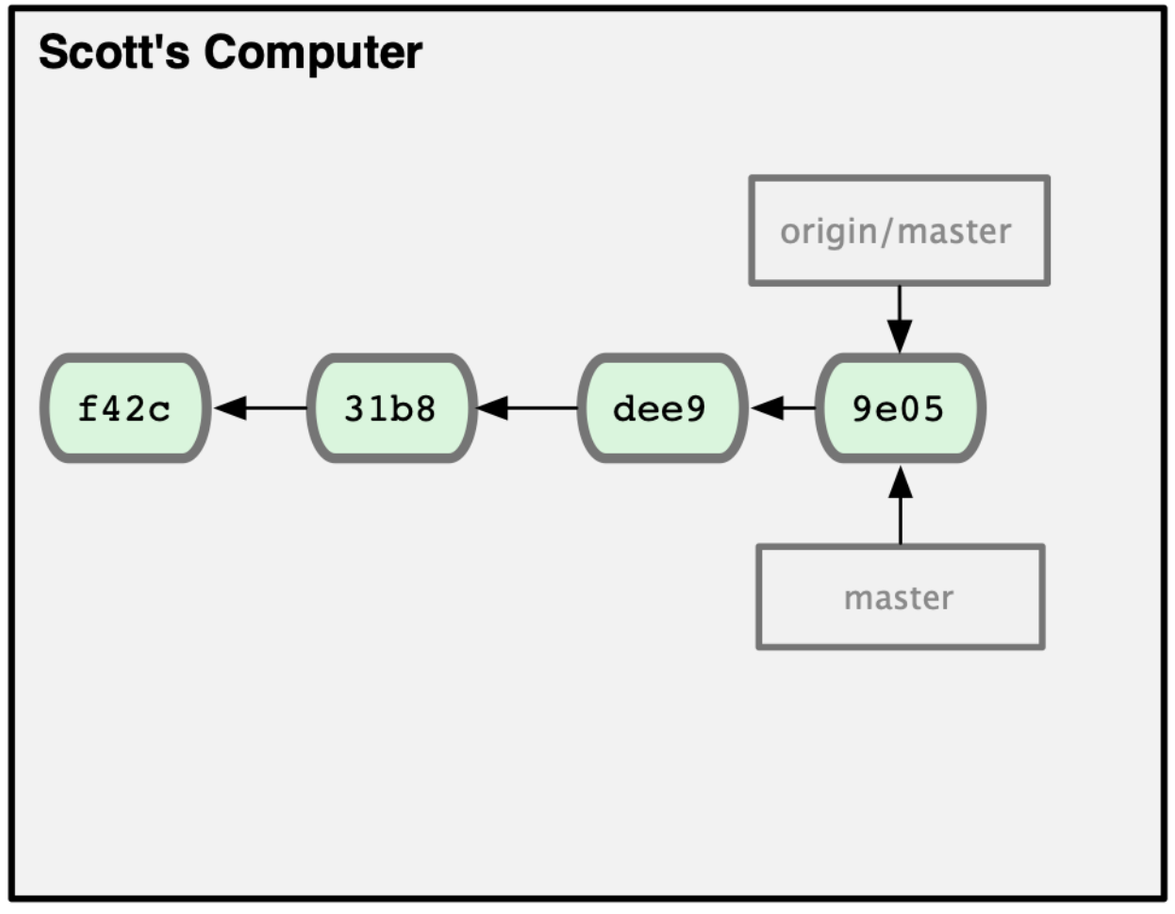

Scott's Computer
origin/master
f42c
31b8
dee9
9e05
master

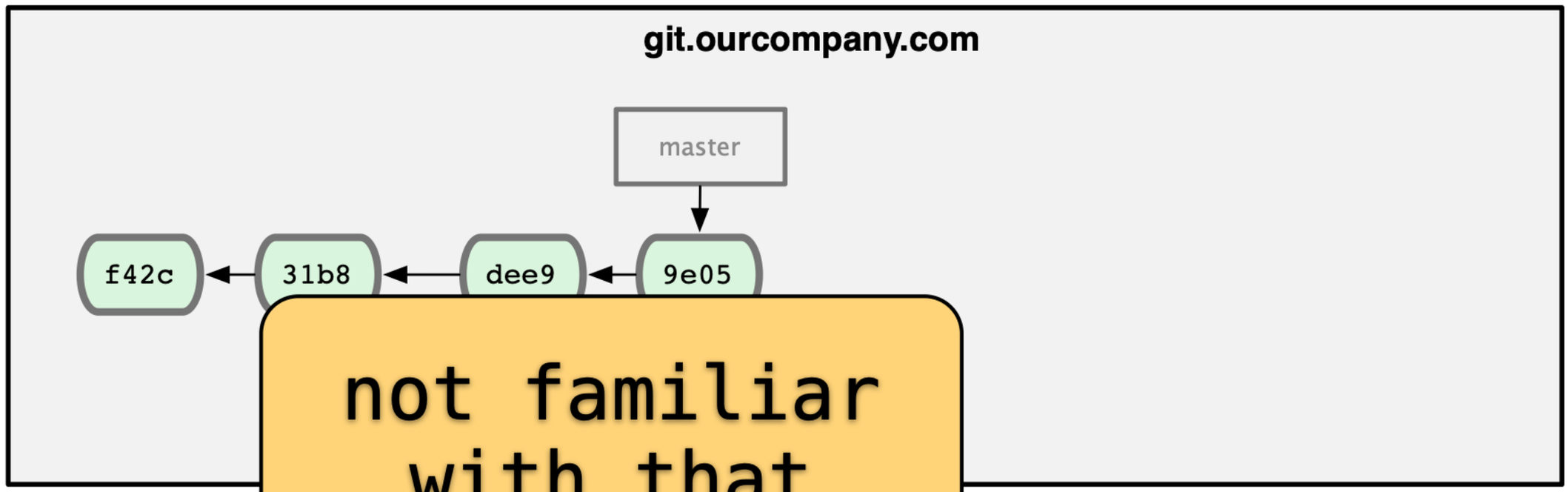

git.ourcompany.com
master
f42c
31b8
dee9
9e05
i want to push
some new stuff
git push origin master
Nick's Computer
origin/master
f42c
31b8
e1cf
8091
master
Scott's Computer
origin/master
f42c
31b8
dee9
9e05
master



i've got master at 9e05

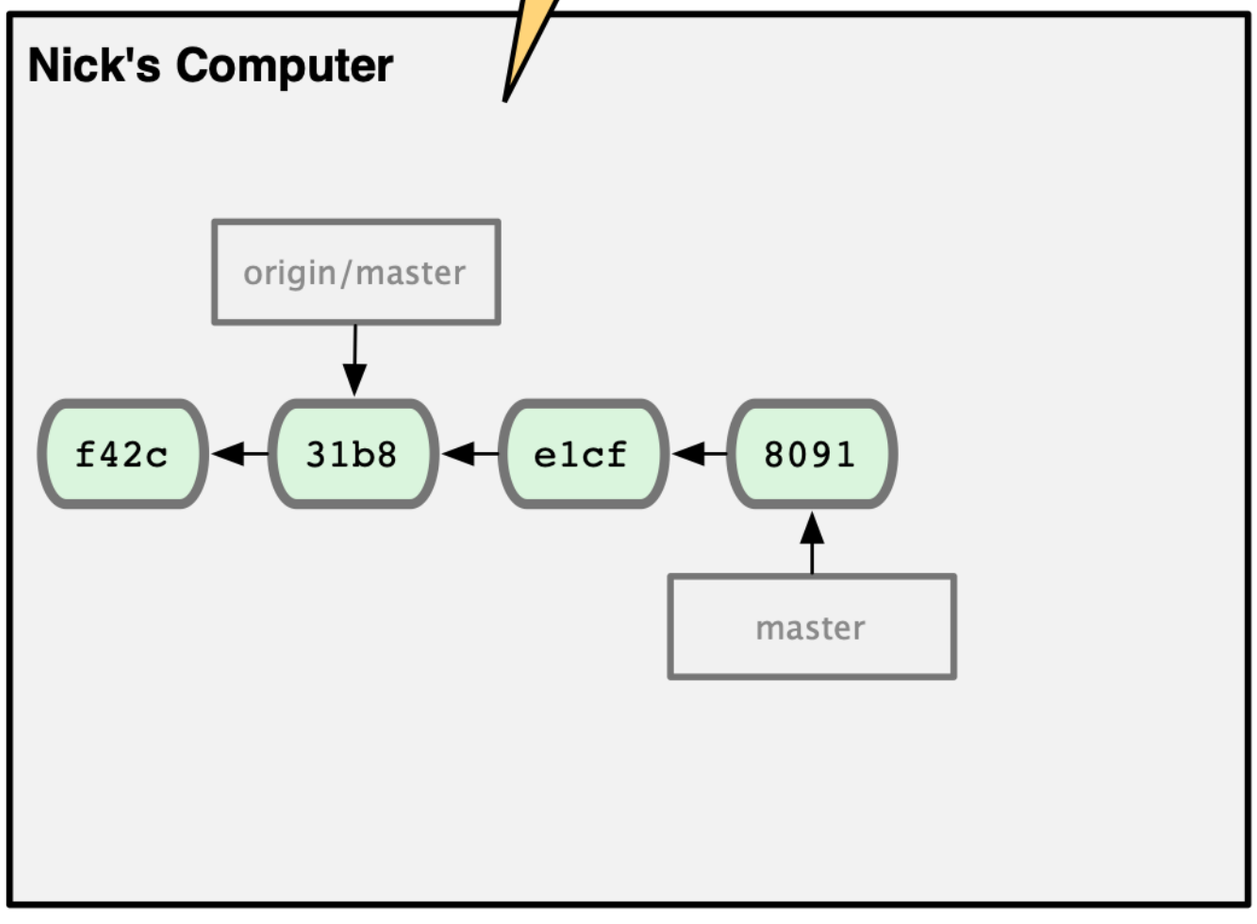


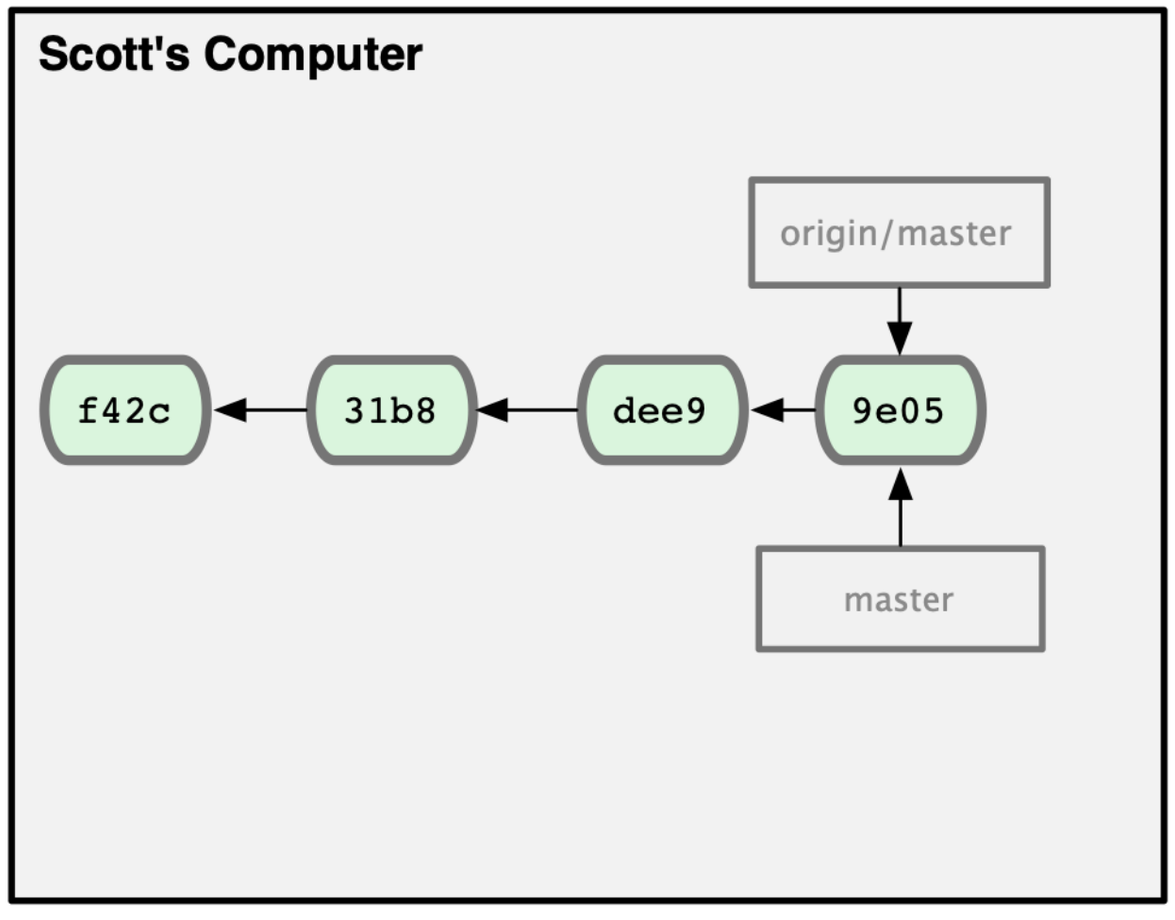




not familiar with that

git pus





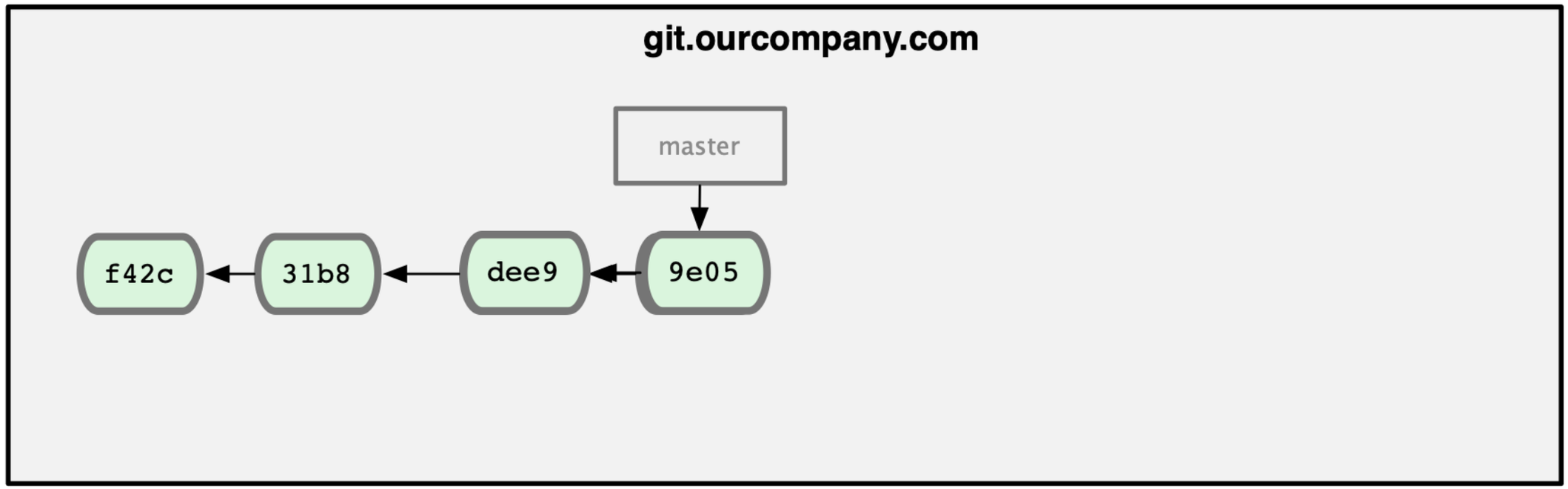
REMOTES – CONFLITTI

Per risolvere questo disallineamento è necessario

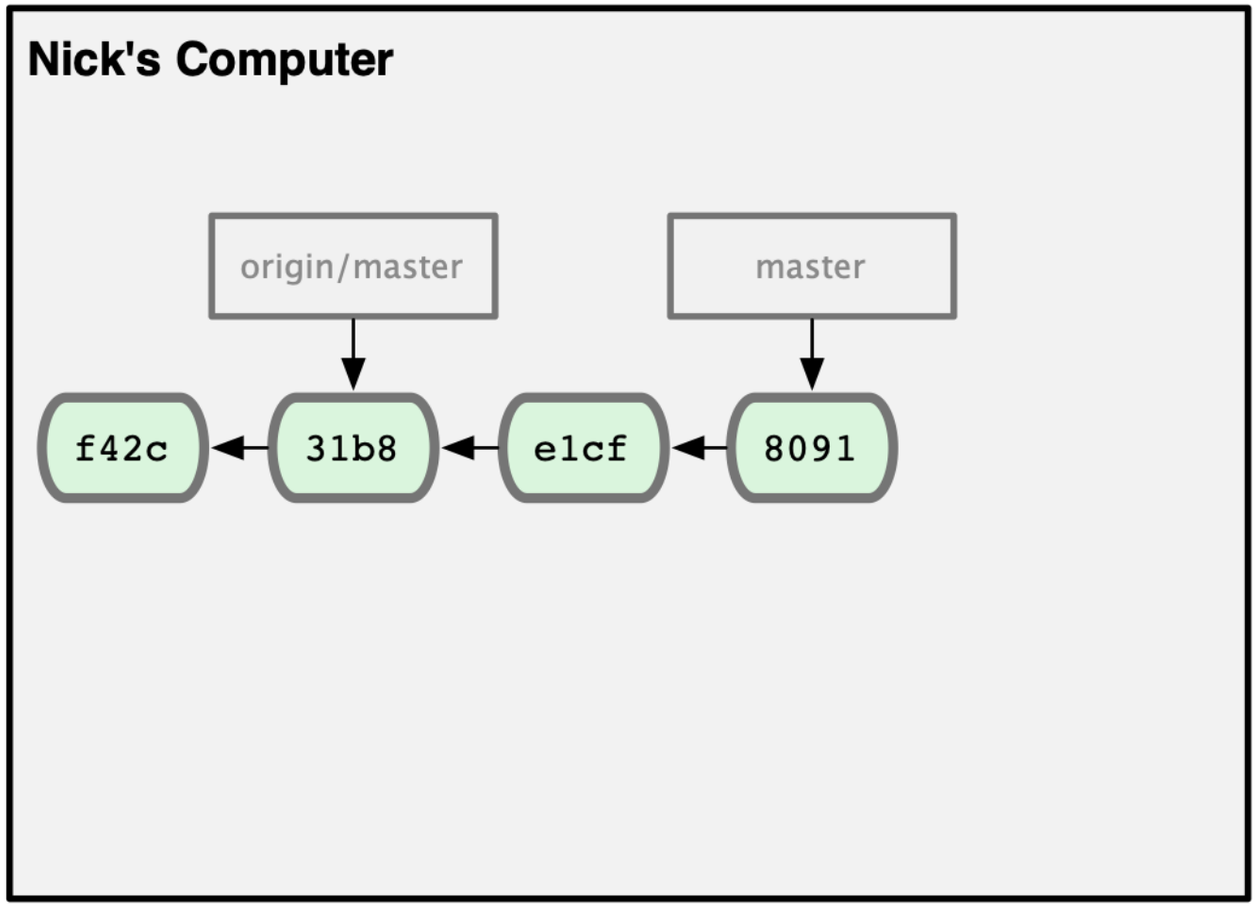
- ottenere dal repository remoto lo stato attuale
- effettuare il merge tra l'indice remoto e quello locale
  - risolvere eventuali conflitti
- aggiornare il repository remoto

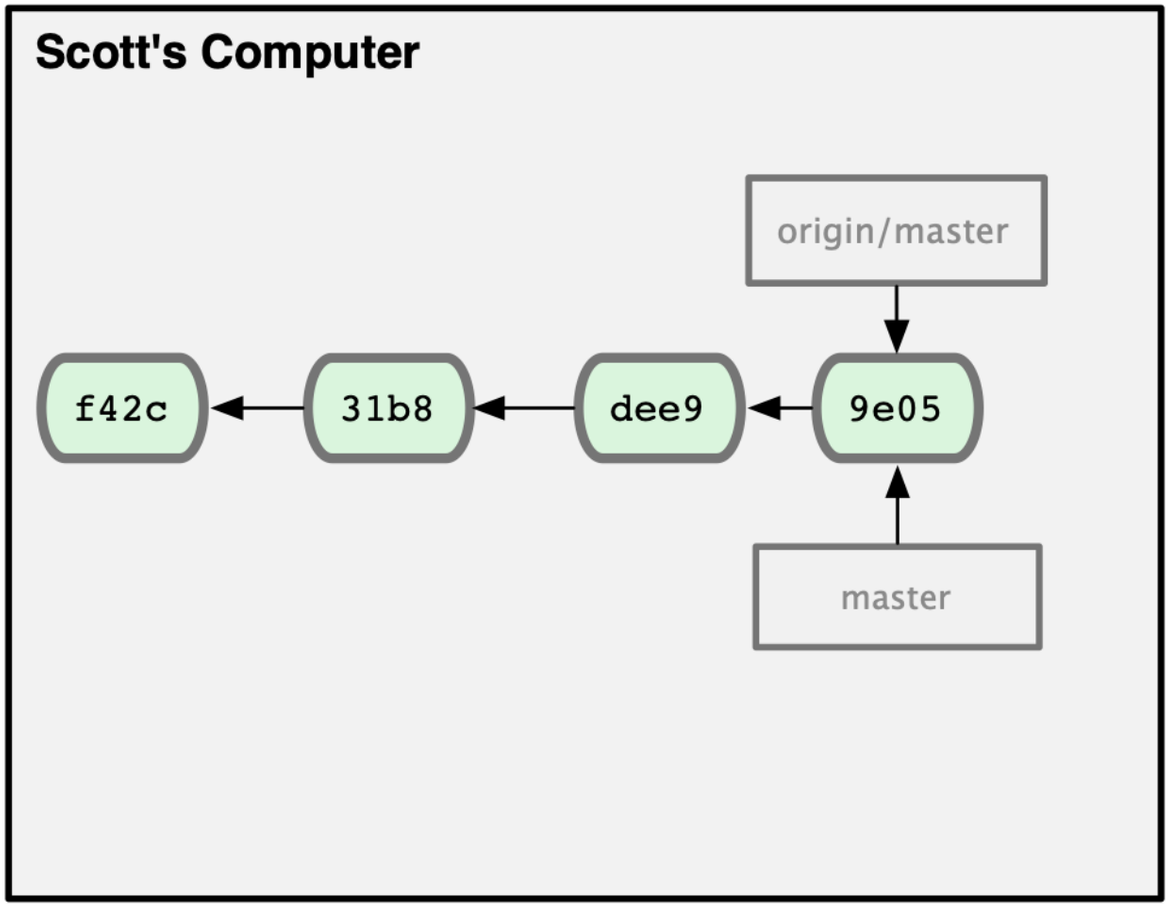
```
git fetch


git pull
```

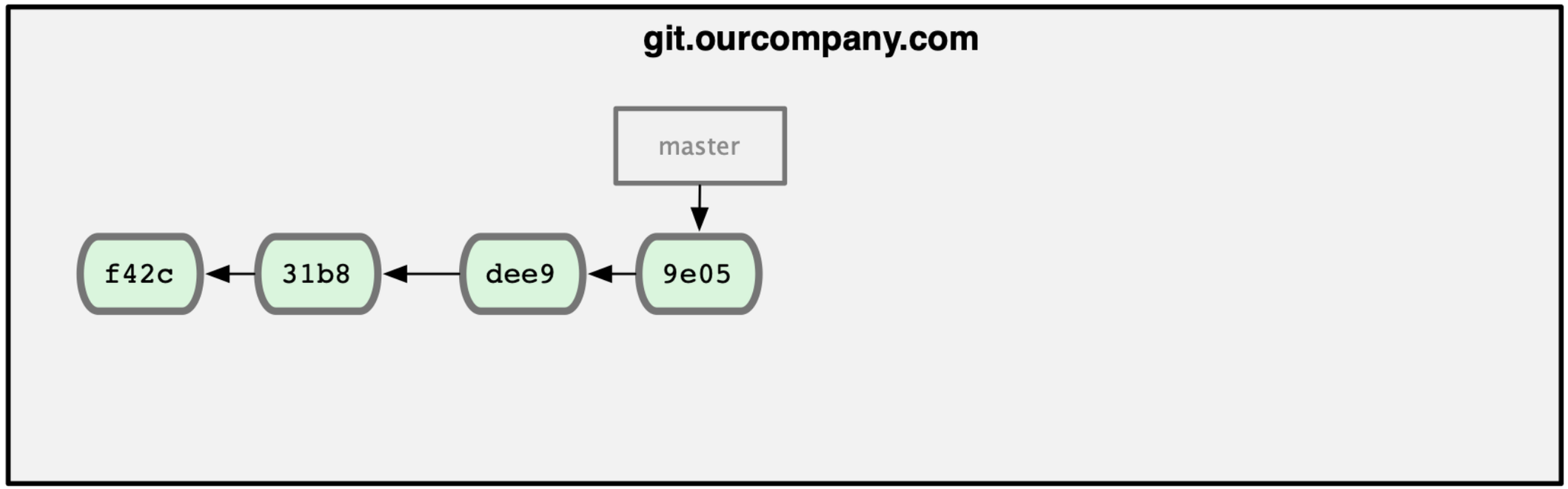
pull = fetch + merge


git.ourcompany.com
master
f42c
31b8
dee9
9e05

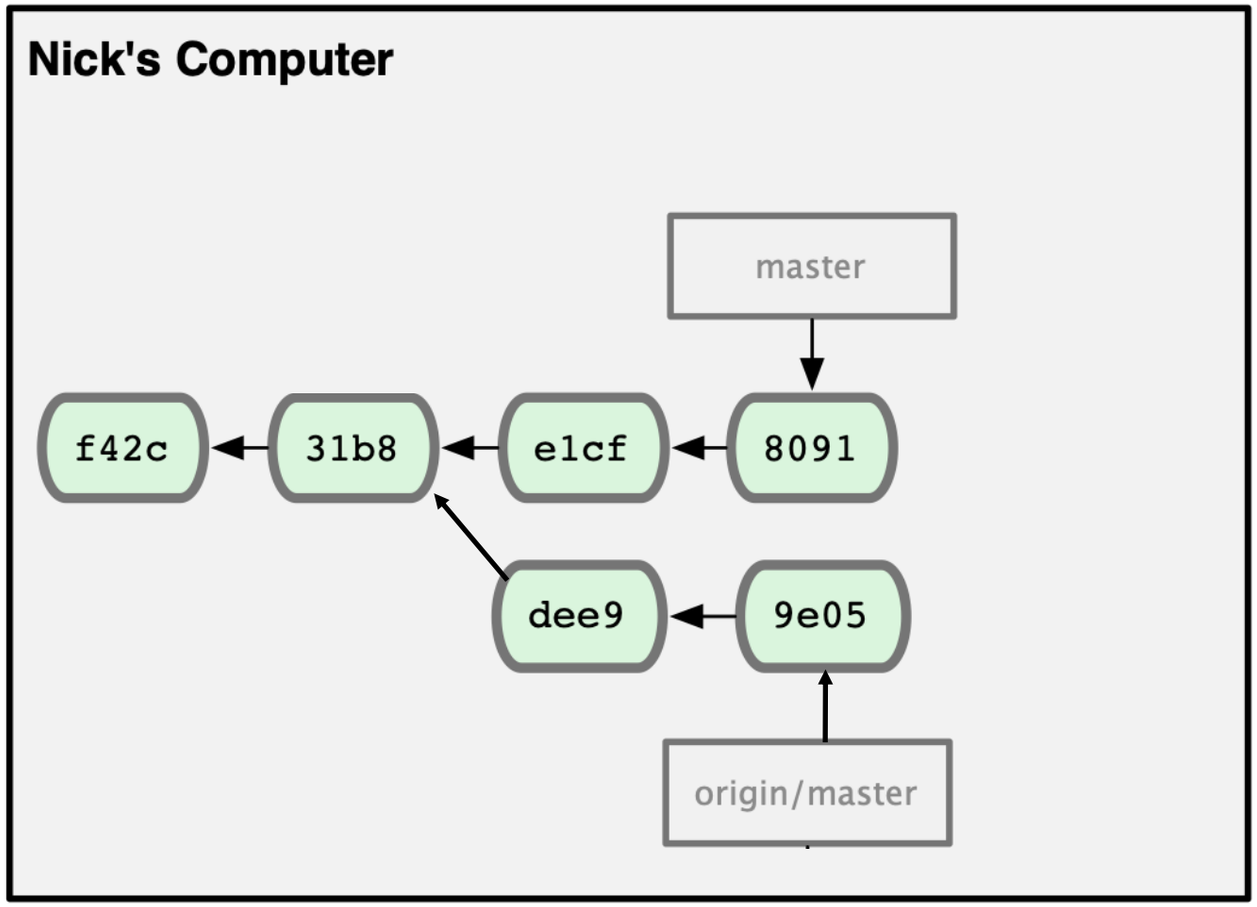
`git fetch`

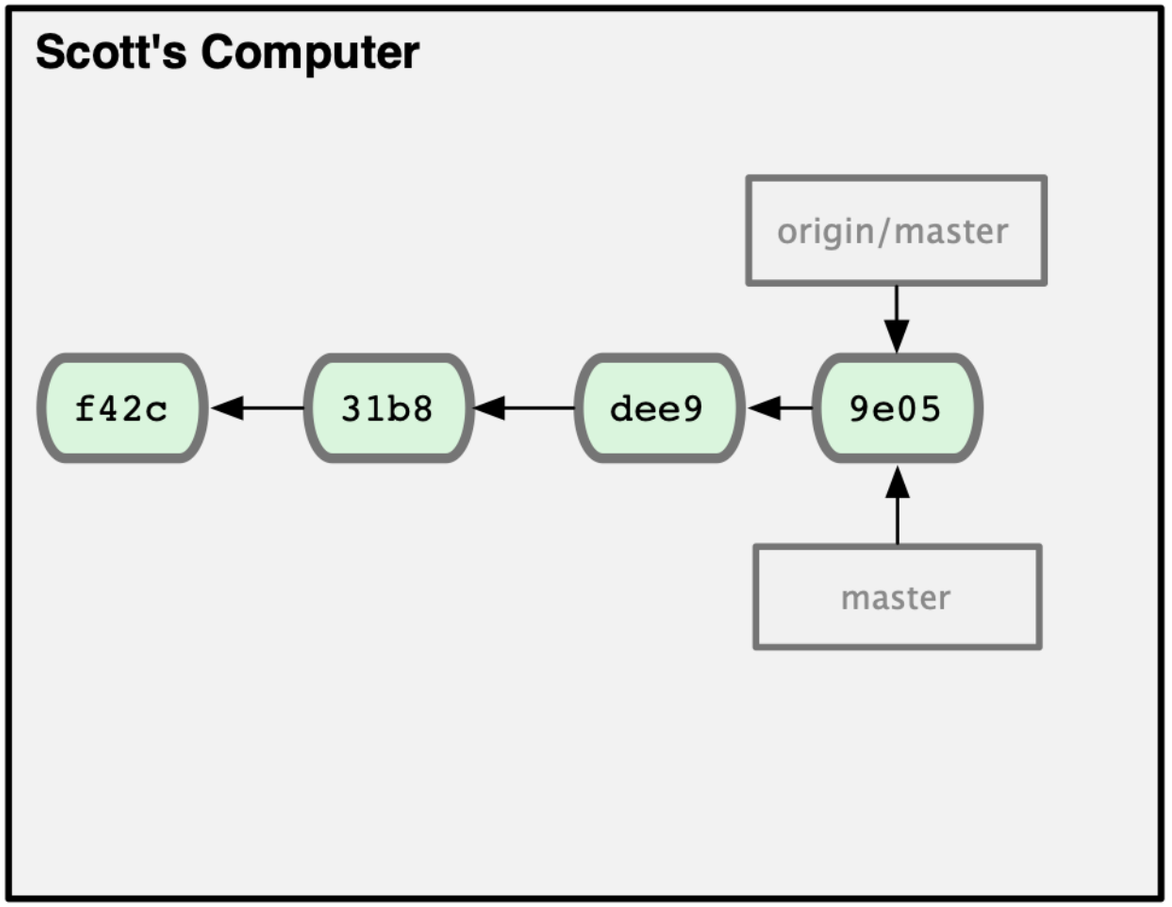

Nick's Computer
origin/master
master
f42c
31b8
e1cf
8091

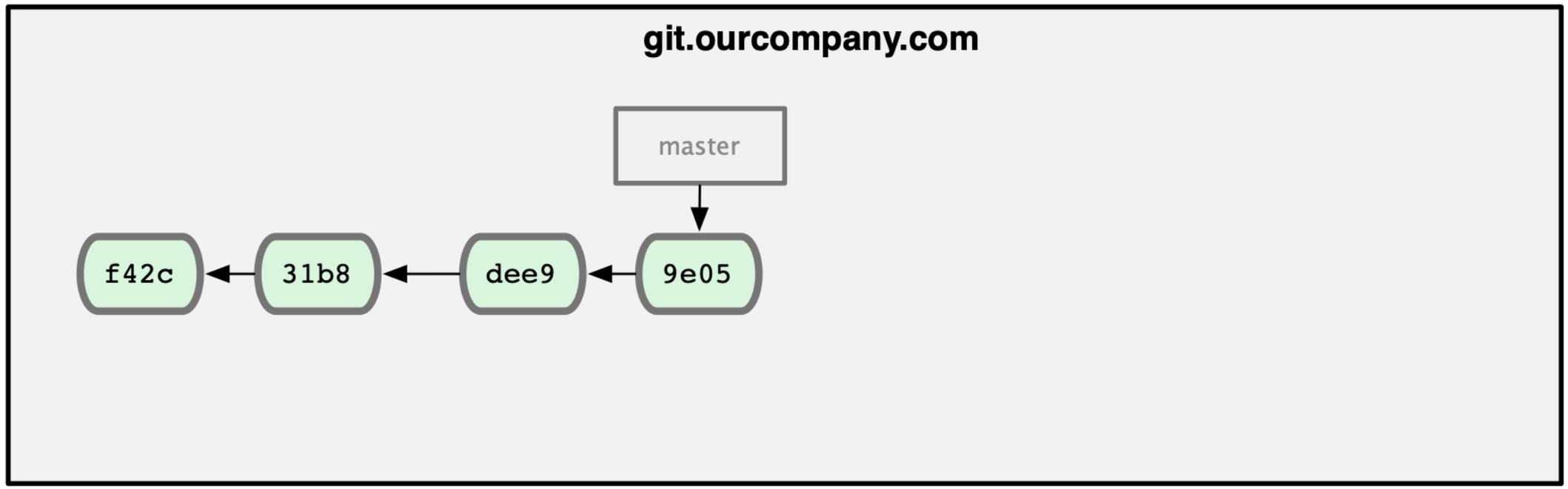

Scott's Computer
origin/master
f42c
31b8
dee9
9e05
master


git.ourcompany.com
master
f42c
31b8
dee9
9e05

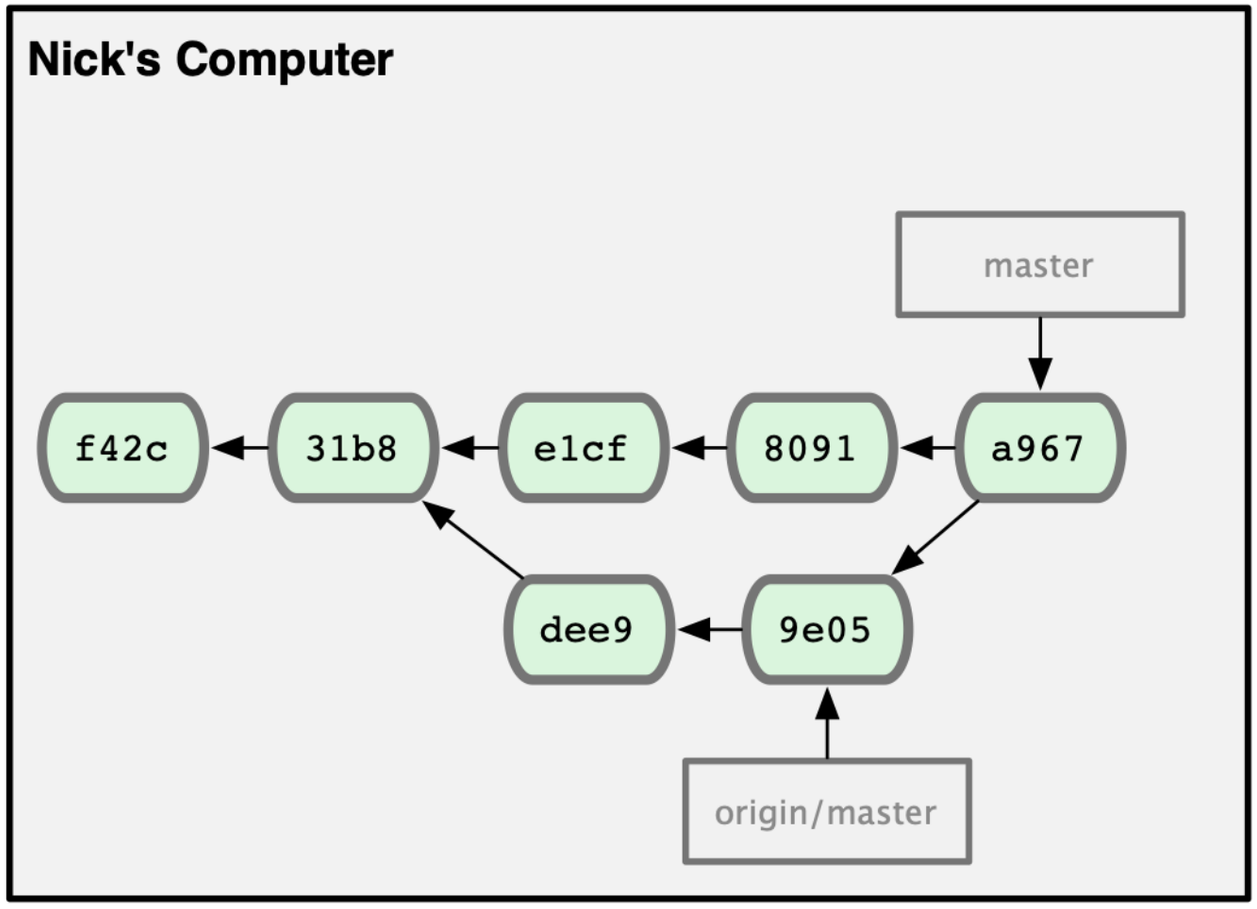
`git fetch`

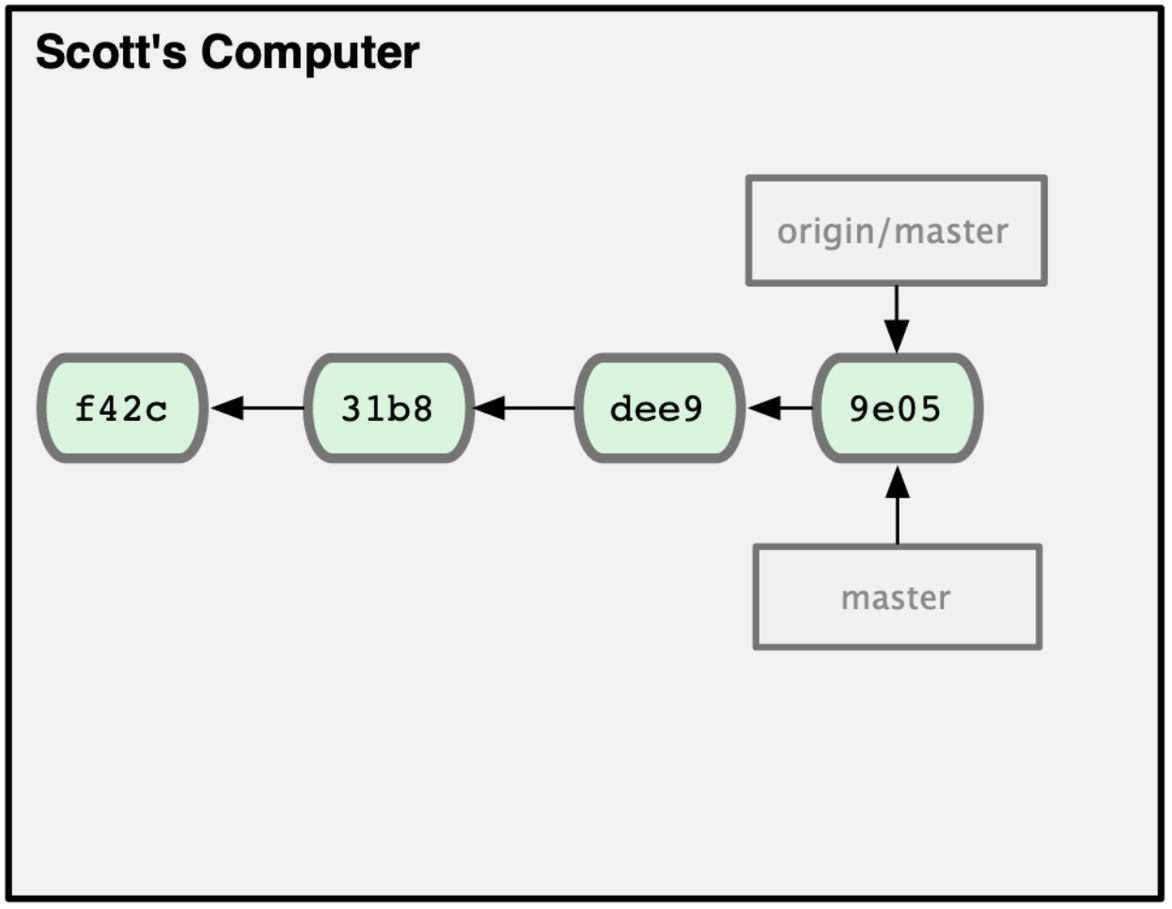

Nick's Computer
master
f42c
31b8
e1cf
8091
dee9
9e05
origin/master

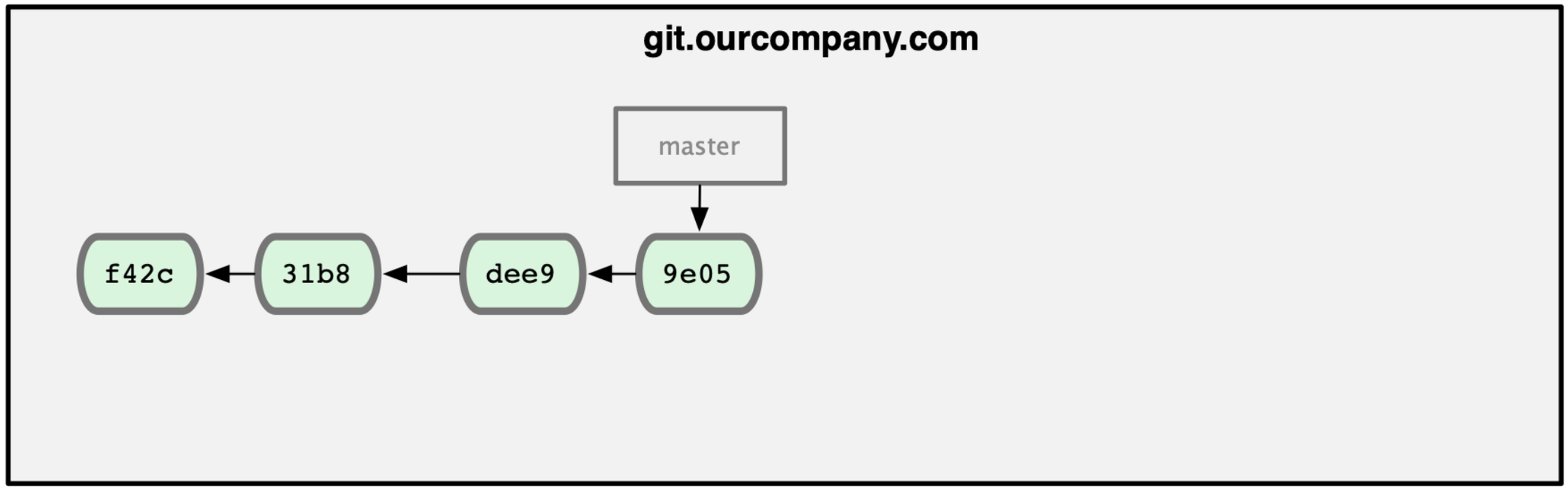

Scott's Computer
origin/master
f42c
31b8
dee9
9e05
master



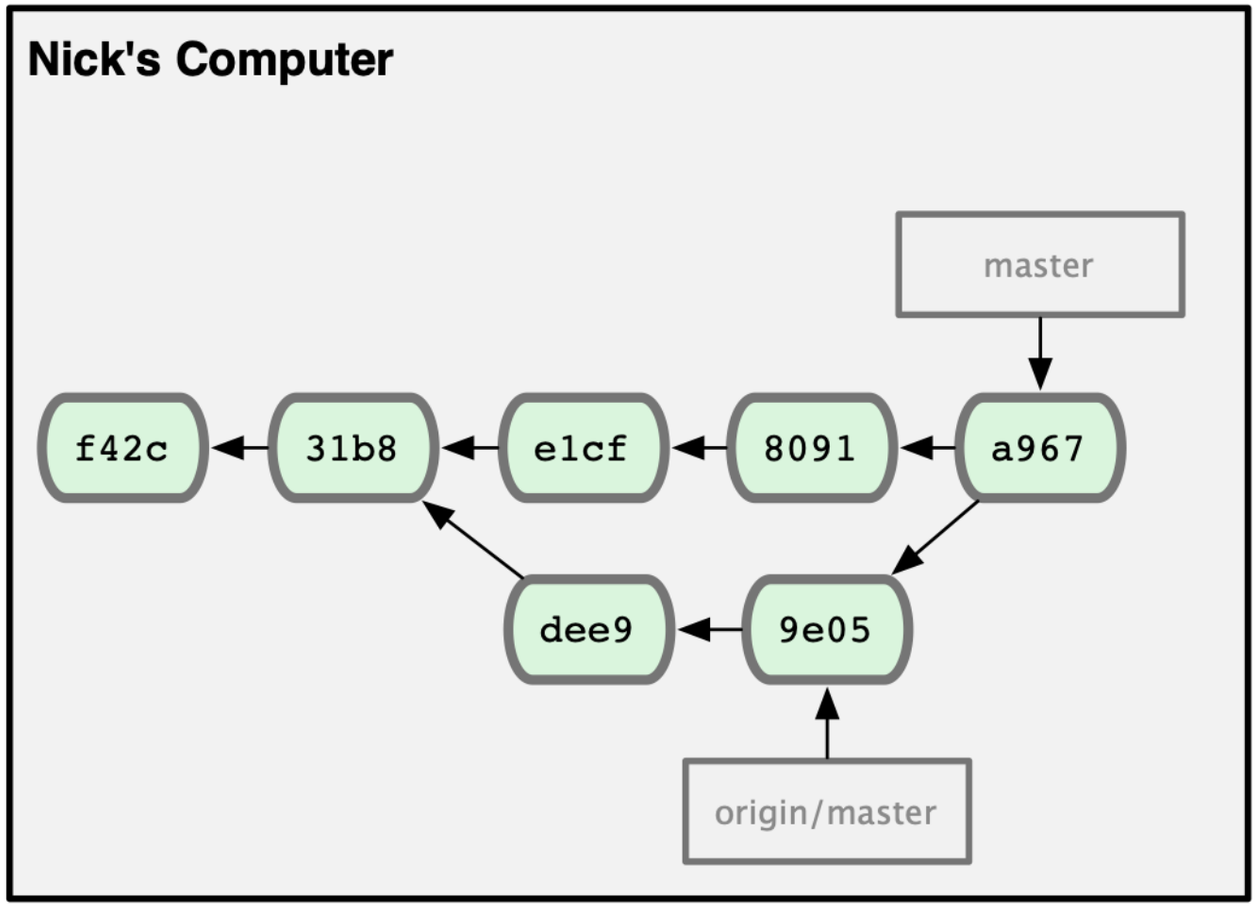

```
git merge origin/master
```

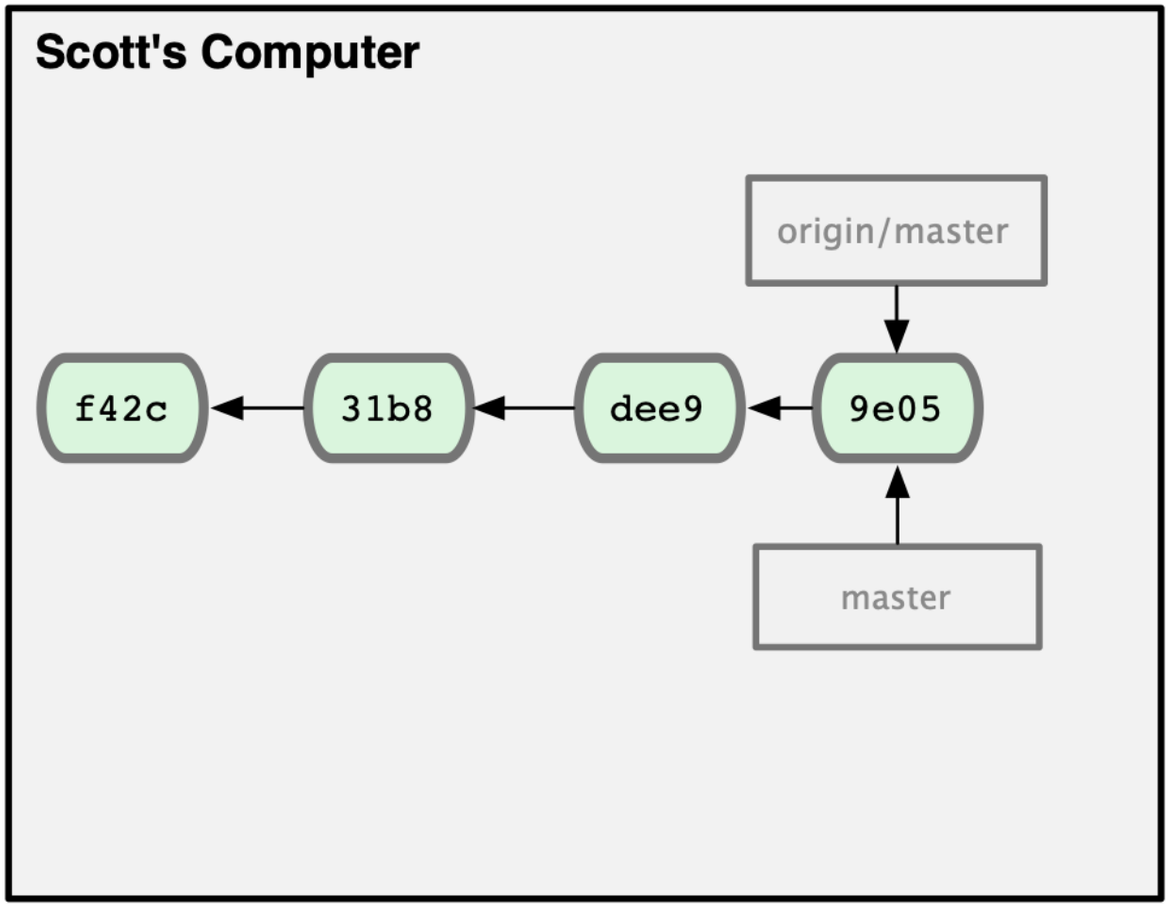


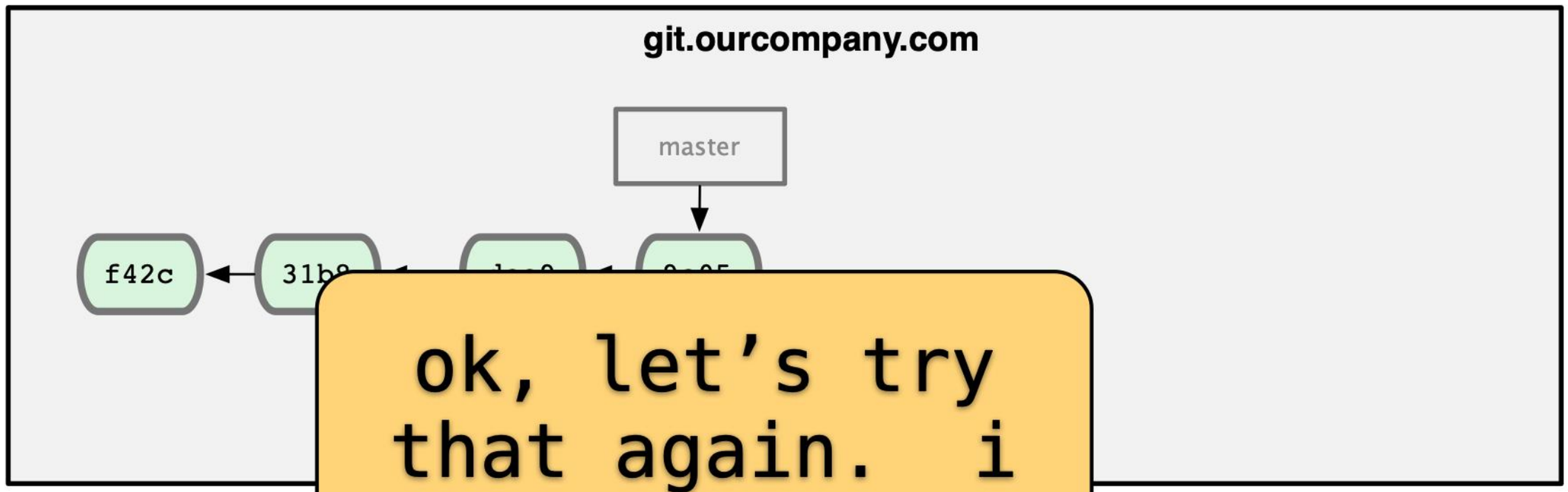





```
git push origin master
```

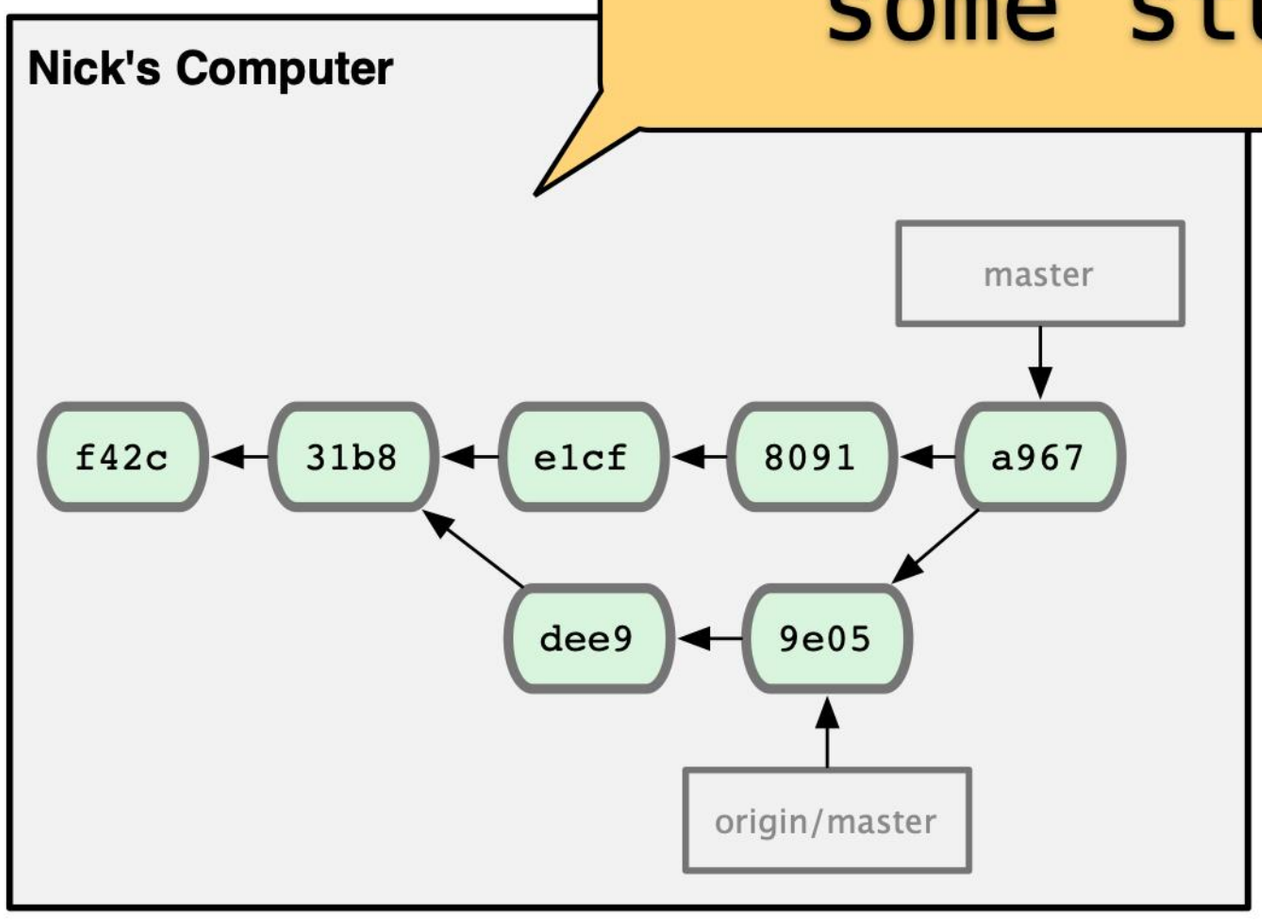


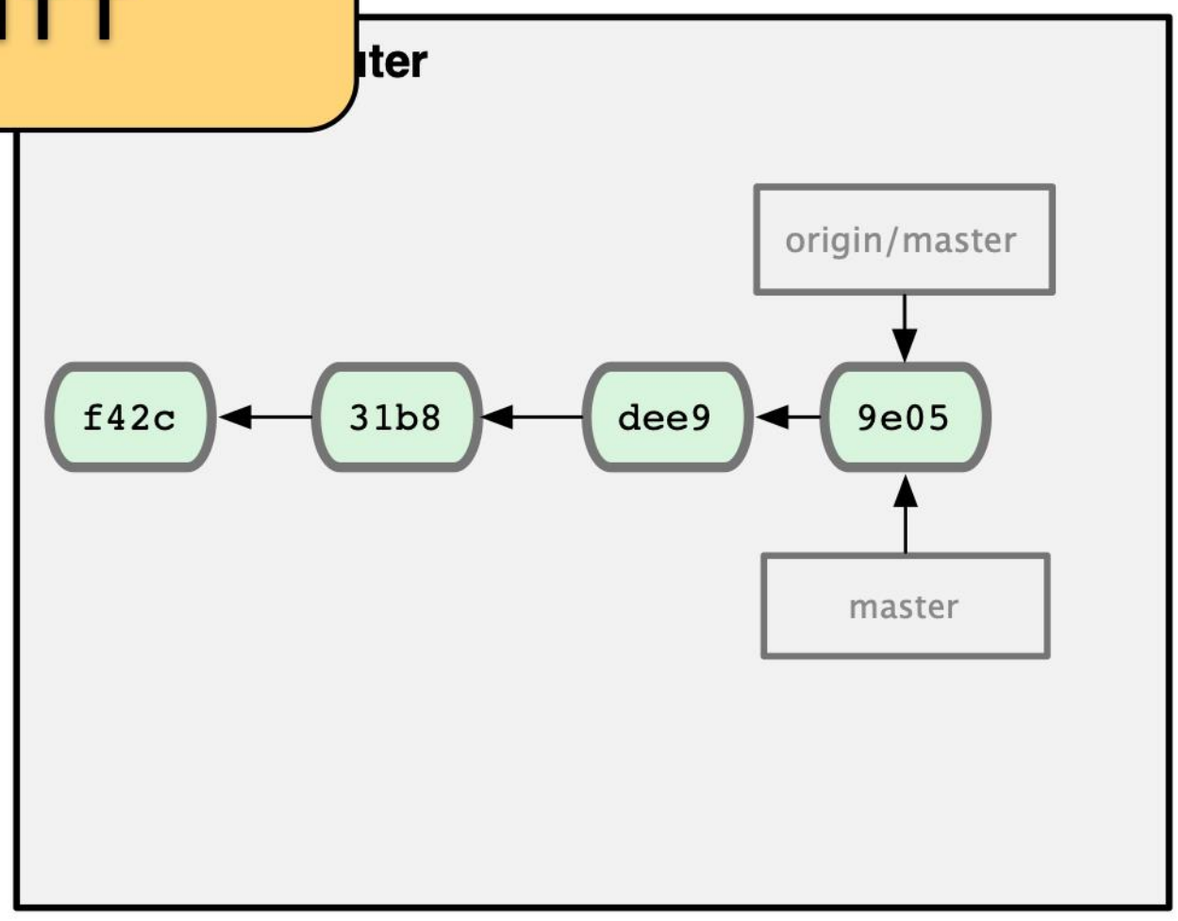


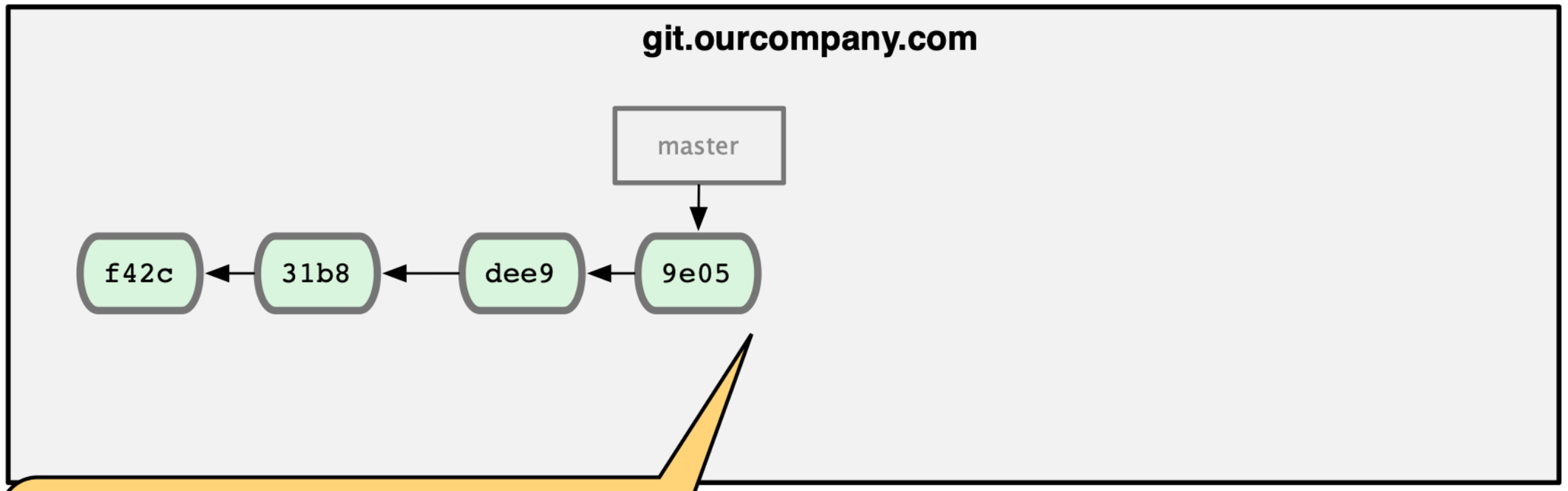


ok, let's try that again. i want to push some stuff

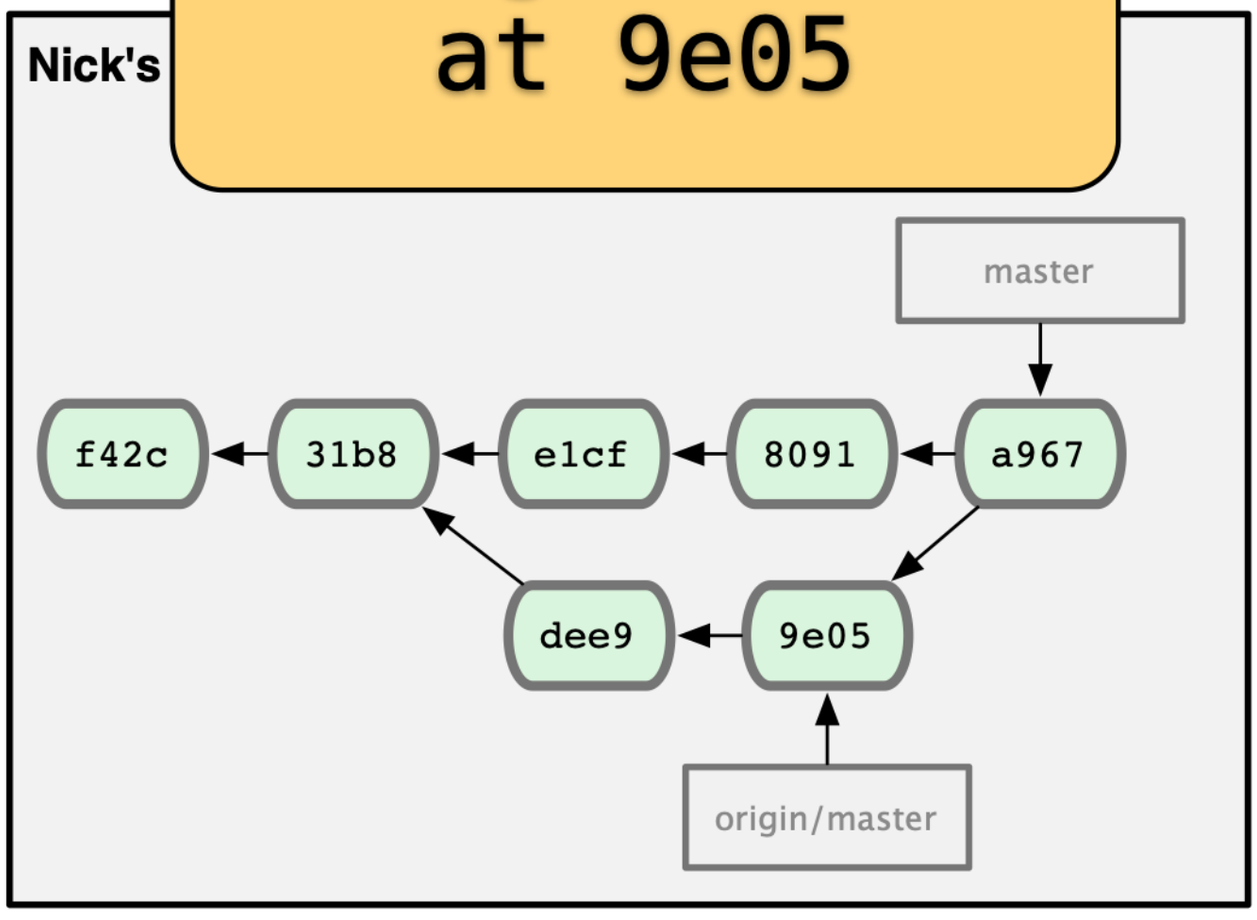
git push

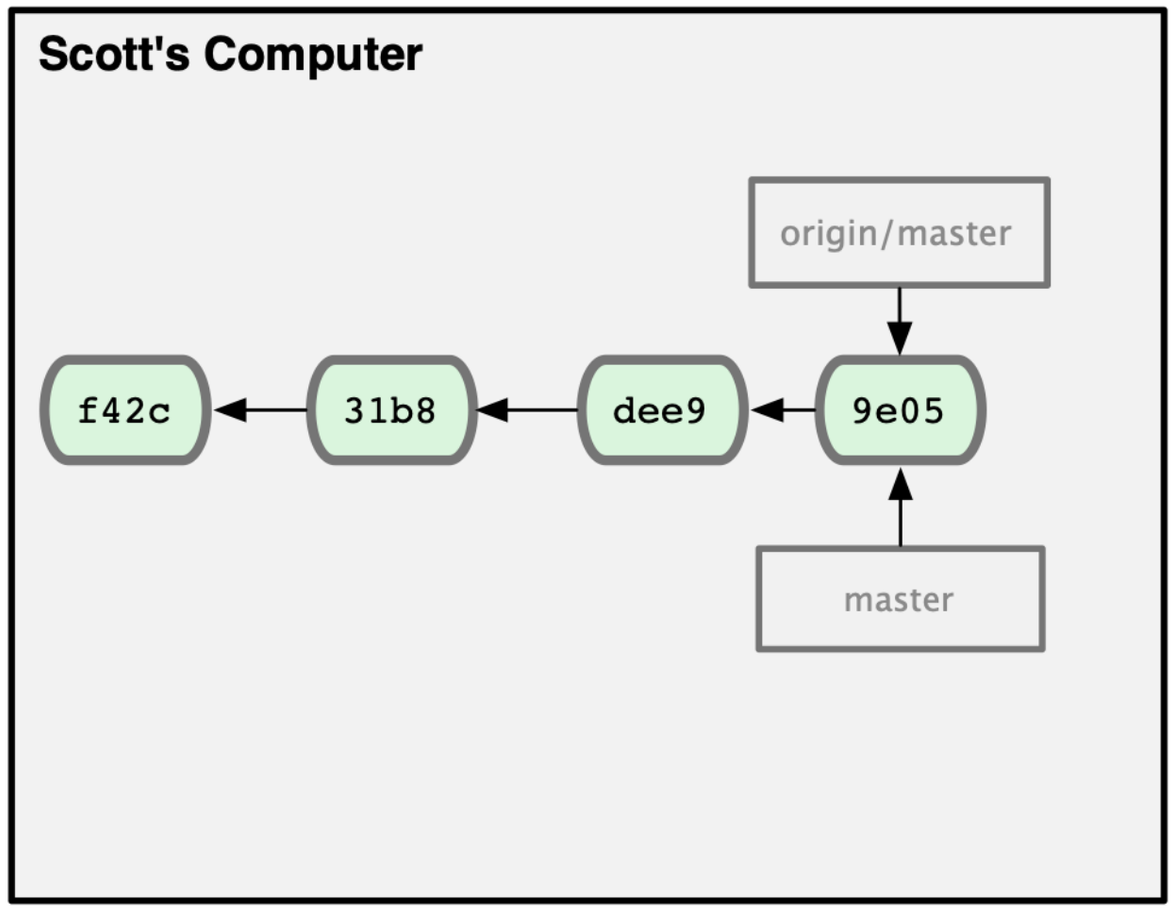


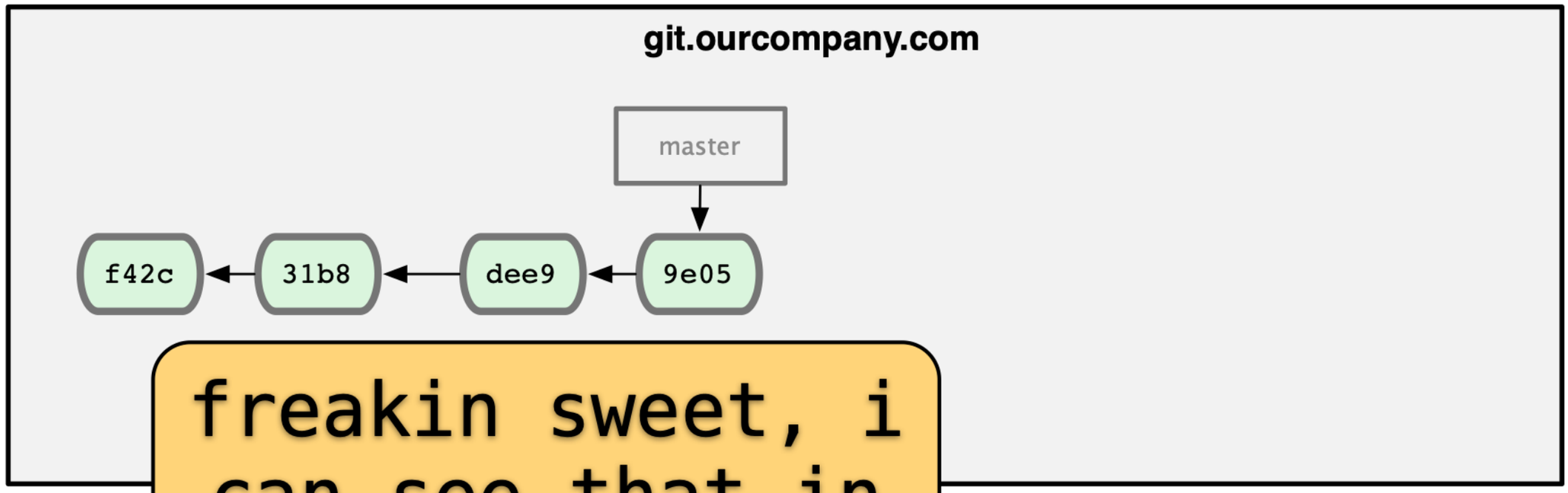




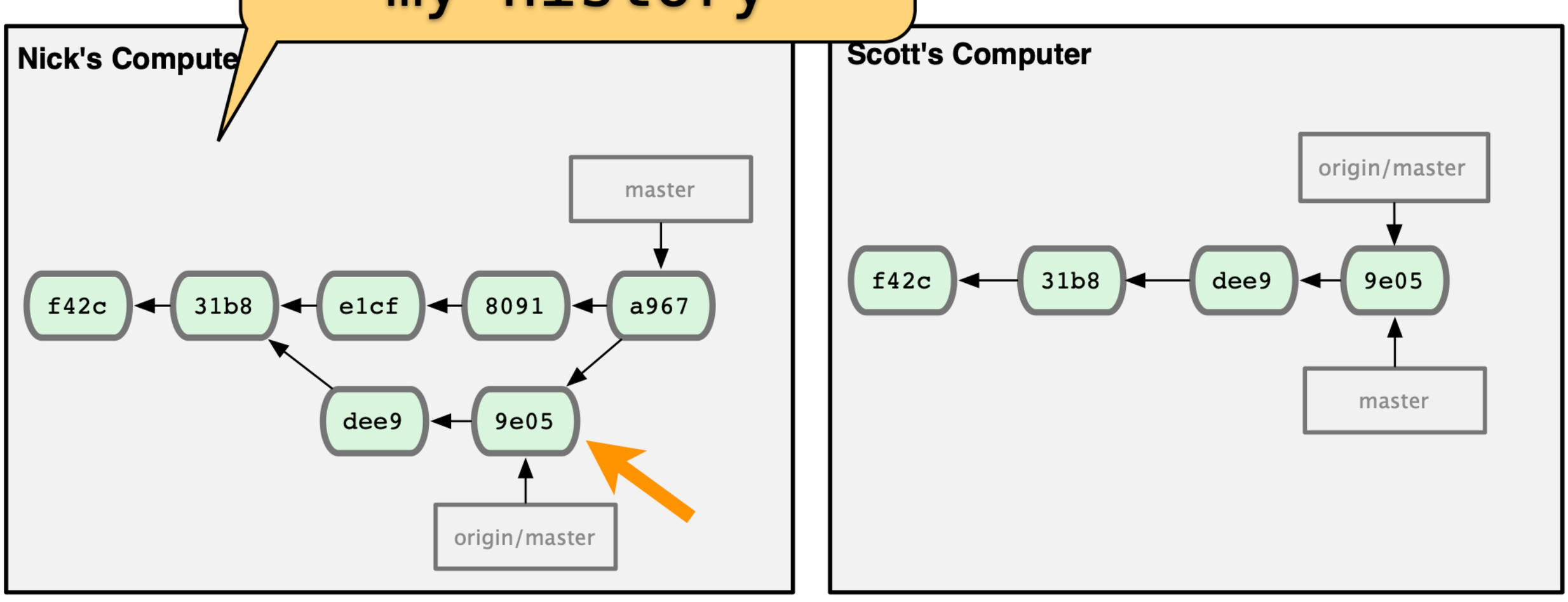
i've got master at 9e05

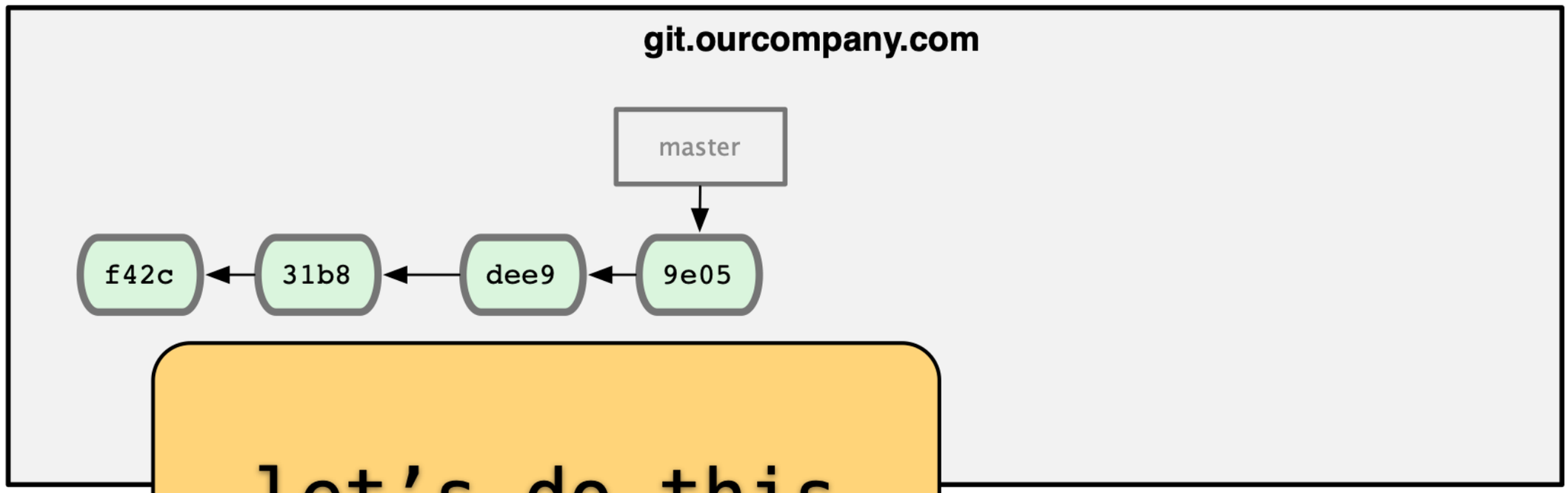






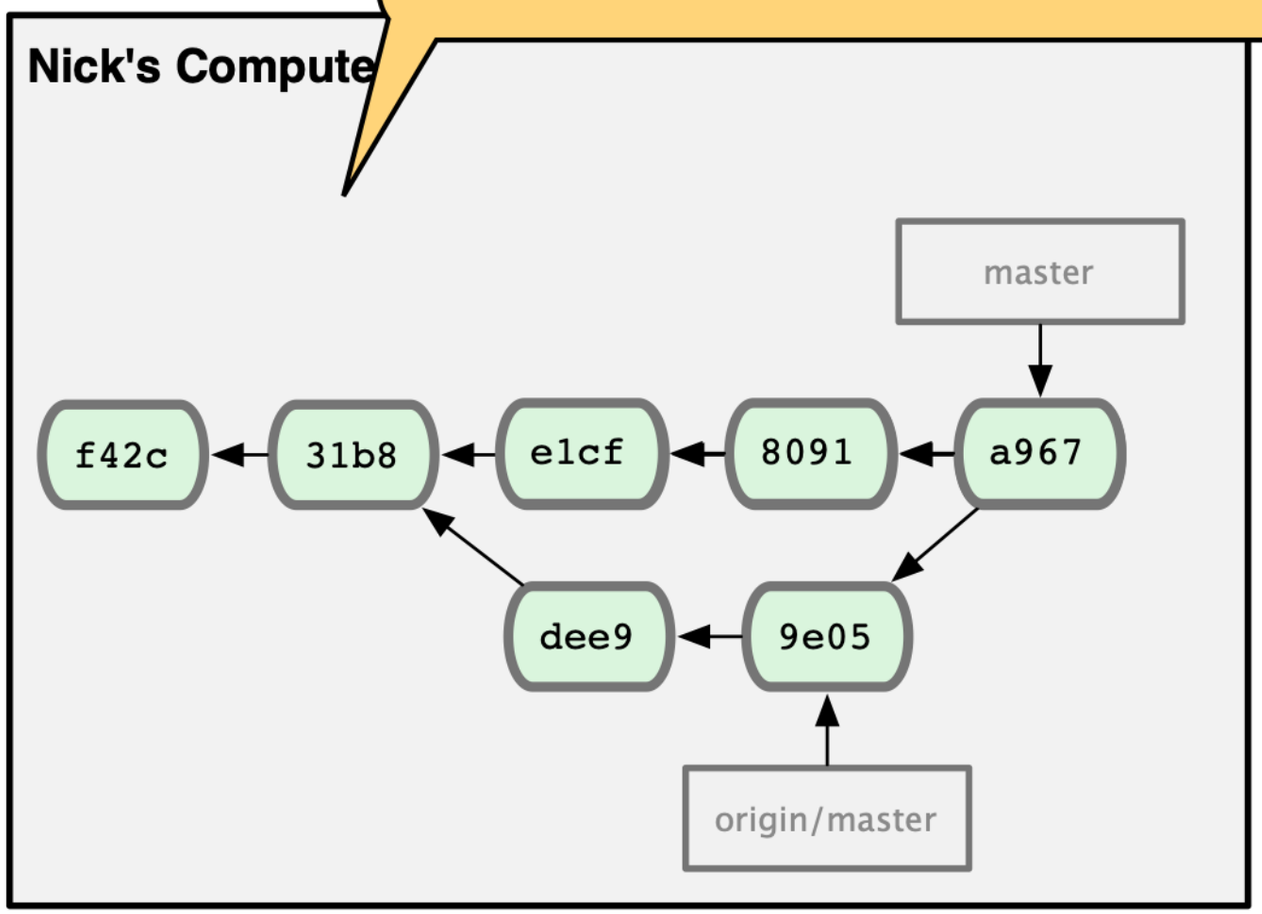
freakin sweet, i can see that in my history

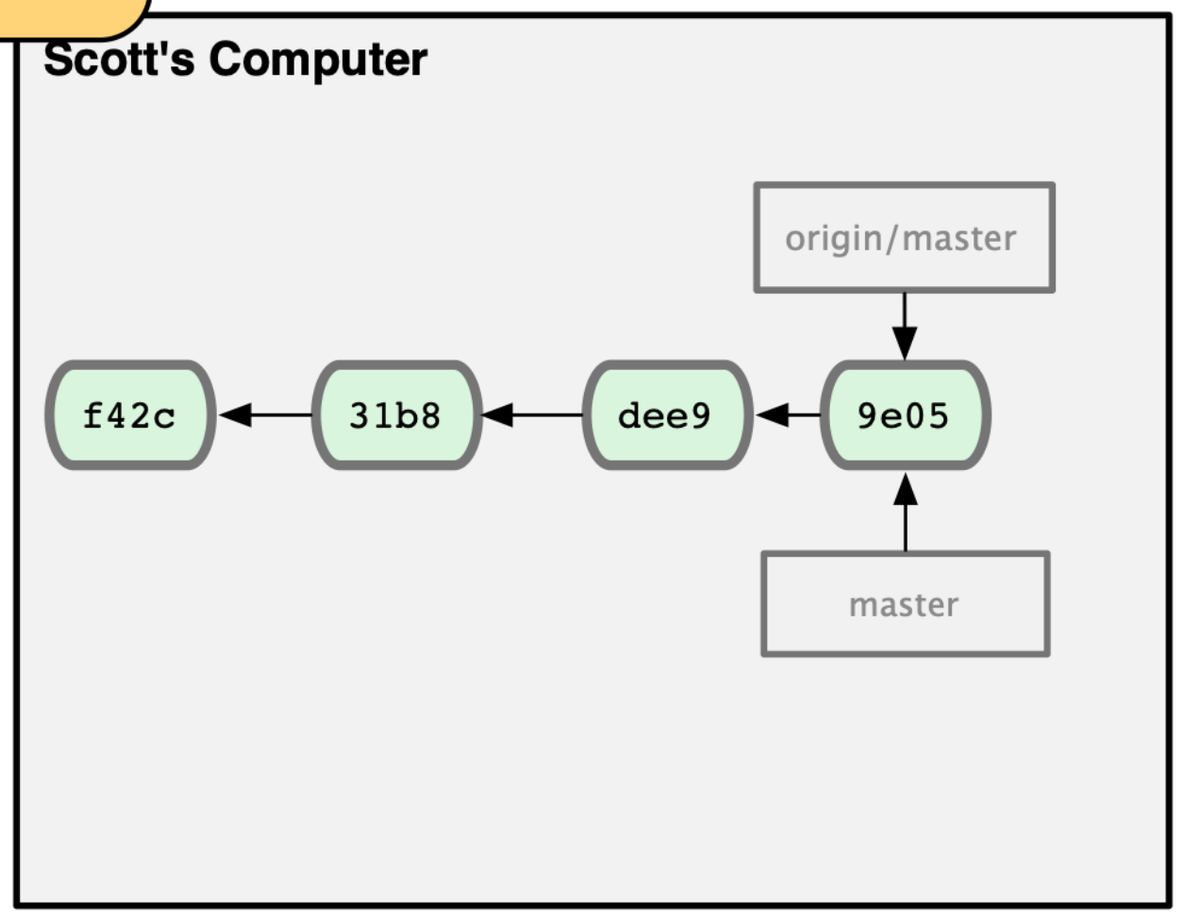


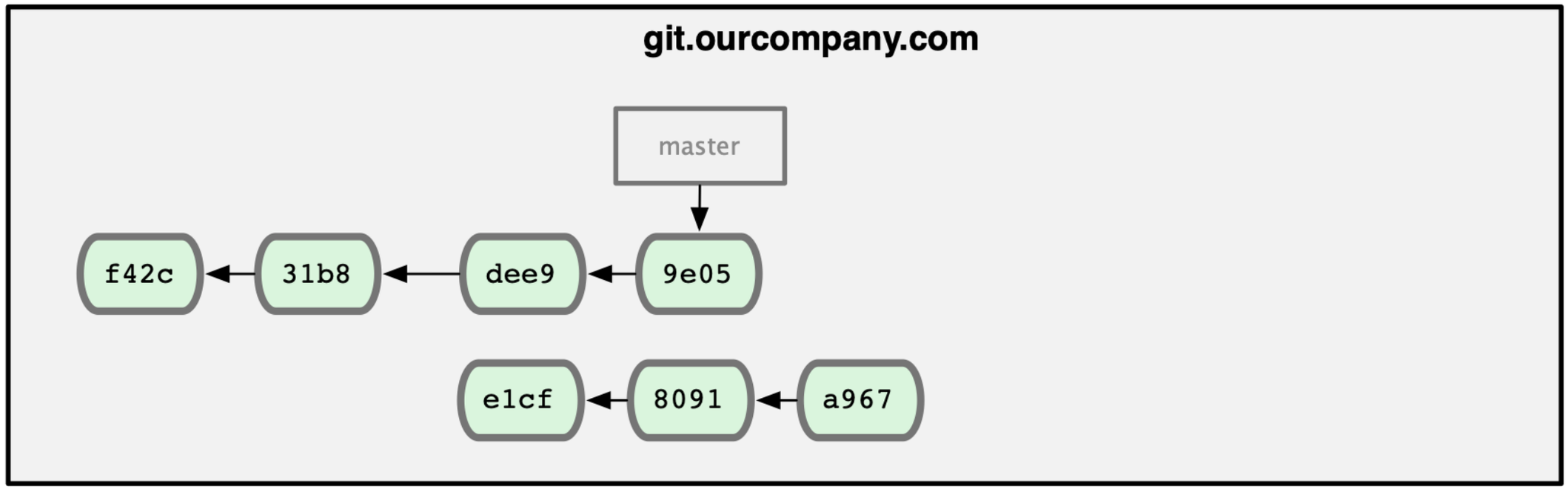


let’s do this

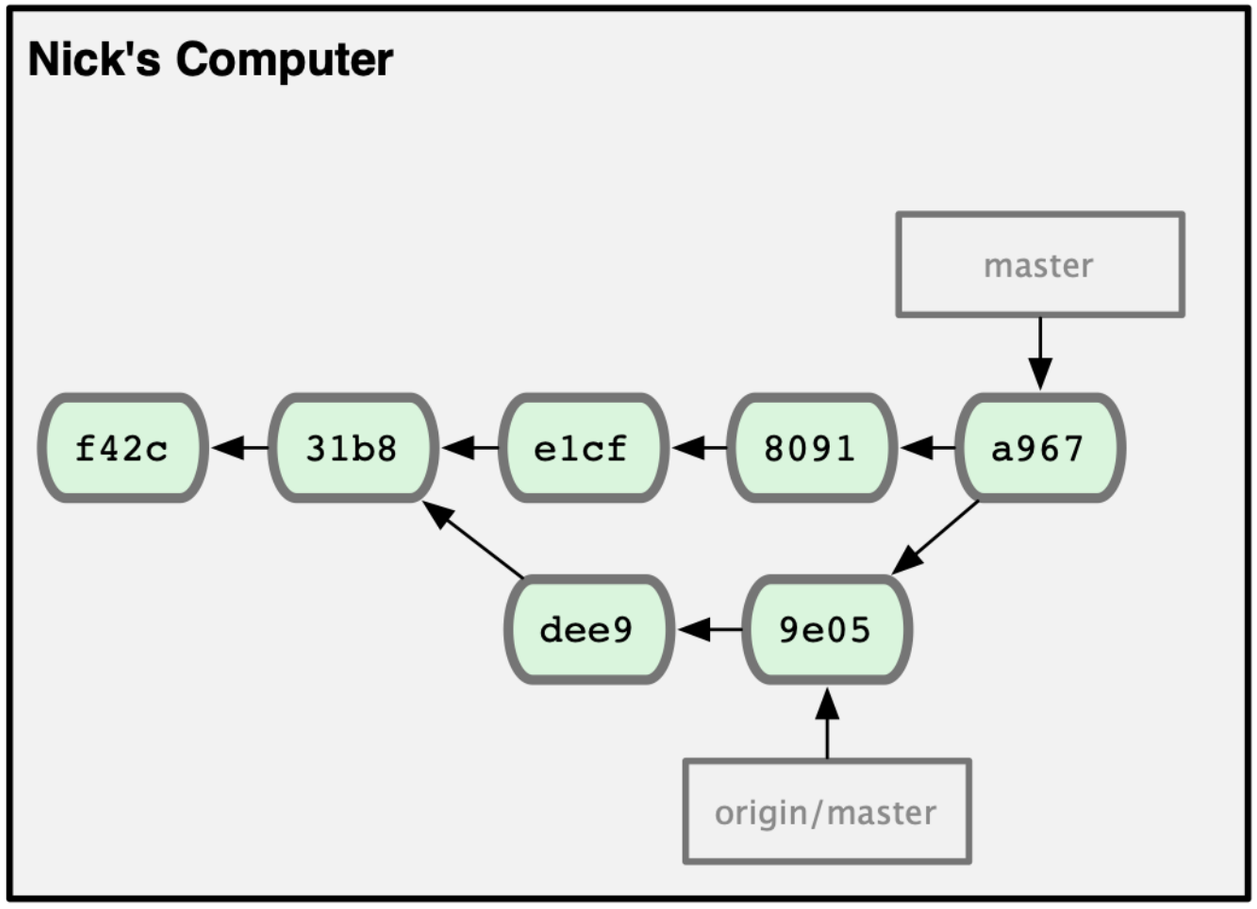
git

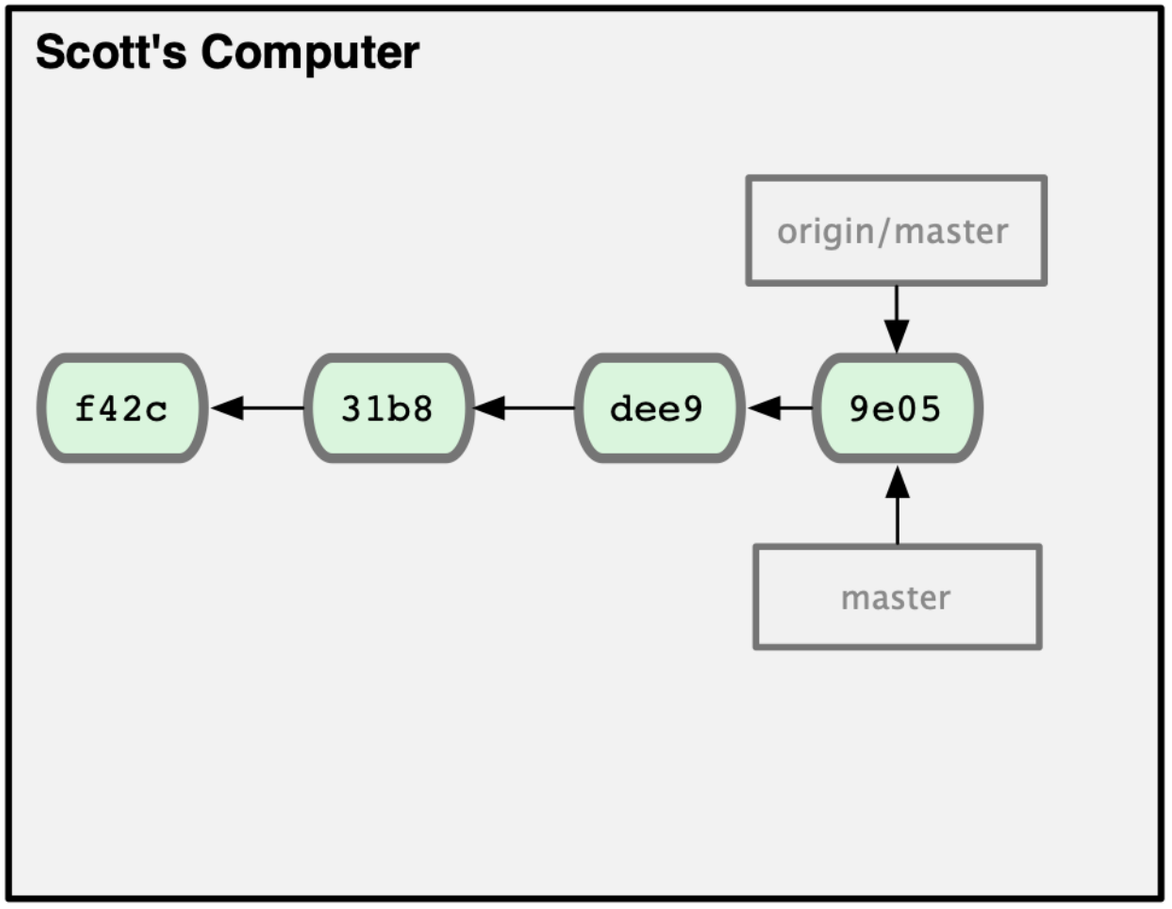


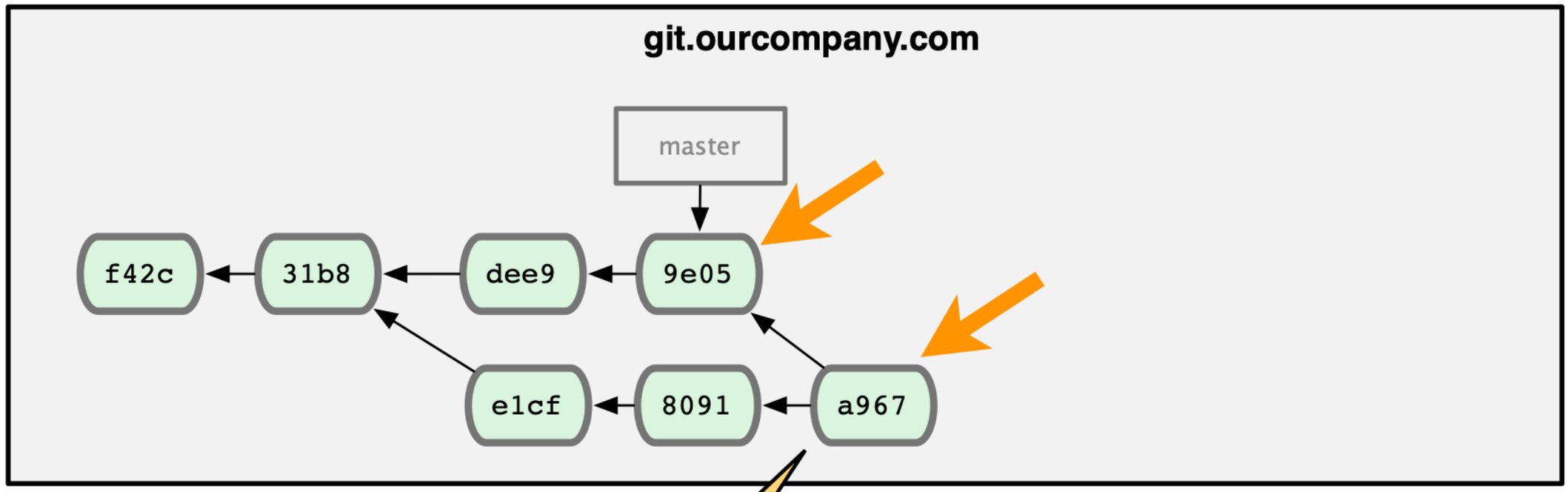




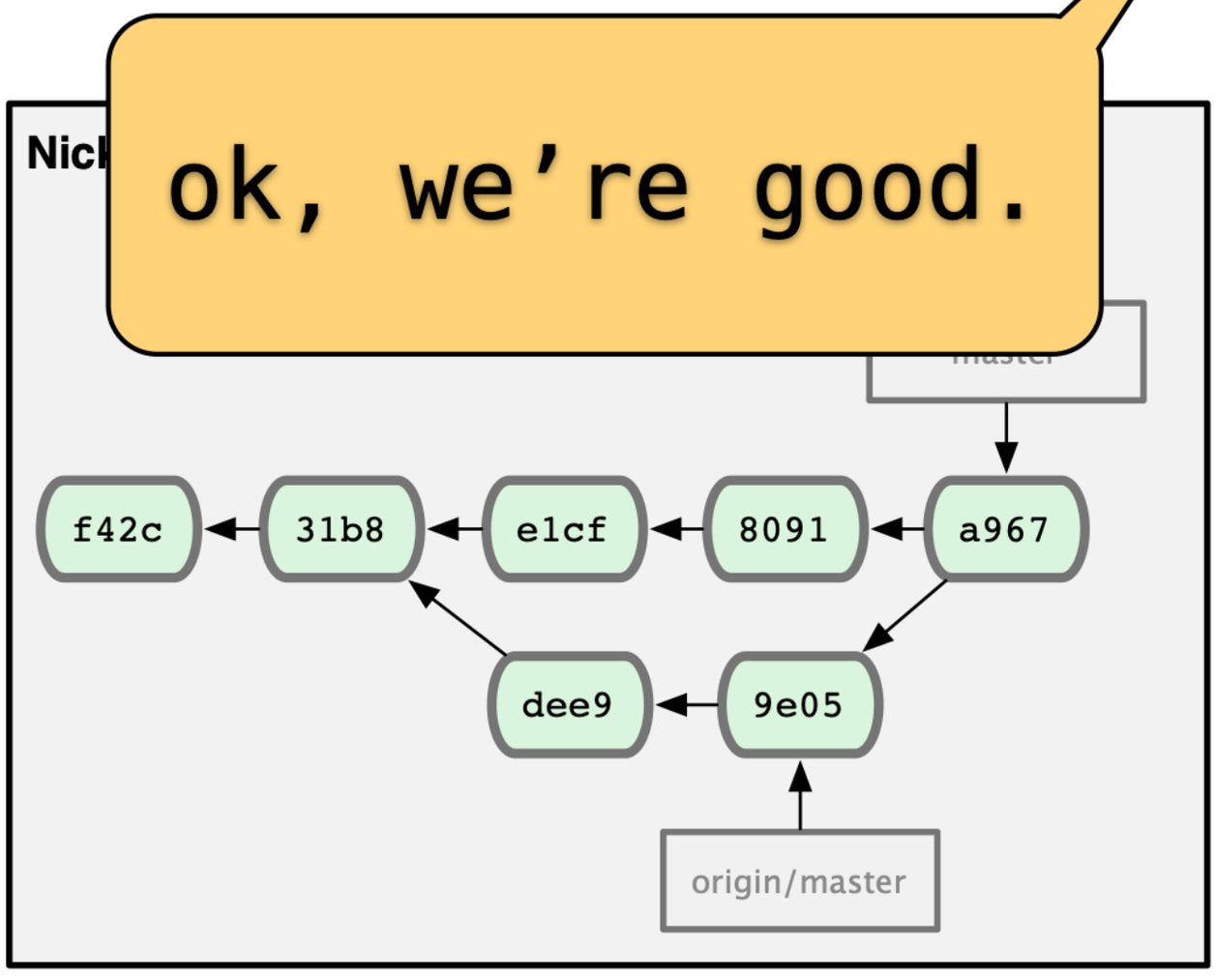

```
git push origin master
```

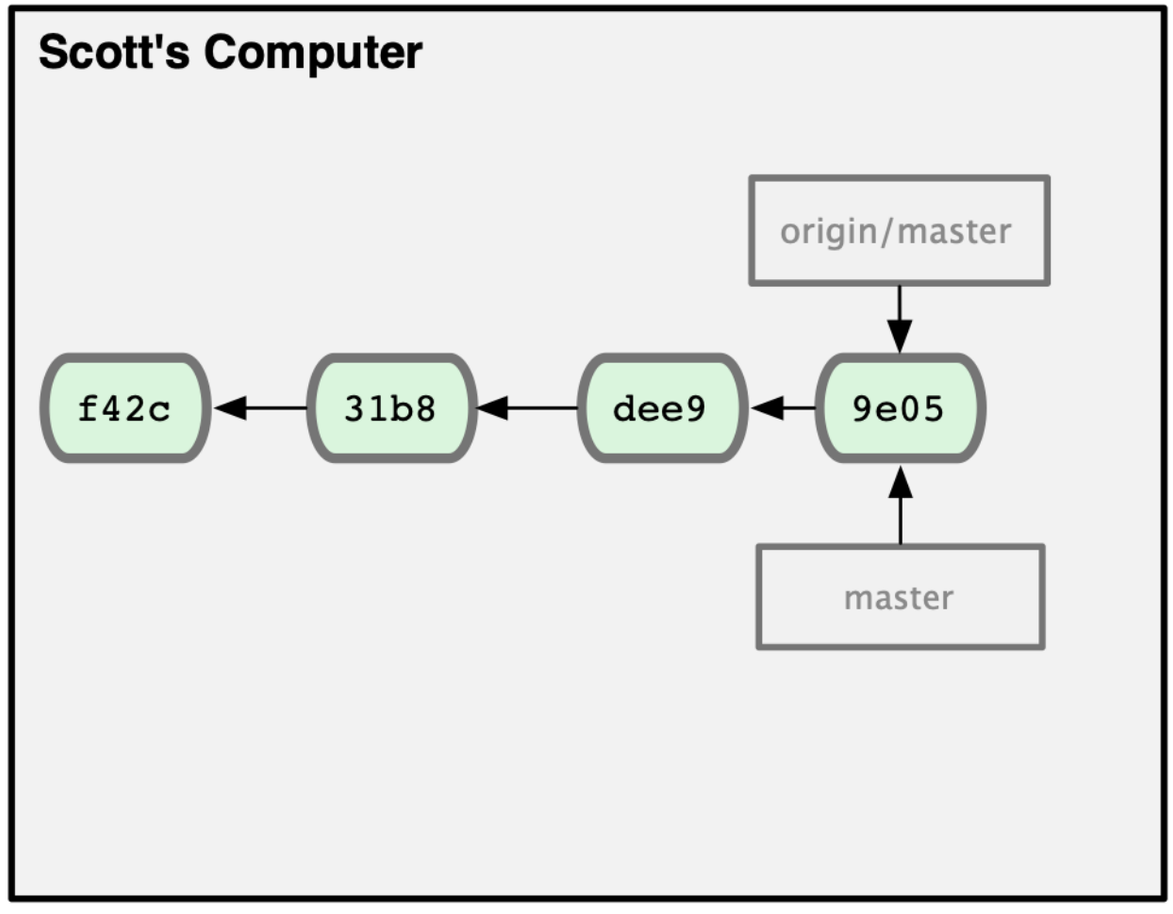


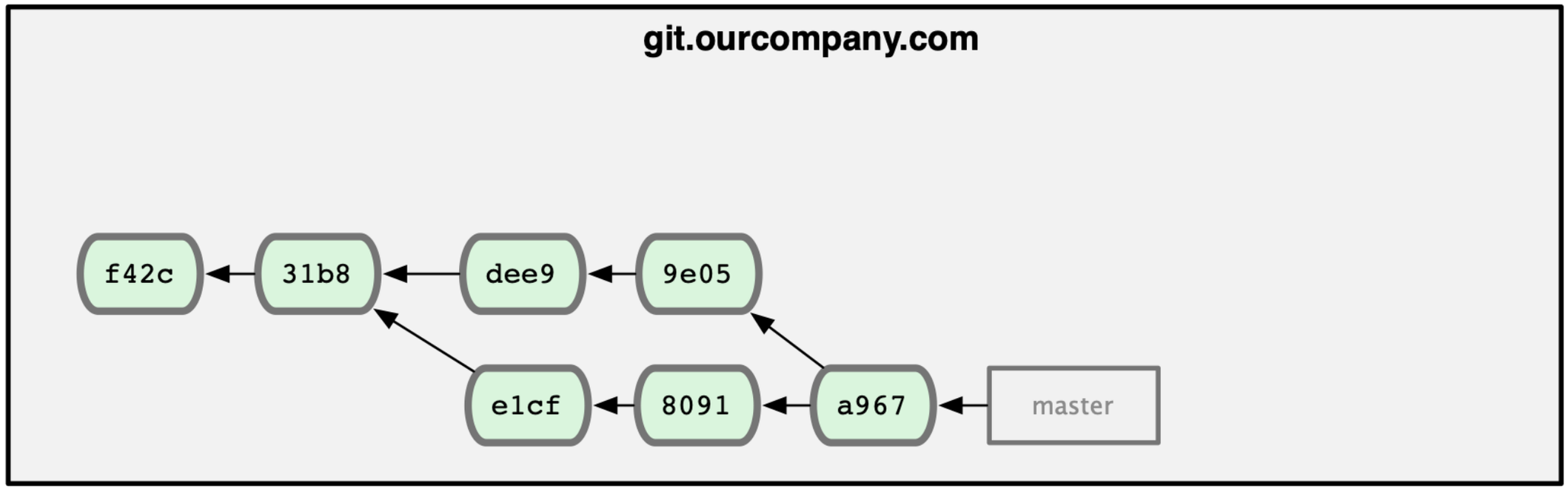


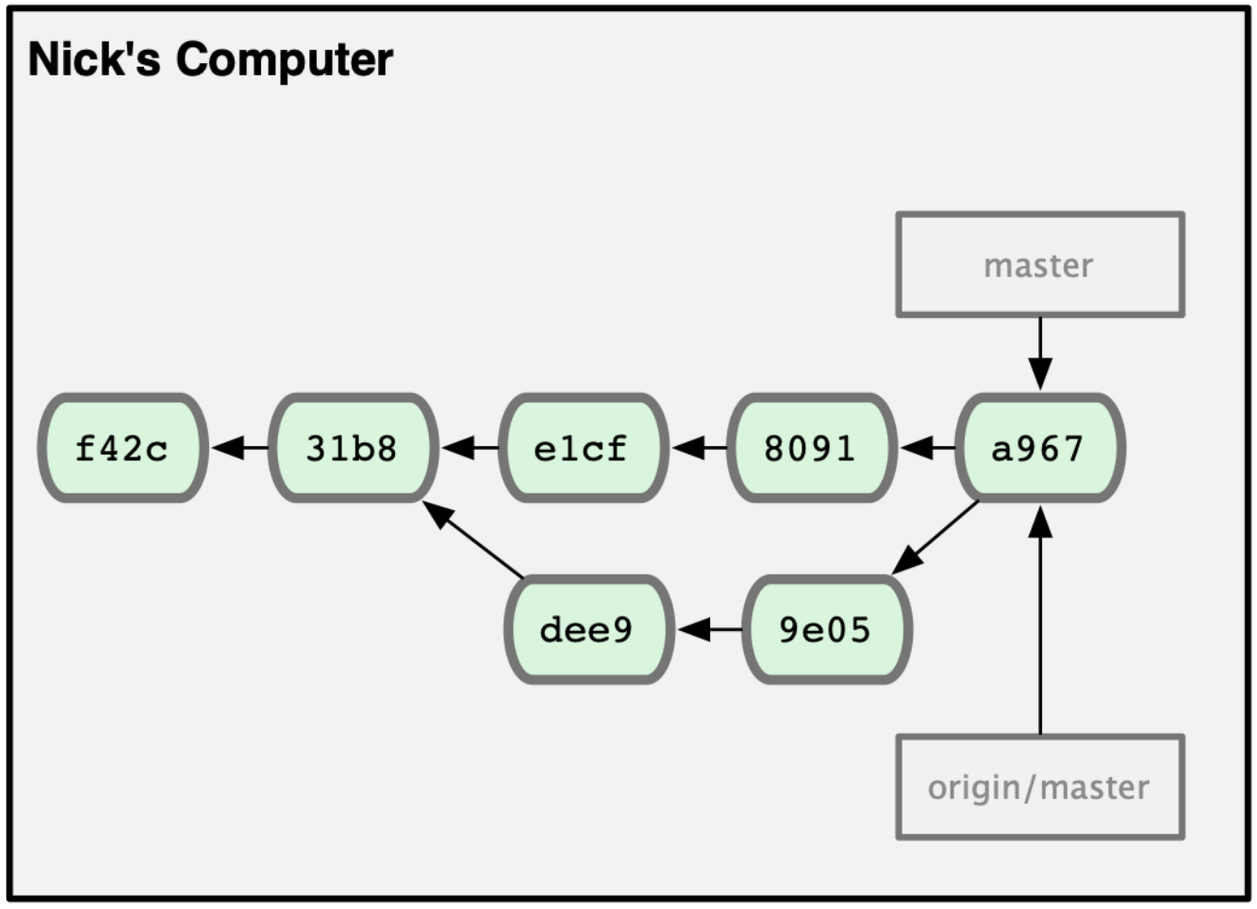

git.ourcompany.com
master
f42c
31b8
dee9
9e05
e1cf
8091
a967

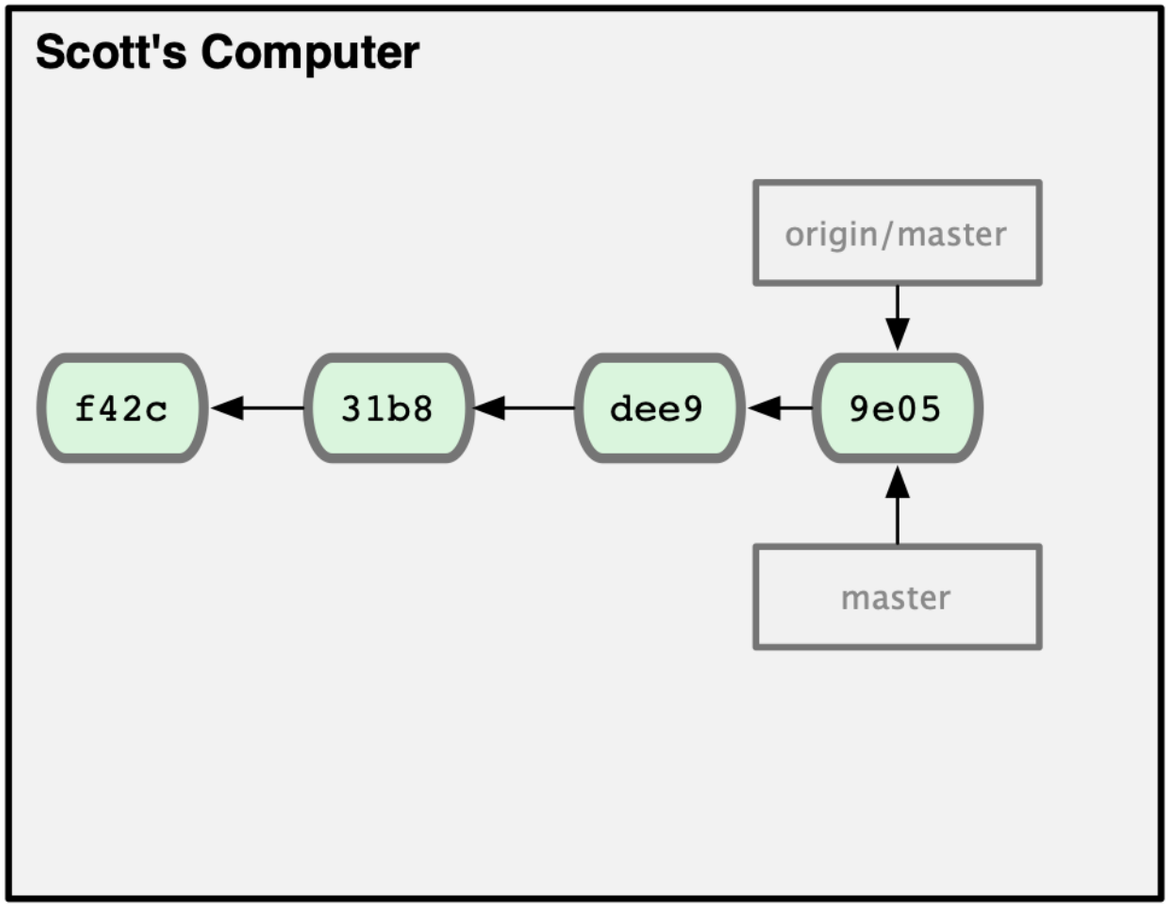
ok, we're good.


f42c
31b8
e1cf
8091
a967
dee9
9e05
origin/master


Scott's Computer
origin/master
f42c
31b8
dee9
9e05
master


git.ourcompany.com
f42c
31b8
dee9
9e05
e1cf
8091
a967
master

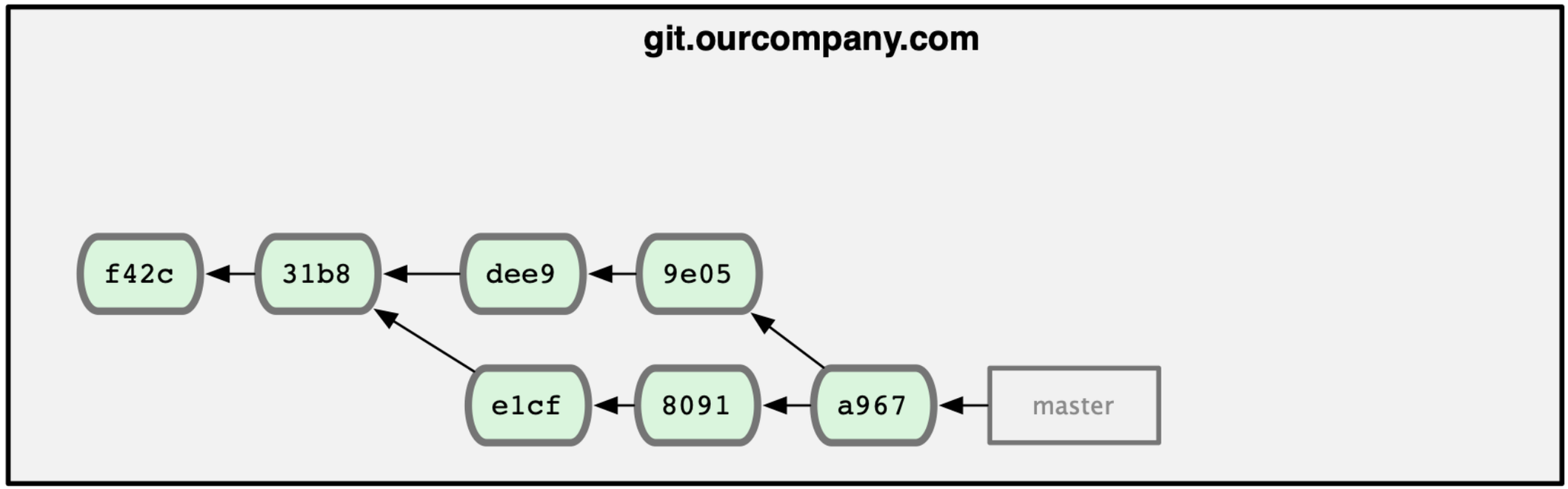

Nick's Computer
master
f42c
31b8
e1cf
8091
a967
dee9
9e05
origin/master


Scott's Computer
origin/master
f42c
31b8
dee9
9e05
master

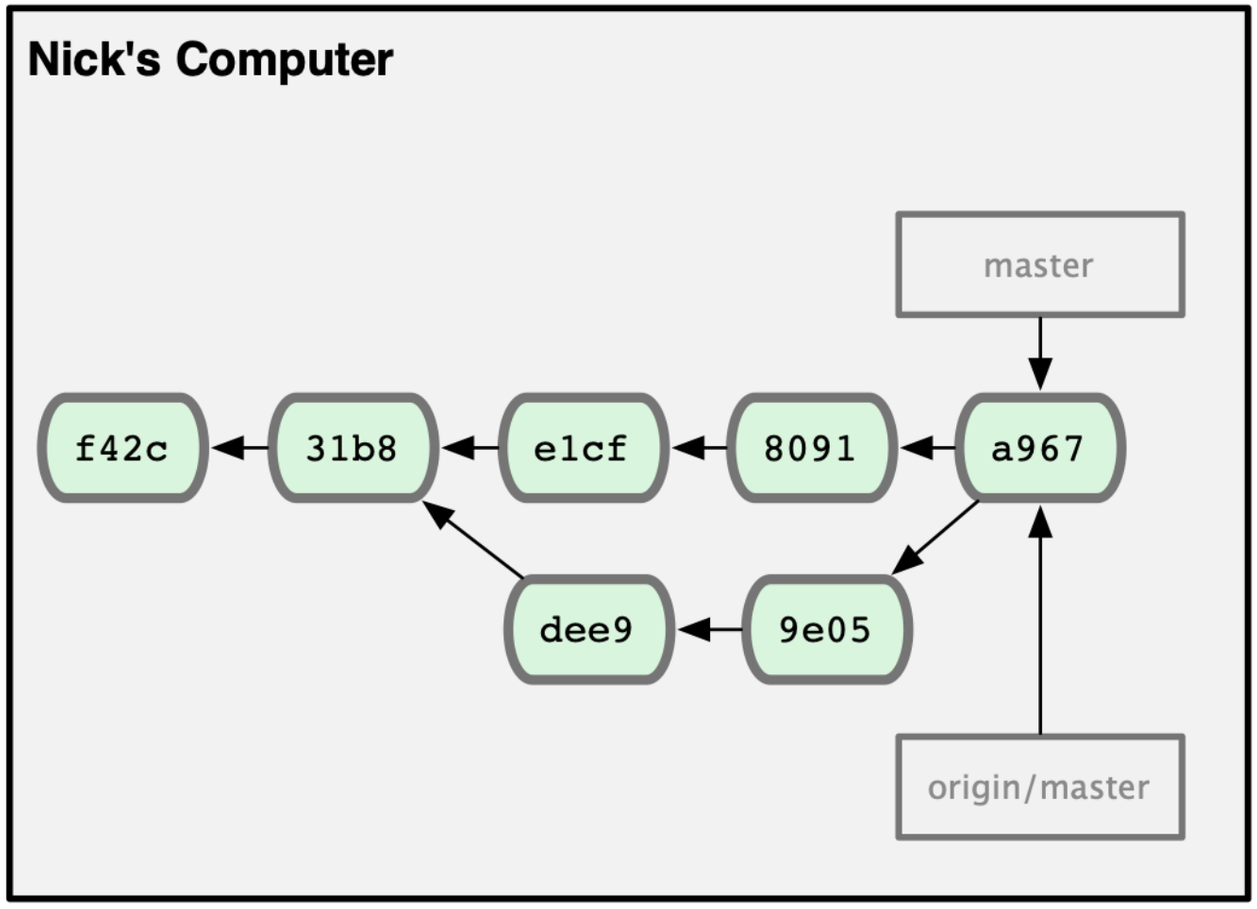
BRANCH E REMOTE

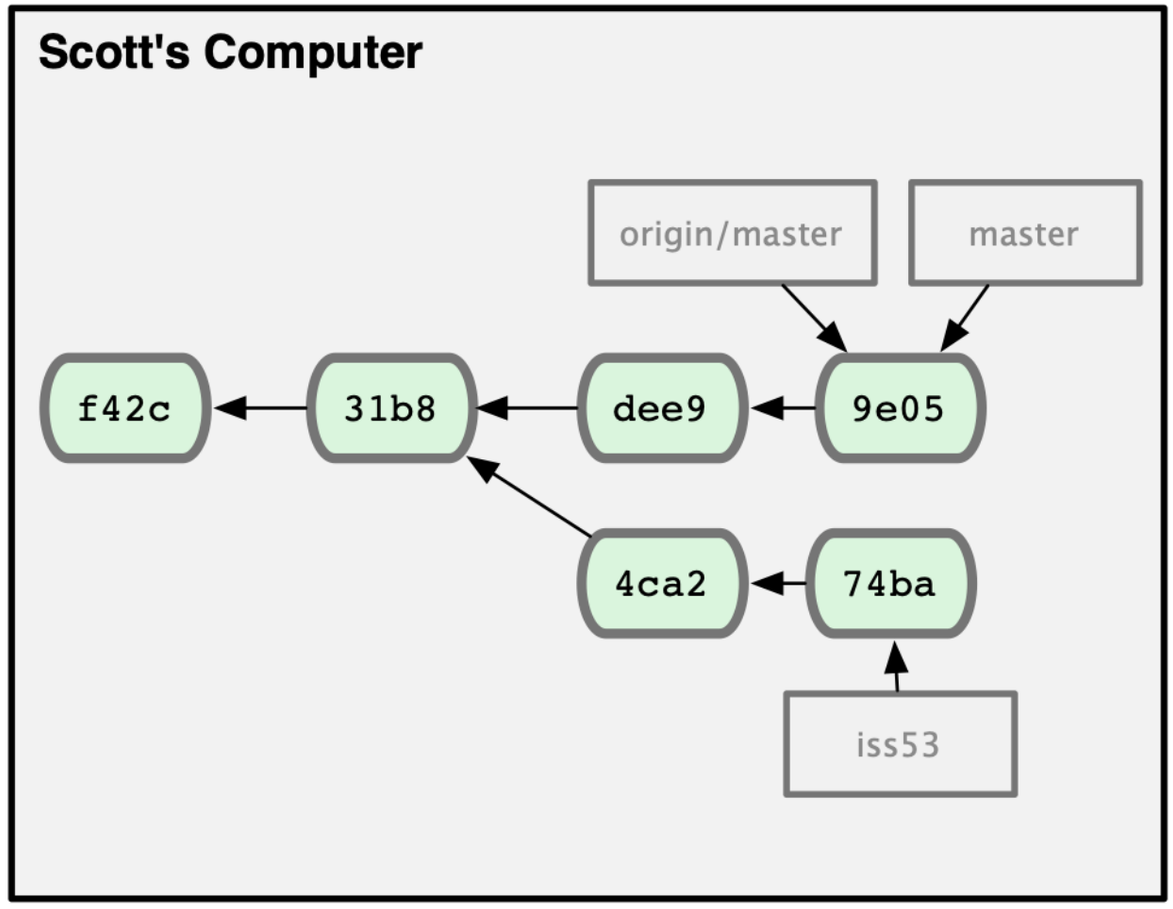
Branch creati localmente devono essere esportati esplicitamente su un repository remoto:

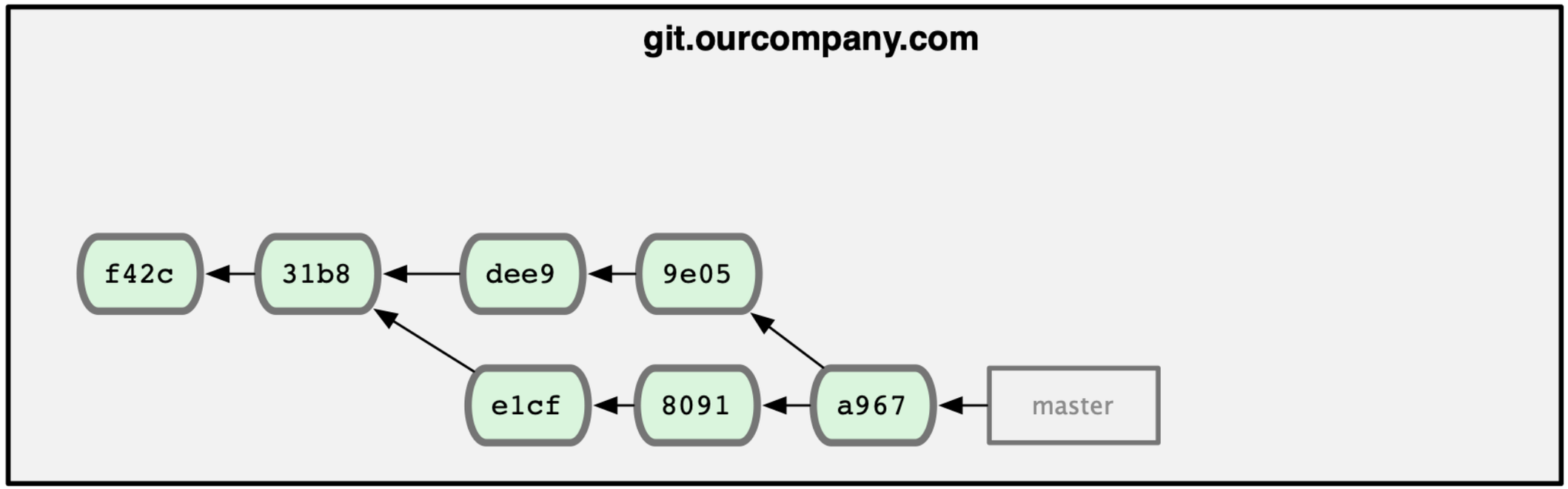
```
git push <remote> <branch>
```



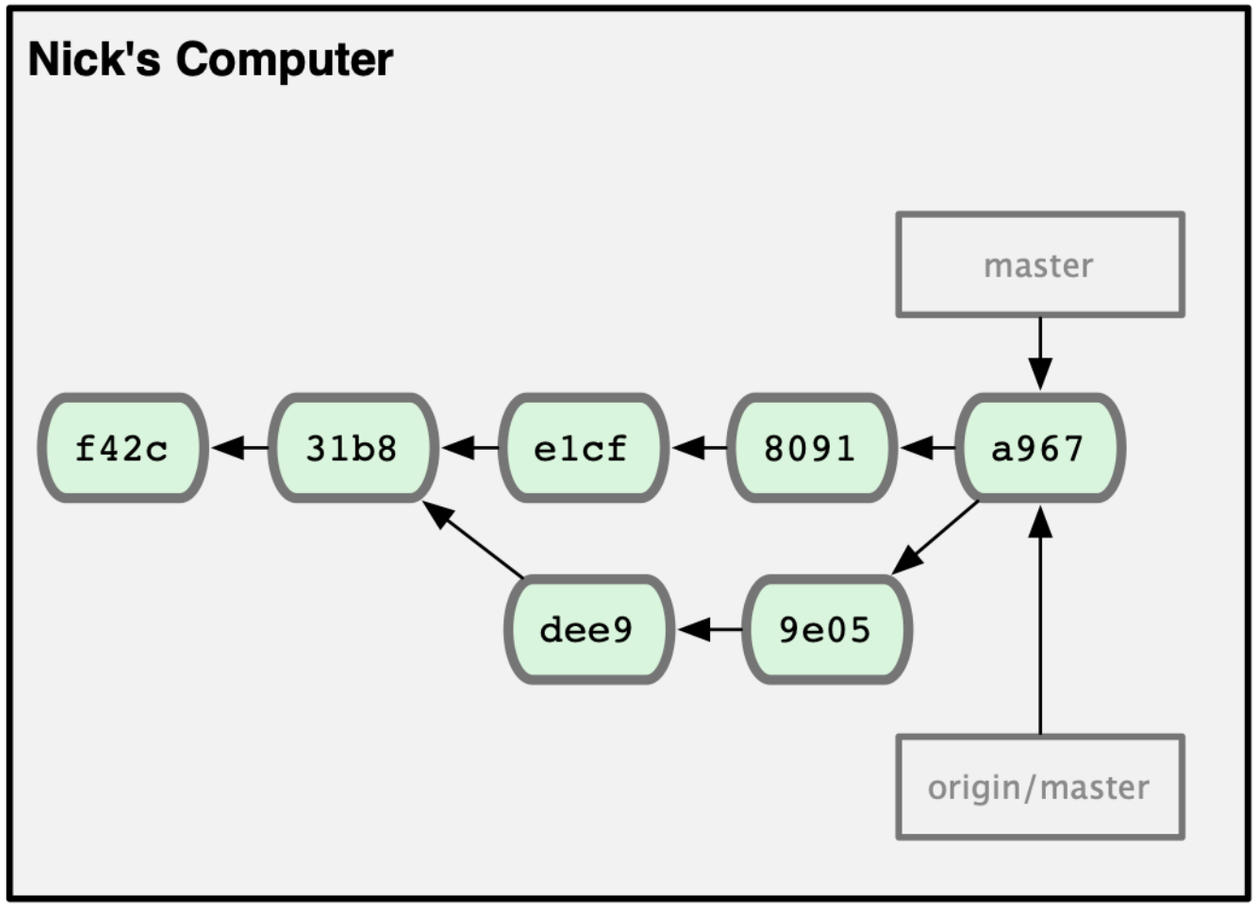

```
git checkout -b iss53 31b8; git commit; git commit
```

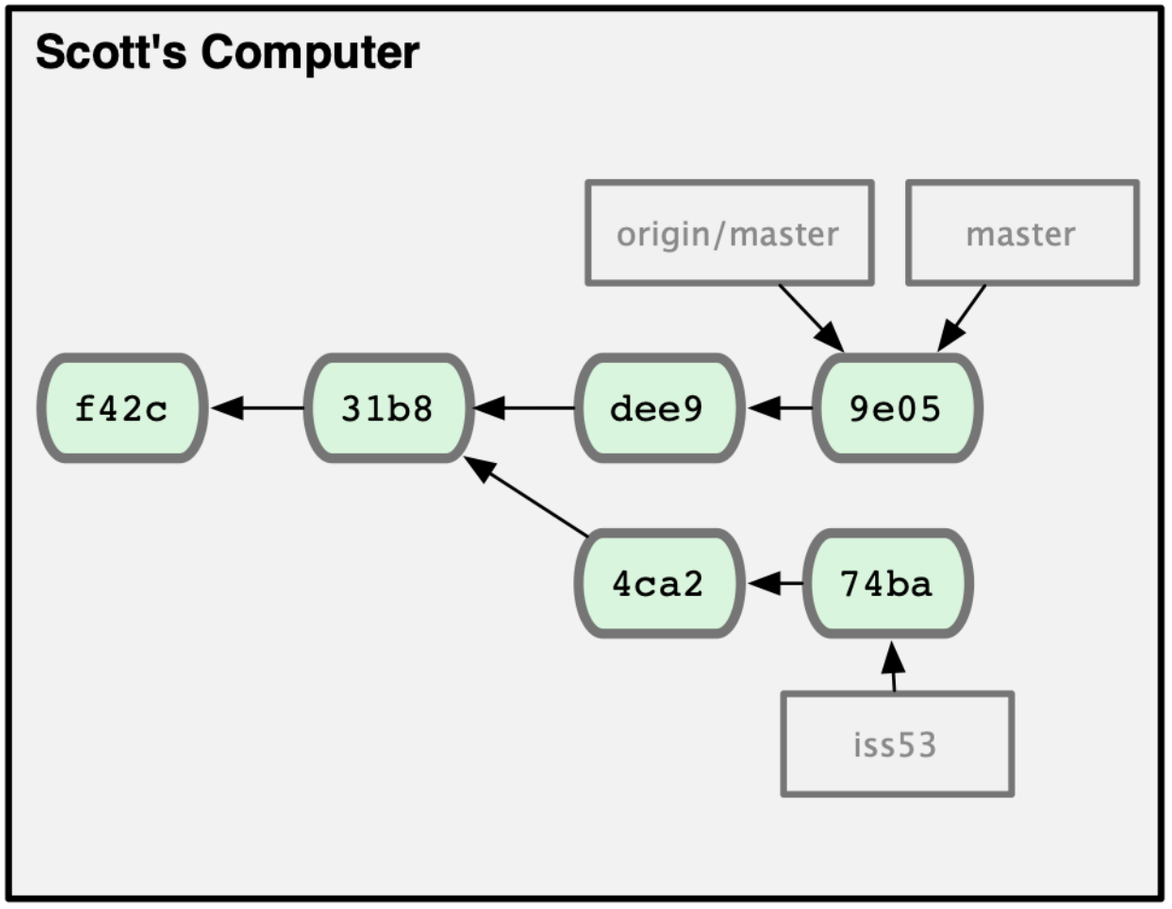


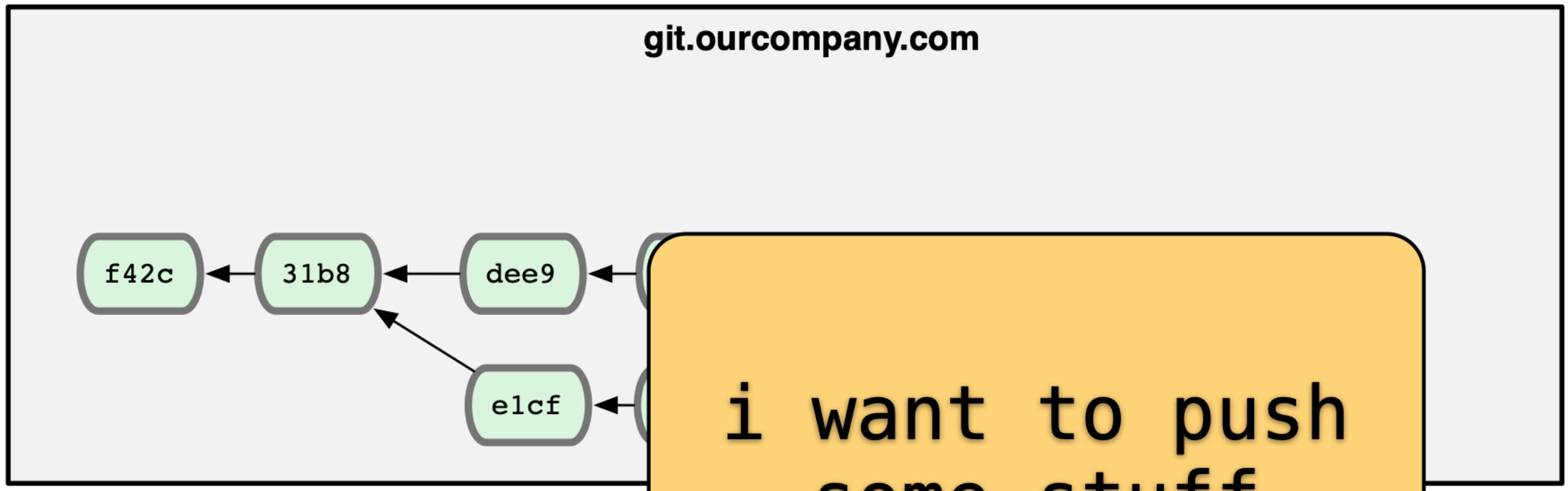


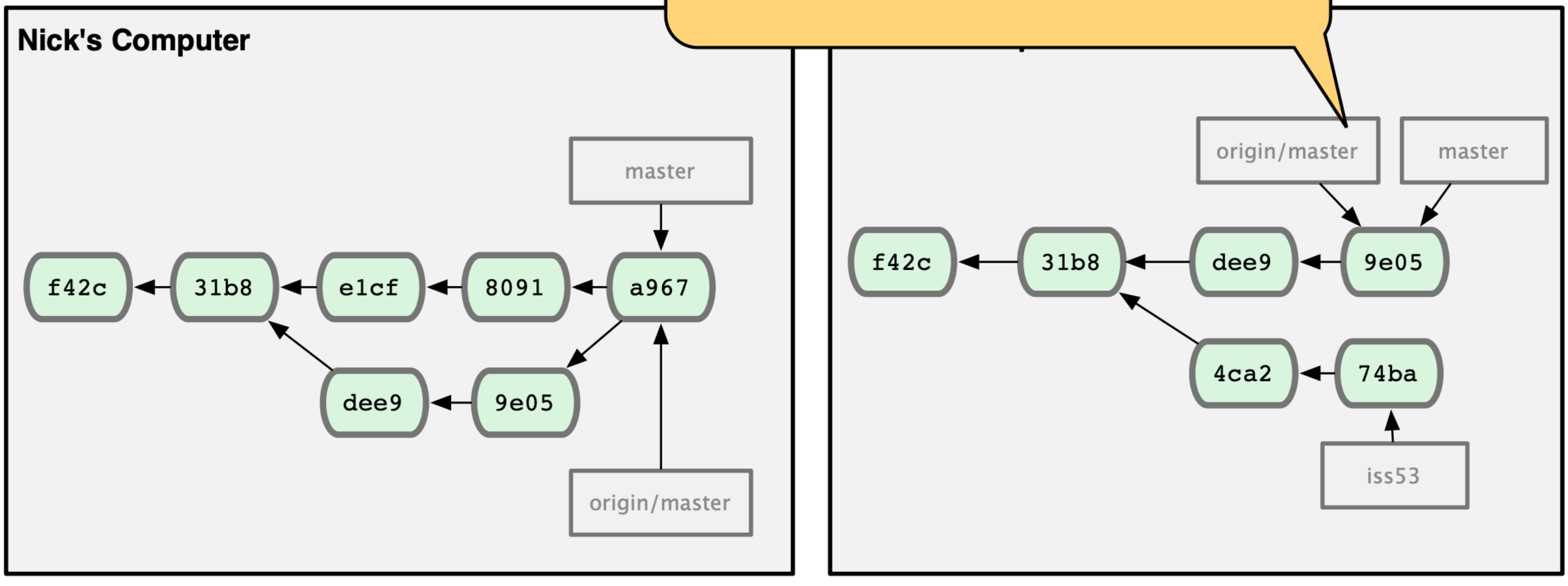


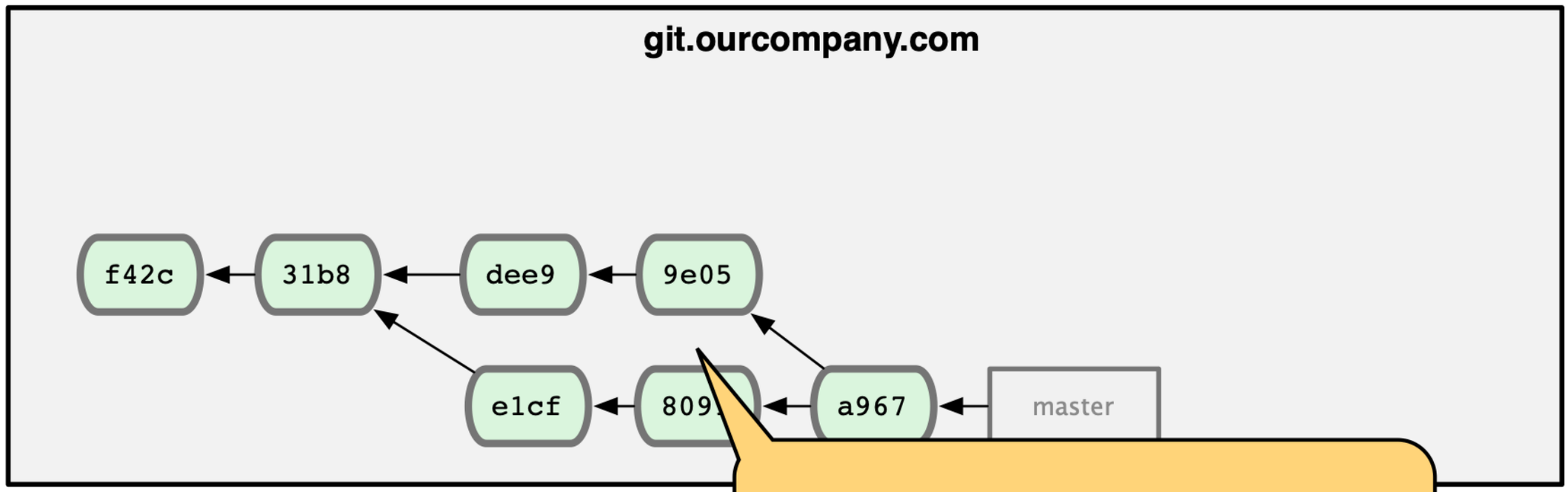

```
git push origin iss53
```





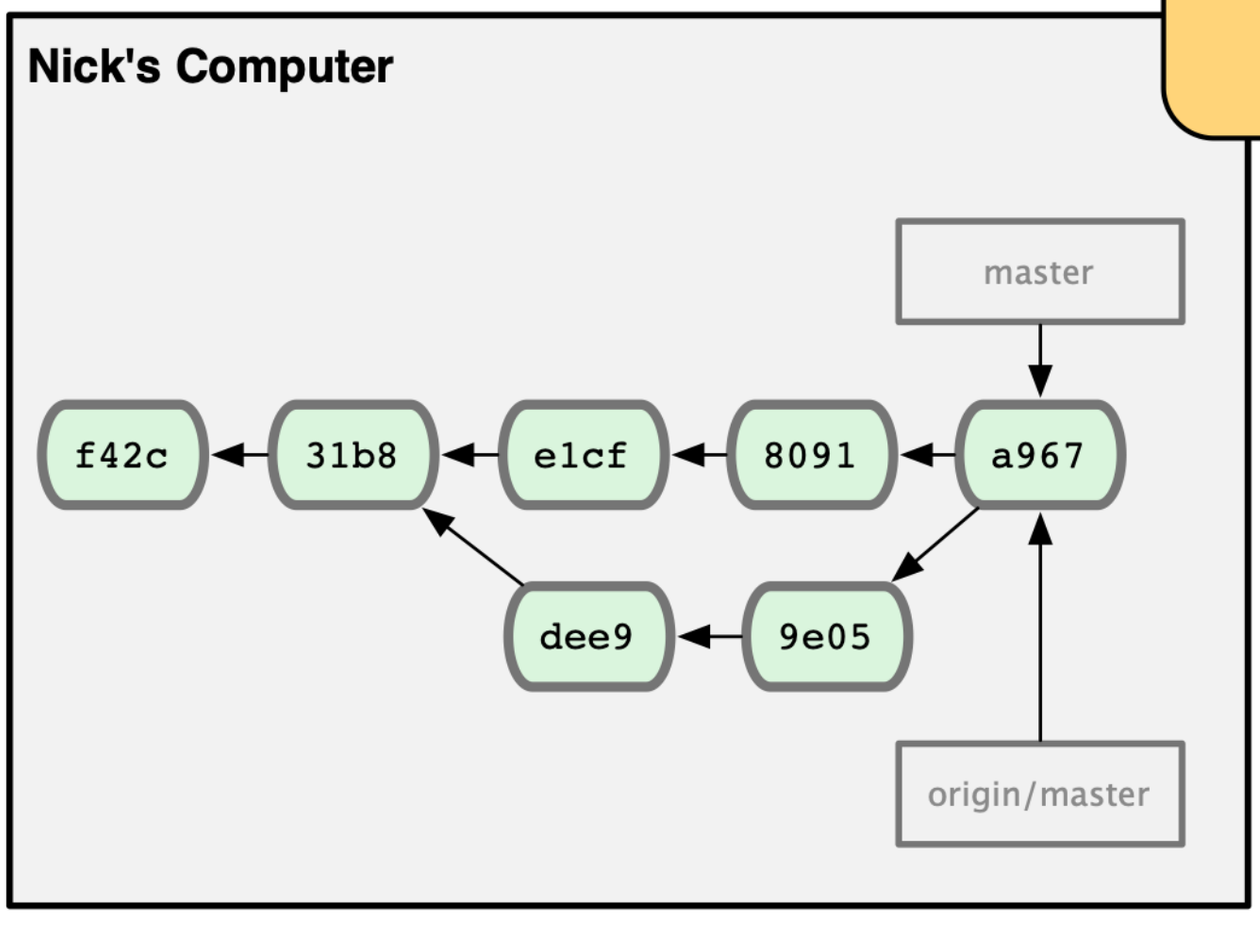

git.ourcompany.com
f42c
31b8
dee9
e1cf
i want to push
some stuff
iss53

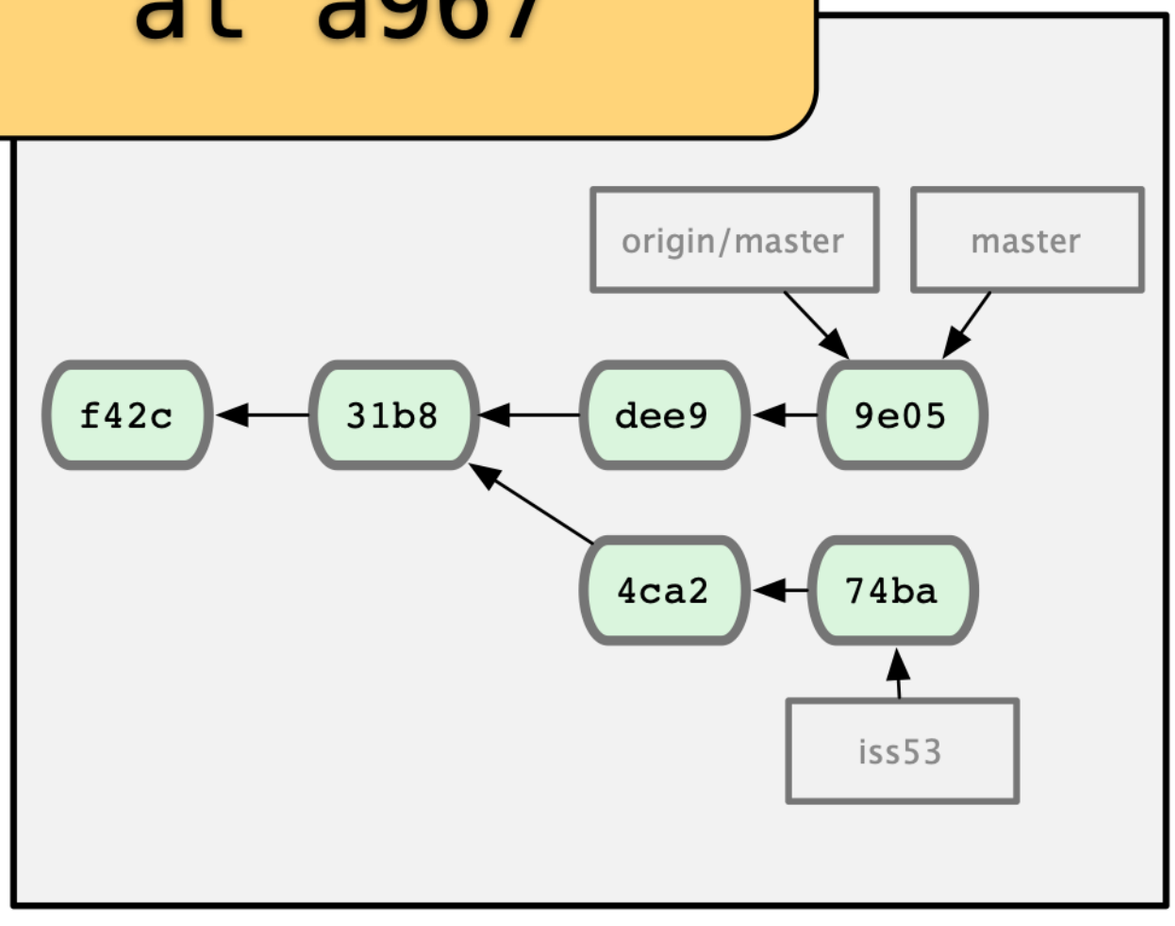

Nick's Computer
master
f42c
31b8
e1cf
8091
a967
dee9
9e05
origin/master
origin/master
master
f42c
31b8
dee9
9e05
4ca2
74ba
iss53

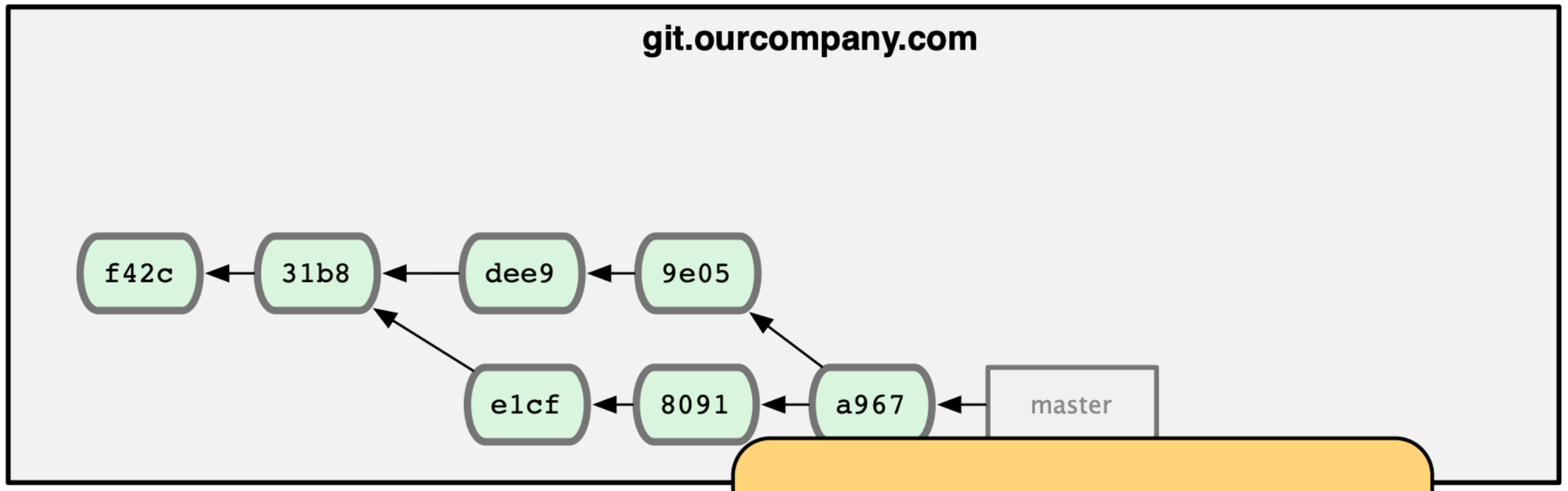


i've got master at a967

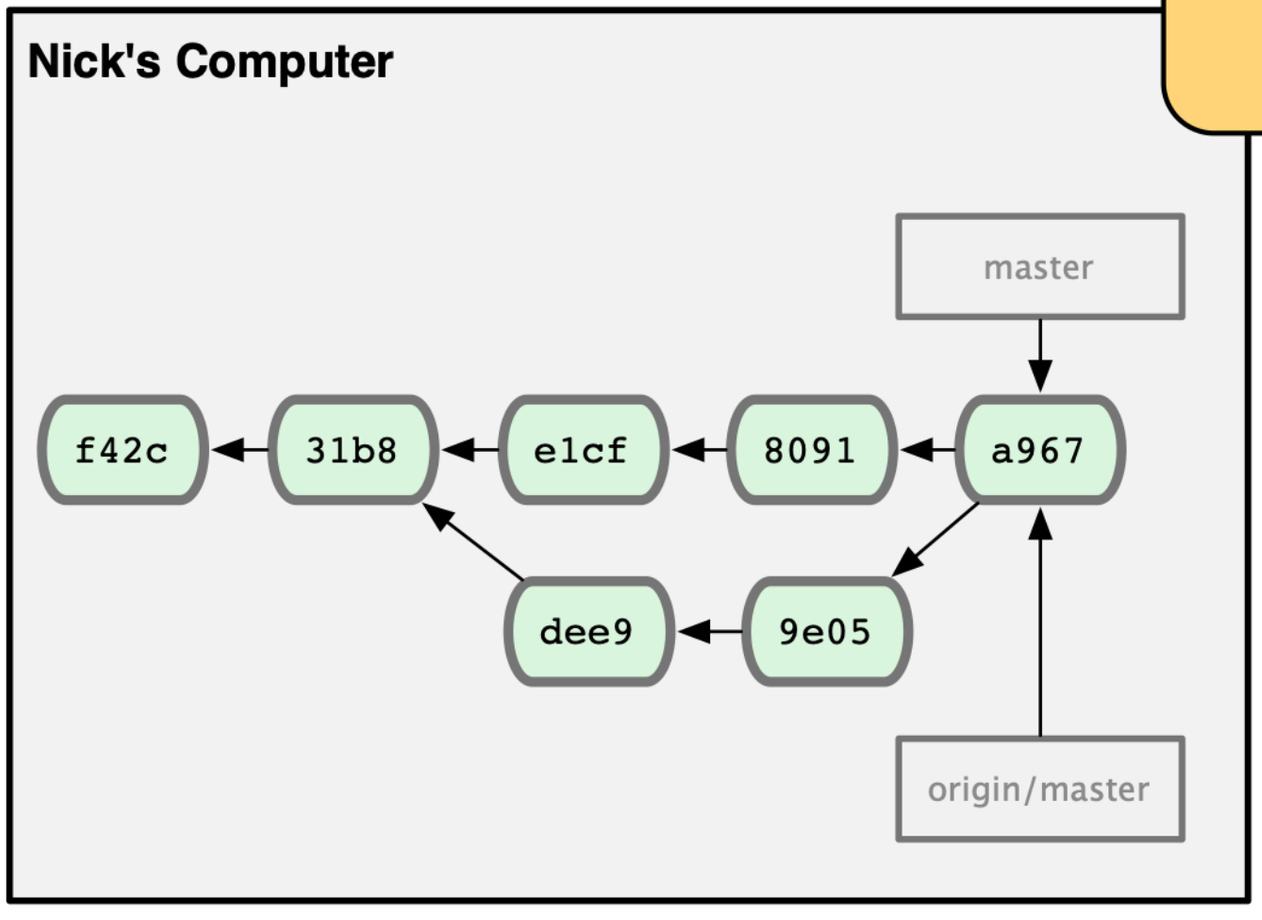
iss53

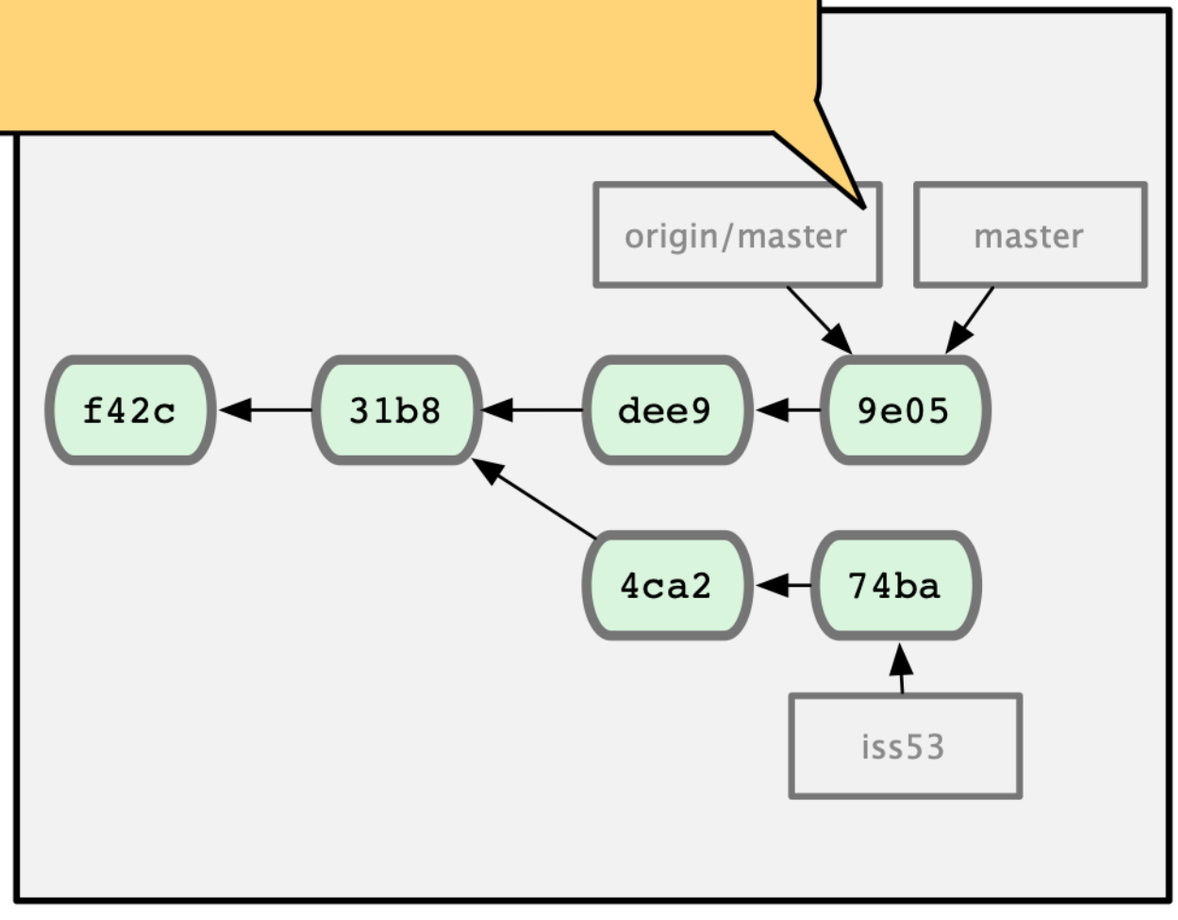


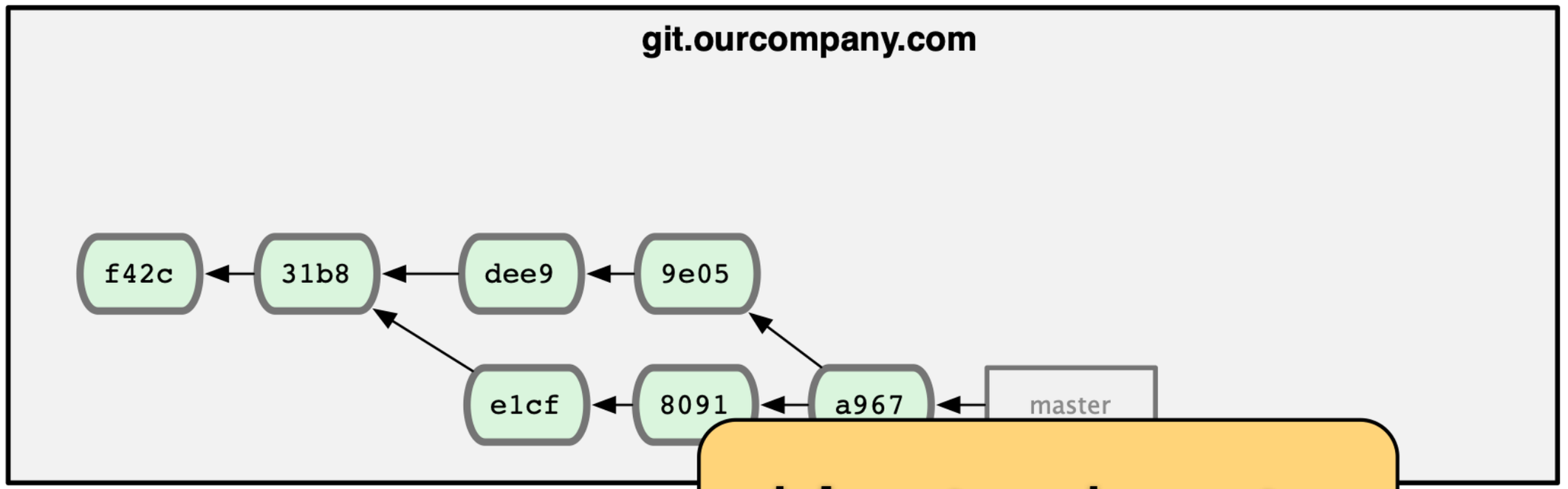



git.ourcompany.com
f42c
31b8
dee9
9e05
e1cf
8091
a967
master

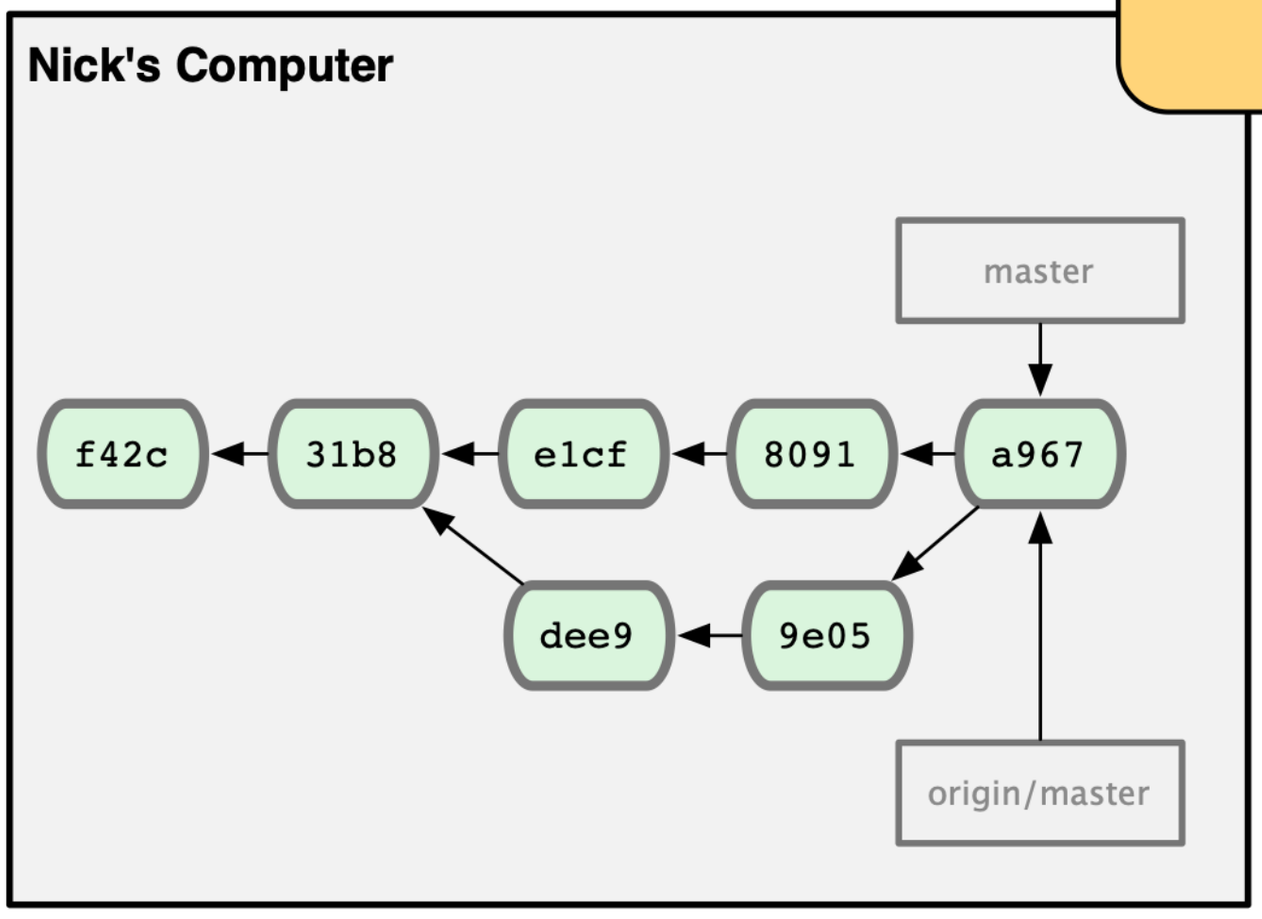
i don’t care

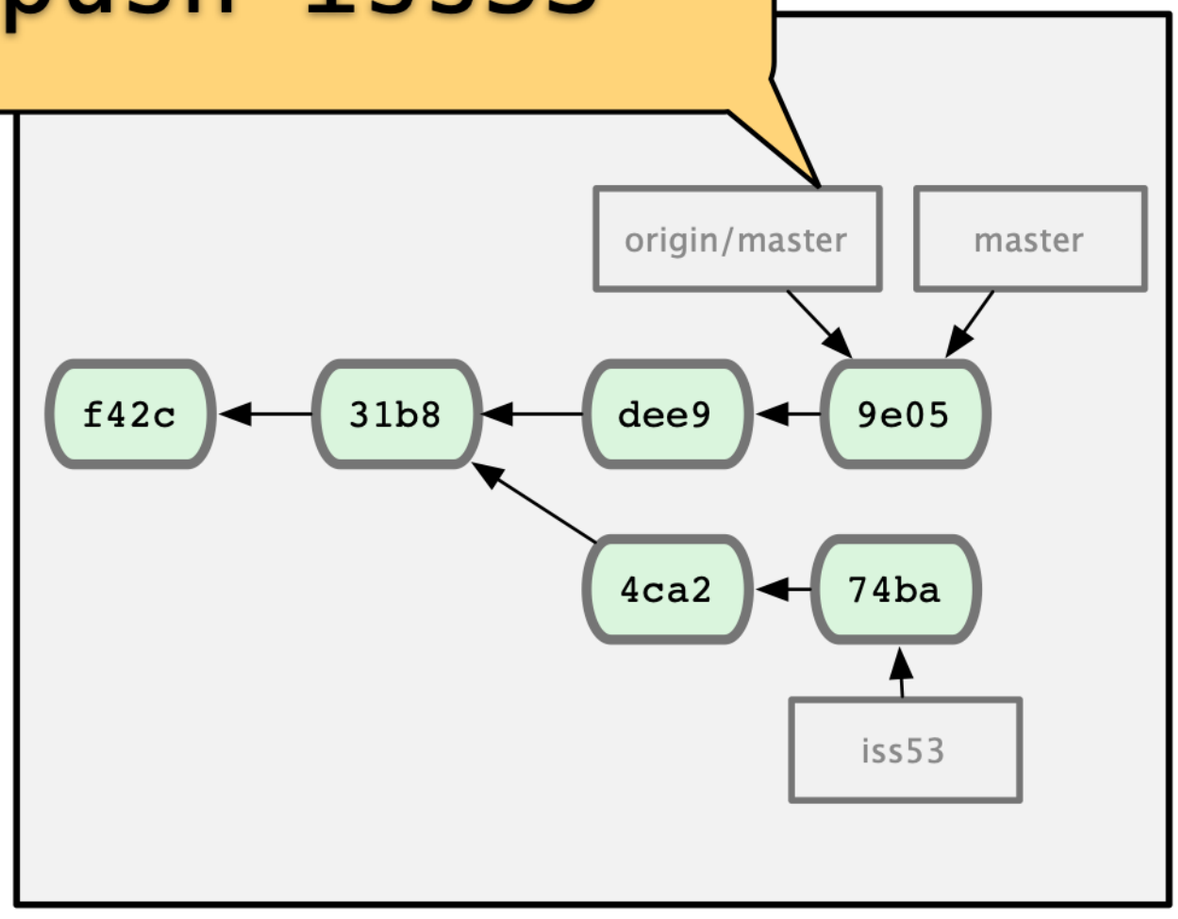

Nick's Computer
master
f42c
31b8
e1cf
8091
a967
dee9
9e05
origin/master

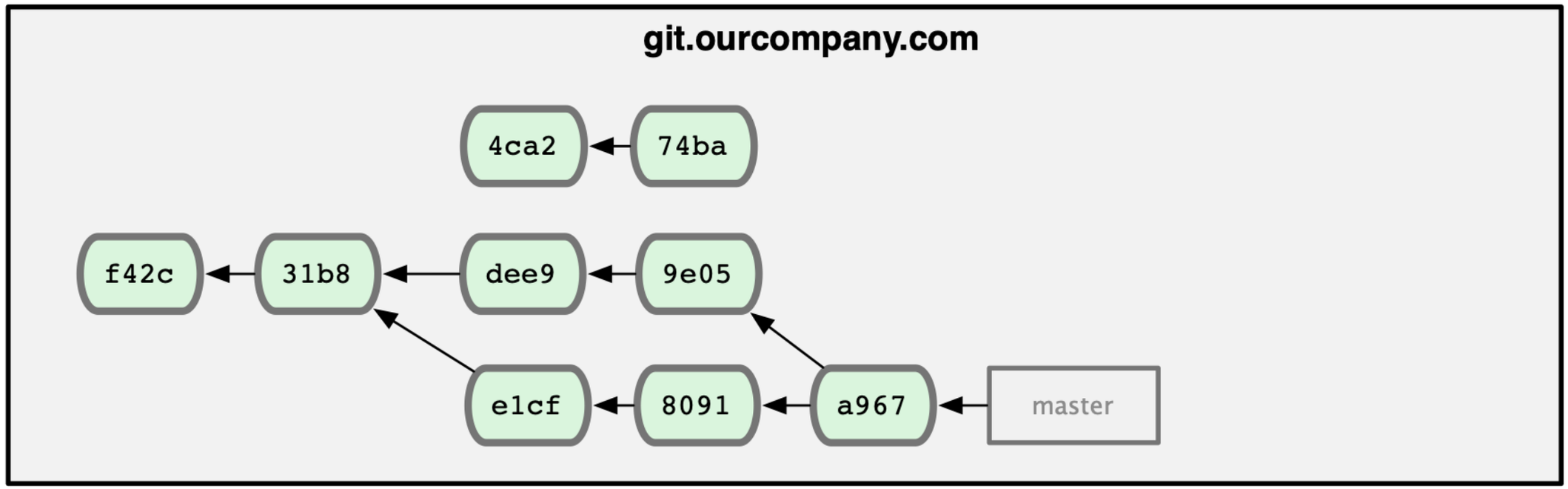

origin/master
master
f42c
31b8
dee9
9e05
4ca2
74ba
iss53



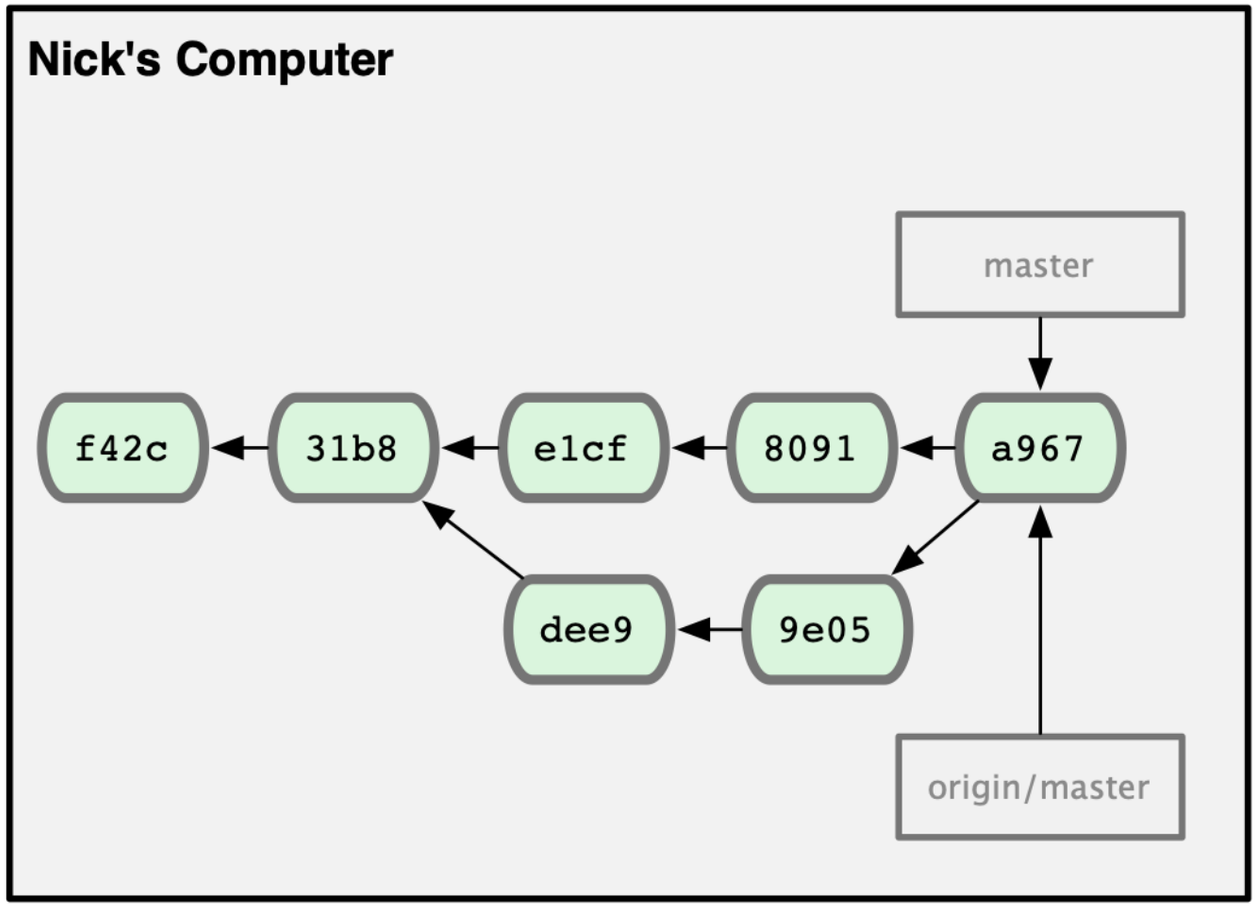
i'm trying to push iss53

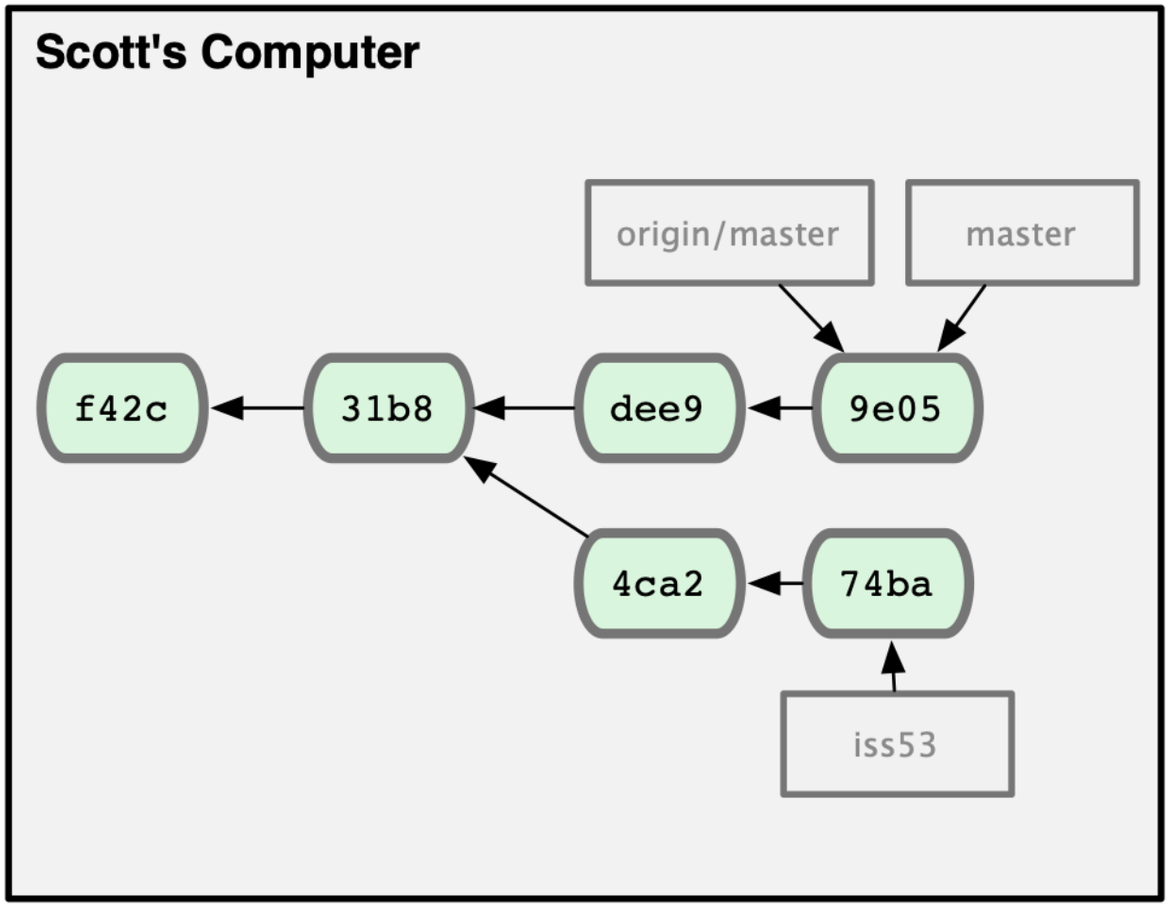


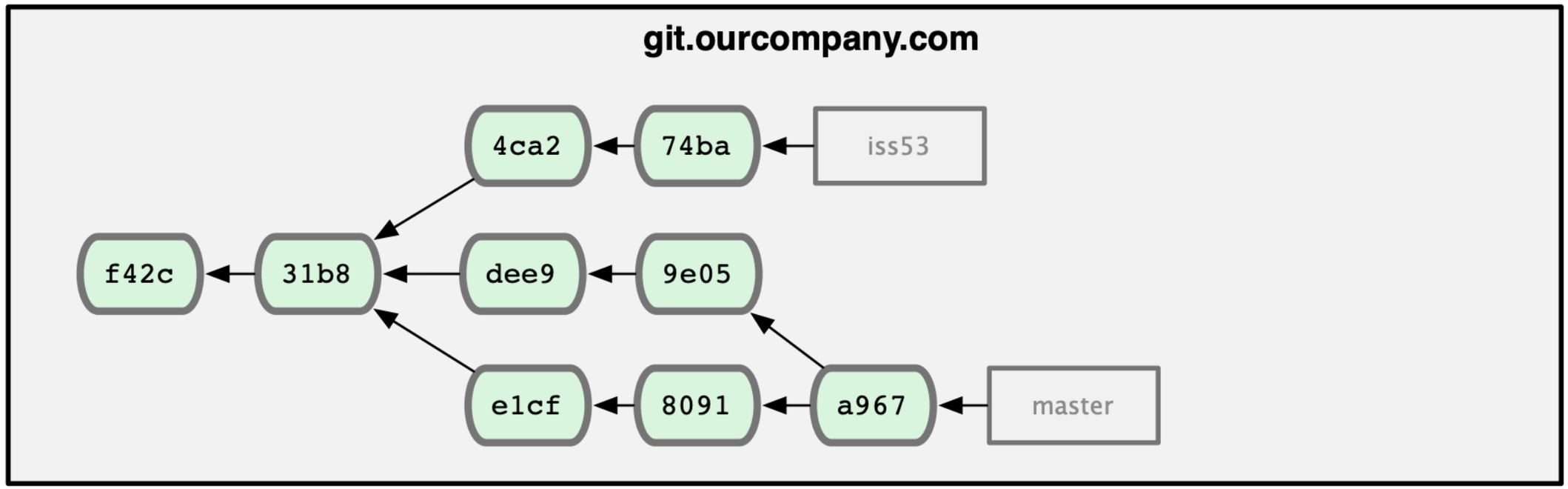


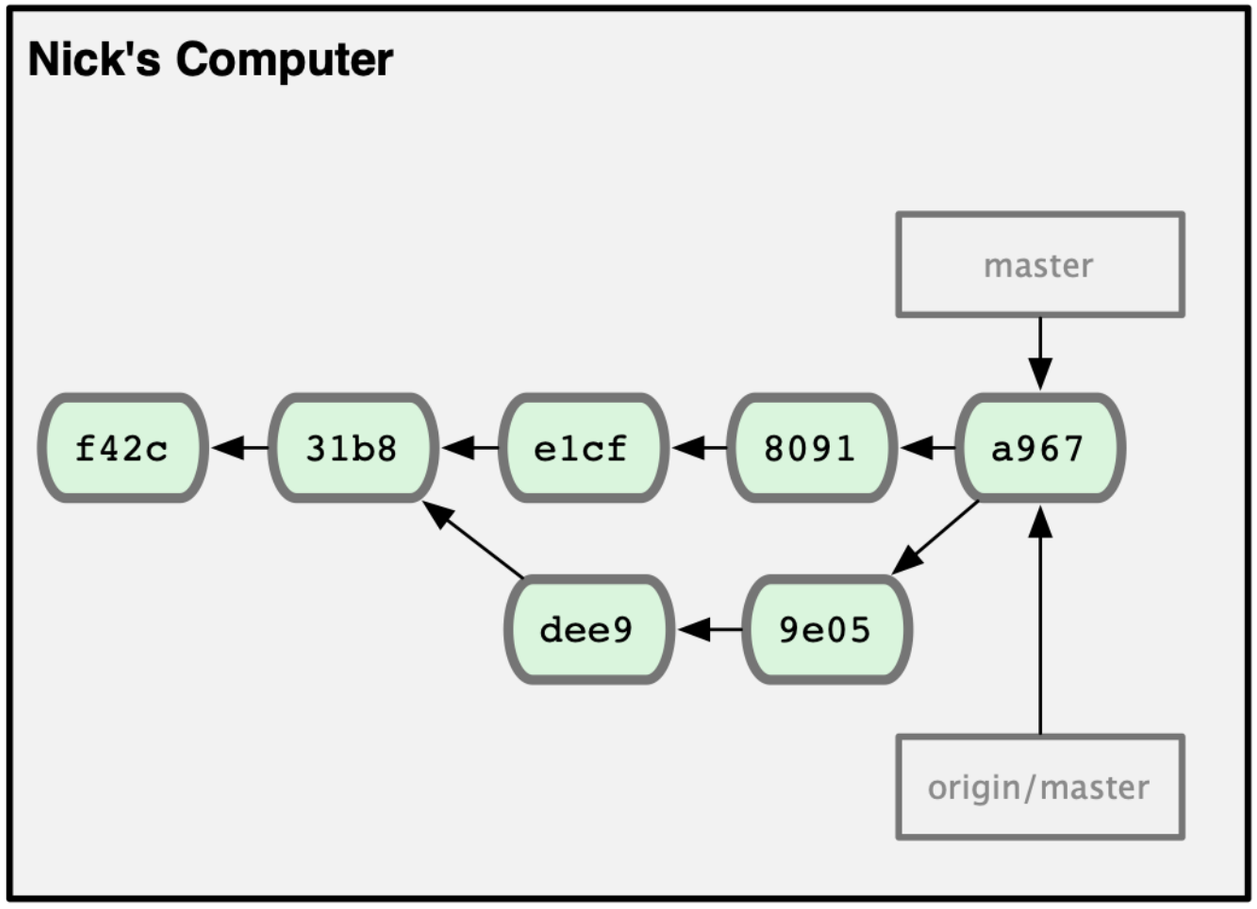


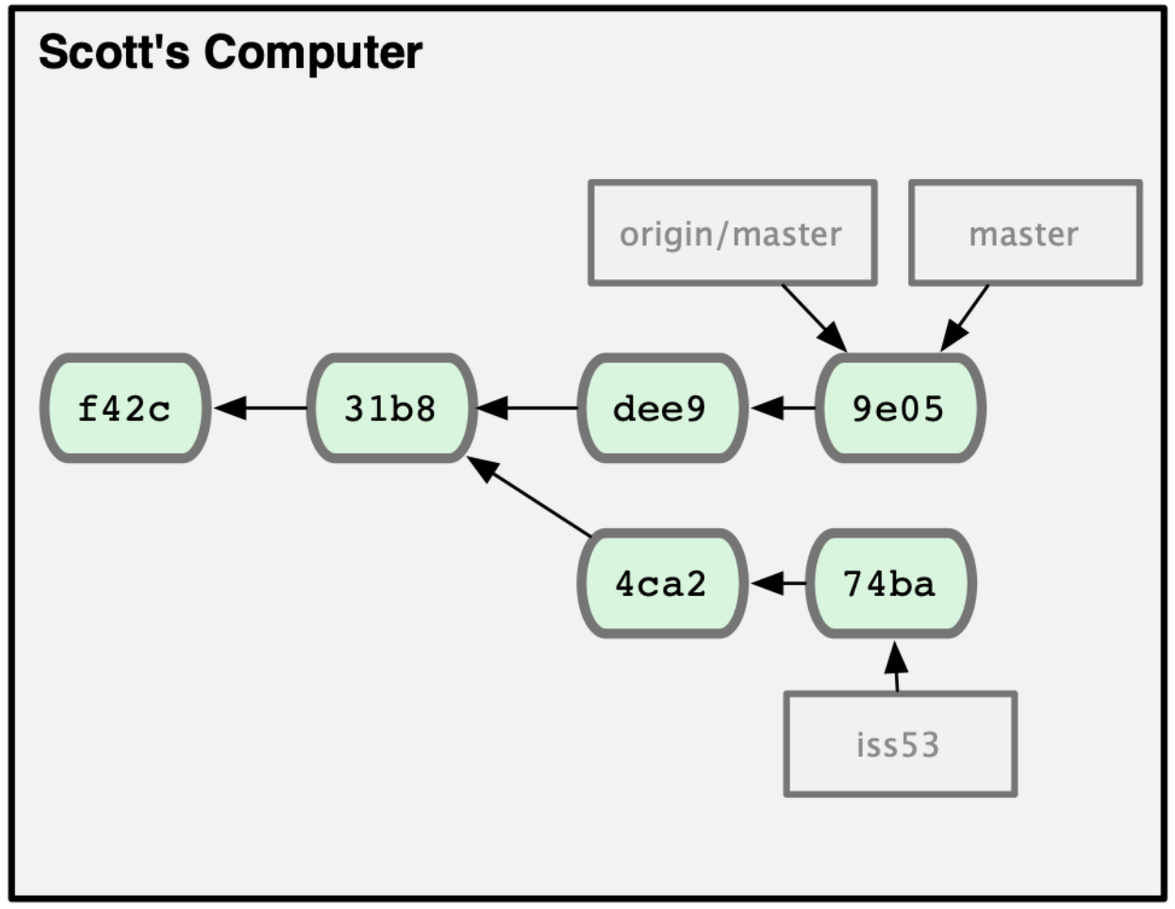

```
git push origin iss53
```






git.ourcompany.com
4ca2
74ba
iss53
f42c
31b8
dee9
9e05
e1cf
8091
a967
master


Nick's Computer
master
f42c
31b8
e1cf
8091
a967
dee9
9e05
origin/master


Scott's Computer
origin/master
master
f42c
31b8
dee9
9e05
4ca2
74ba
iss53

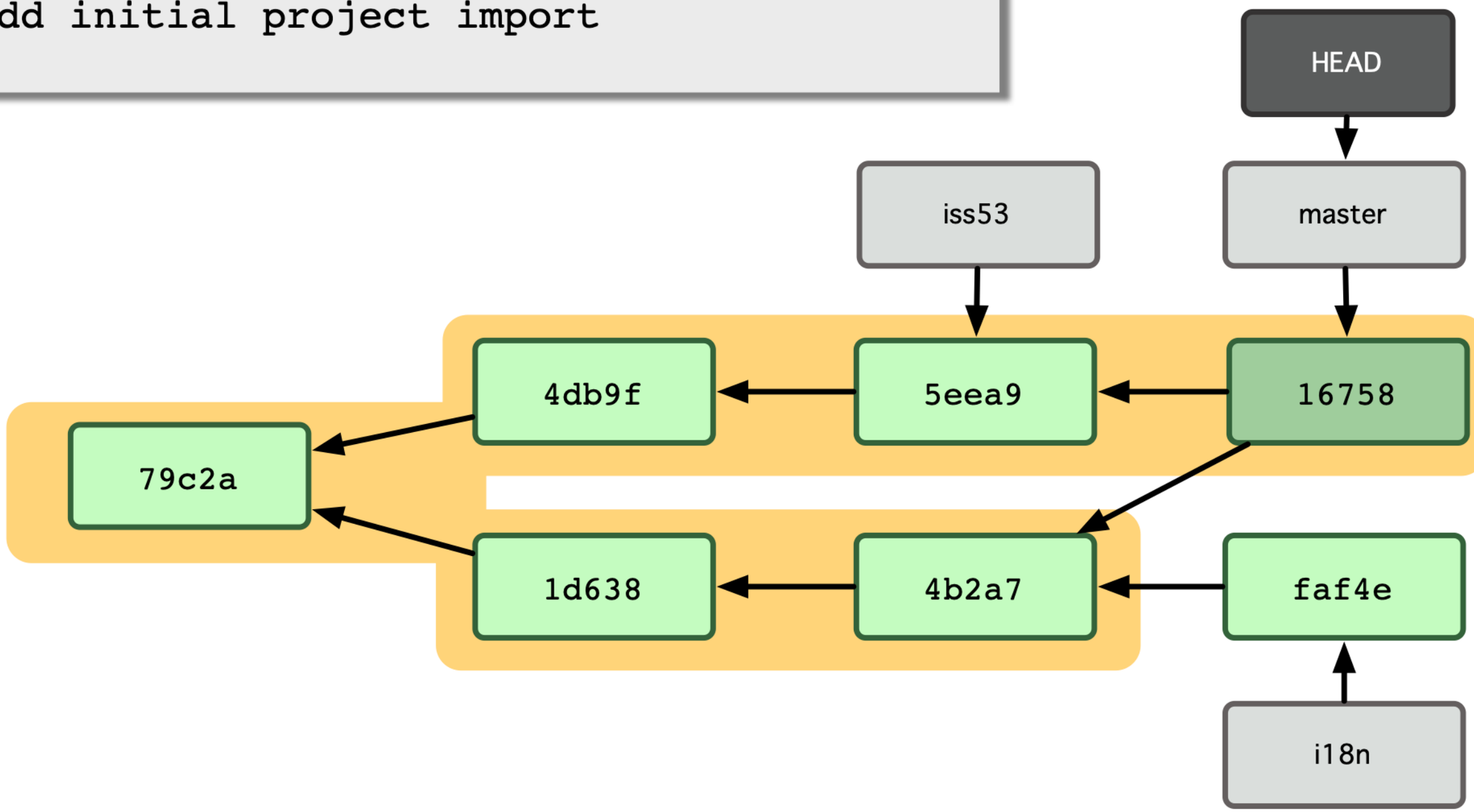
LOGS

VISUALIZZARE IL LOG

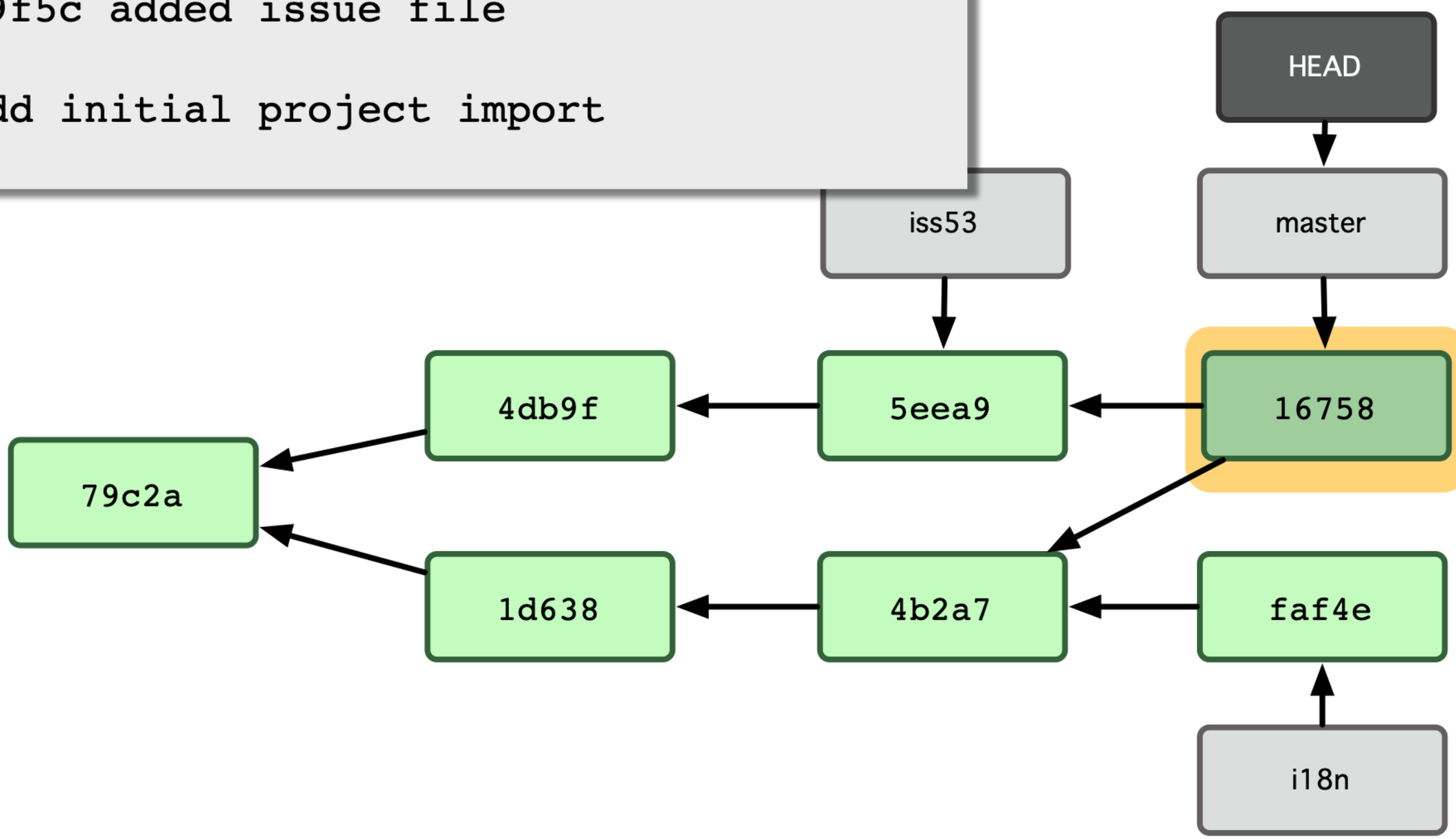
il comando git log permette di analizzare l’evoluzione del repository rispetto ad uno specifico branch

- --oneline > visualizza le entry su una singola riga
- --graph > visualizza il log con una rappresentazione a grafo
- --decorate > aggiunge ulteriori informazioni alle entry
- --pretty > permette di definire il proprio formato di visualizzazione
- --after > limita le entry visualizzate sulla base della marca temporale
- --author > limita le entry visualizzate sulla base dell’autore
- --<filename|path> > limita le entry visualizzate sulla base di un path contenuti nei relativi commit
- --all > mostra tutti i branch

```
$ git log --oneline
16758d8 Merge branch 'i18n'
4b2a7ae fix spacing issues in both c files
1d6389c added i18n file
5eea9cf documented issue file
4db9f5c added issue file
79c2add initial project import
```



```
$ git log --oneline --graph
*   16758d8 Merge branch 'i18n'
|\
| * 4b2a7ae fix spacing issues in both c files
| * 1d6389c added i18n file
* | 5eea9cf documented issue file
* | 4db9f5c added issue file
|/
* 79c2add initial project import
```



TAGS

TAGS

- E’ possibile assegnare nomi mnemonici a specifici commit.
  - Questi nomi sono chiamati **tag**
- Per ottenere la lista dei tag

```
git tag
```

- Per creare un nuovo tag

```
git tag <nome-tag>
```

- Per tornare a uno specifico commit usando un tag.

```
git checkout <nome-tag>
```

- Per maggiori informazioni
  - **https://git-scm.com/book/en/v2/Git-Basics-Tagging**

MAIN BRANCHES

SUPPORTING BRANCHES

FEATURE DEVELOPMENT

RELEASE DEVELOPMENT

HOT FIXES IN PRODUCTION

GIT WORKFLOW MODEL

MAIN BRANCHES

MAIN BRANCHES ARE PERMANENT BRANCHES

WHICH ARE CREATED AT THE VERY BEGINNING

and never deleted!

BRANCH PRINCIPALI

- **Development**
  - Branch per lo sviluppo

- Test e Quality Assurance (opzionale)
  - Utilizzata dal team di test

- Staging e User Acceptance Test (opzionale)
  - Versione del codice da testare con l’utente.

- **Master** (Produzione)
  - Versione del codice pronta all’uso da parte dell’utente finale.

MAIN BRANCHES

DEVELOPMENT

OPTIONAL

TEST & QA

STAGING & UAT

MASTER

PRODUCTION

It reflects the latest delivered development changes for the next release

Automatic nightly builds

MAIN BRANCHES

OPTIONAL

MASTER

DEVELOPMENT

TEST & QA

STAGING & UAT

~~PRODUCTION~~

It reflects the code which is deployed to test/qa environment for testing

Automatic nightly builds

Automatically deployed to test/qa servers every night

MAIN BRANCHES

DEVELOPMENT

OPTIONAL

TEST & QA

STAGING & UAT

MASTER

~~PRODUCTION~~

It reflects the code which is deployed to staging/uat environment for testing

Automatic nightly builds

Automatically deployed to test/qa servers every night

Automatically deployed to staging servers every night

MAIN BRANCHES

OPTIONAL

DEVELOPMENT | TEST & QA | STAGING & UAT | ~~PRODUCTION~~ MASTER

It reflects the code which is deployed to production

Automatic nightly builds

Automatically deployed to test/qa servers every night

Automatically deployed to staging servers every night

Automatically deployed to production servers on demand

SUPPORTING BRANCHES

FEATURE BRANCHES
HOTFIX BRANCHES
RELEASE BRANCHES

THESE HAVE LIMITED LIFE TIME

AND WILL BE REMOVED EVENTUALLY

SUPPORTING BRANCHES

FEATURE BRANCHES
HOTFIX BRANCHES
RELEASE BRANCHES

THESE HAVE SPECIFIC PURPOSE

AND STRICT RULES FOR ORIGINATING AND TARGET BRANCHES

FEATURE DEVELOPMENT

ALL FEATURE BRANCHES SHOULD BE CREATED FROM

THE DEVELOPMENT BRANCH

the next release branch

FEATURE DEVELOPMENT

TAKE A NEW FEATURE BRANCH

REGARDLESS OF THE FEATURE SIZE

you can fix very minor common issues directly in development branch, like fixing typos

FEATURE BRANCH

- Ogni volta che una nuova funzionalità viene implementata si crea un nuovo branch del codice.

- Anche se la funzionalità è semplice da implementare, sarebbe meglio creare un nuovo branch (per uniformità)
  - Per funzionalità molto semplici, si può comunque usare il branch “development”

- **Proviene da**: development
- **Torna a**: development

FEATURE DEVELOPMENT

FEATURE 1

FEATURE 2

FEATURE 3

DEVELOPMENT

NEXT RELEASE

2

Feature 2 is completed and ready for the next release

FEATURE DEVELOPMENT

FEATURE 1

FEATURE 2

FEATURE 3

DEVELOPMENT

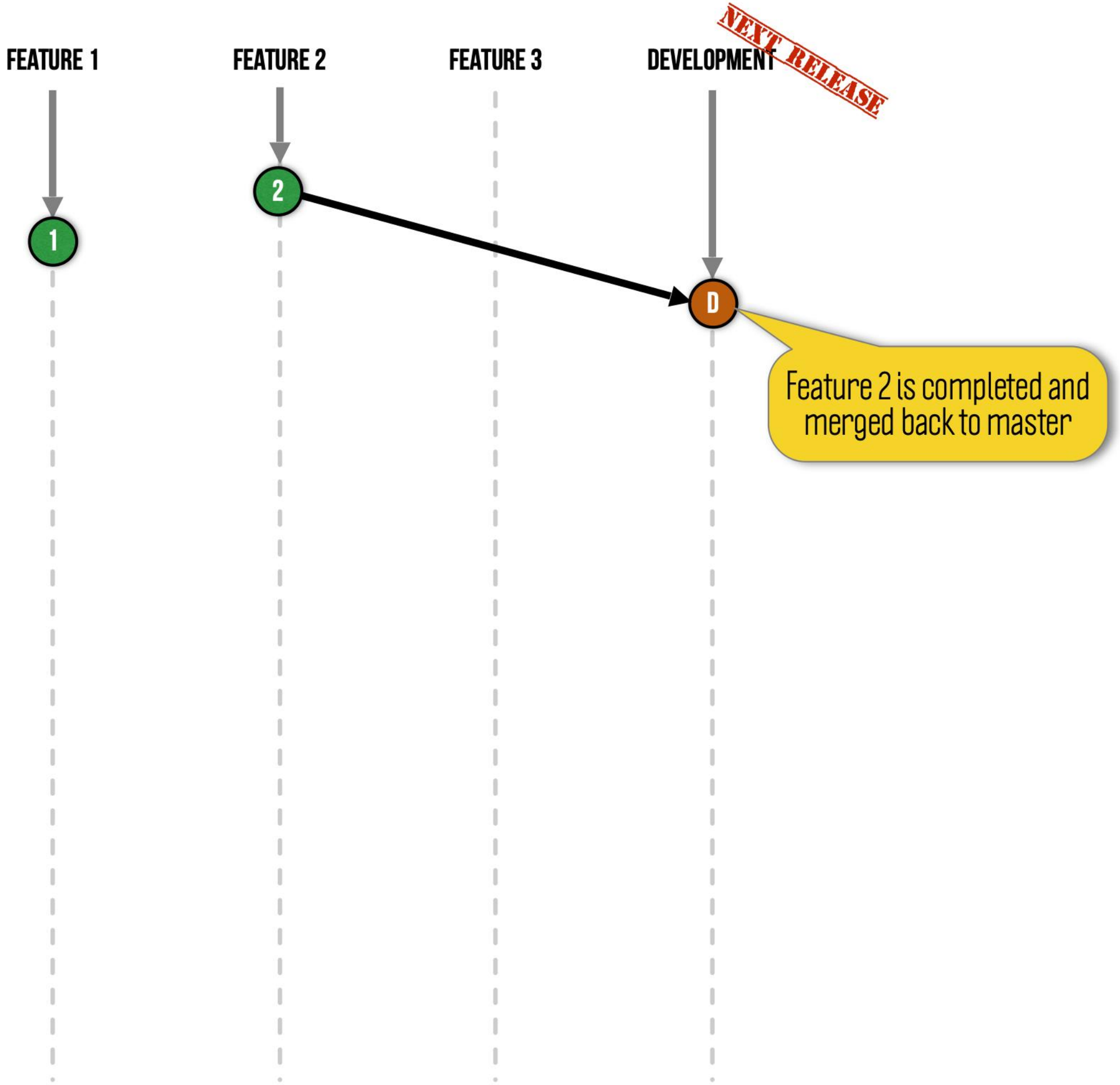
NEXT RELEASE

1

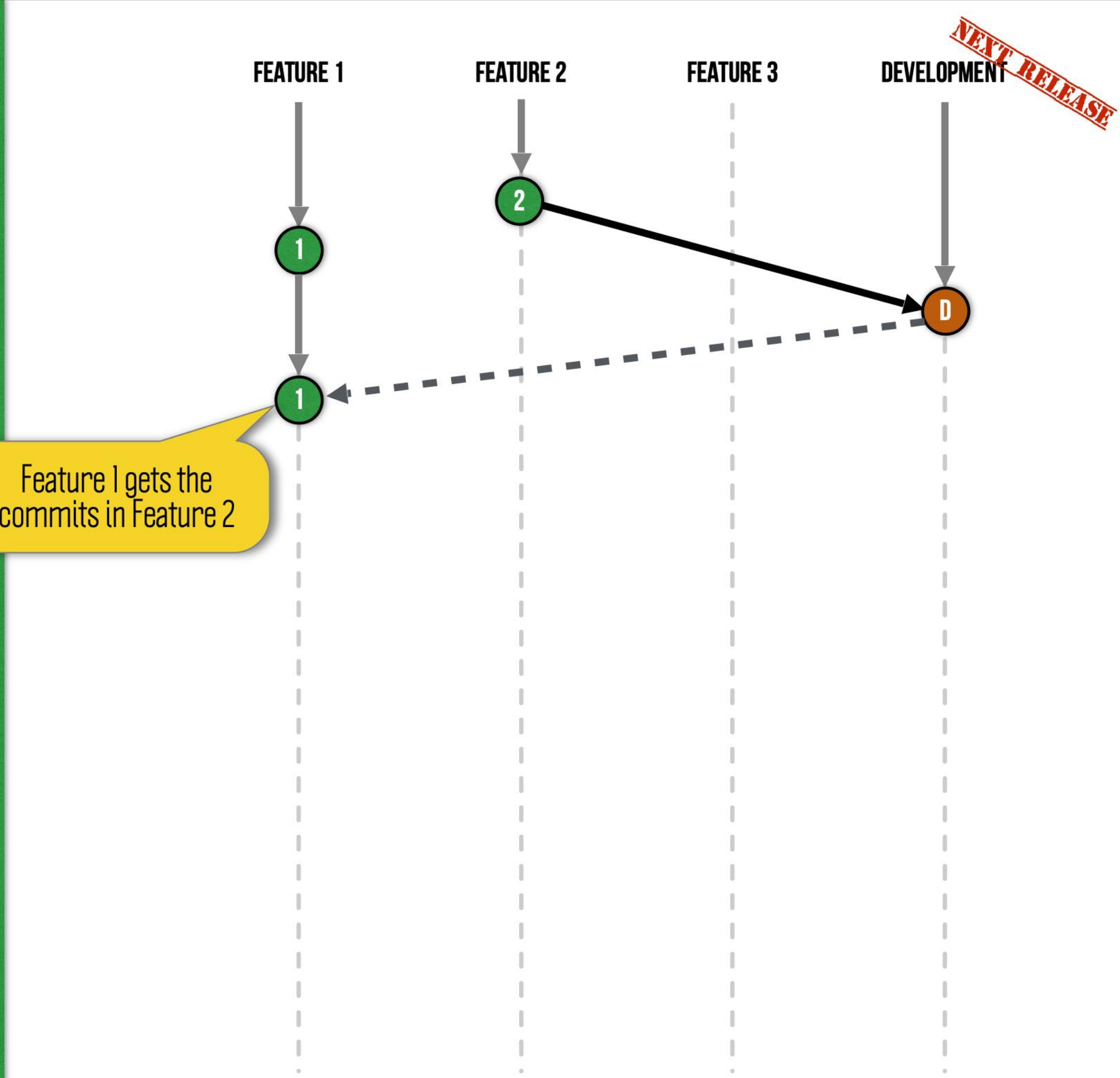
2

Feature 1 is being developed

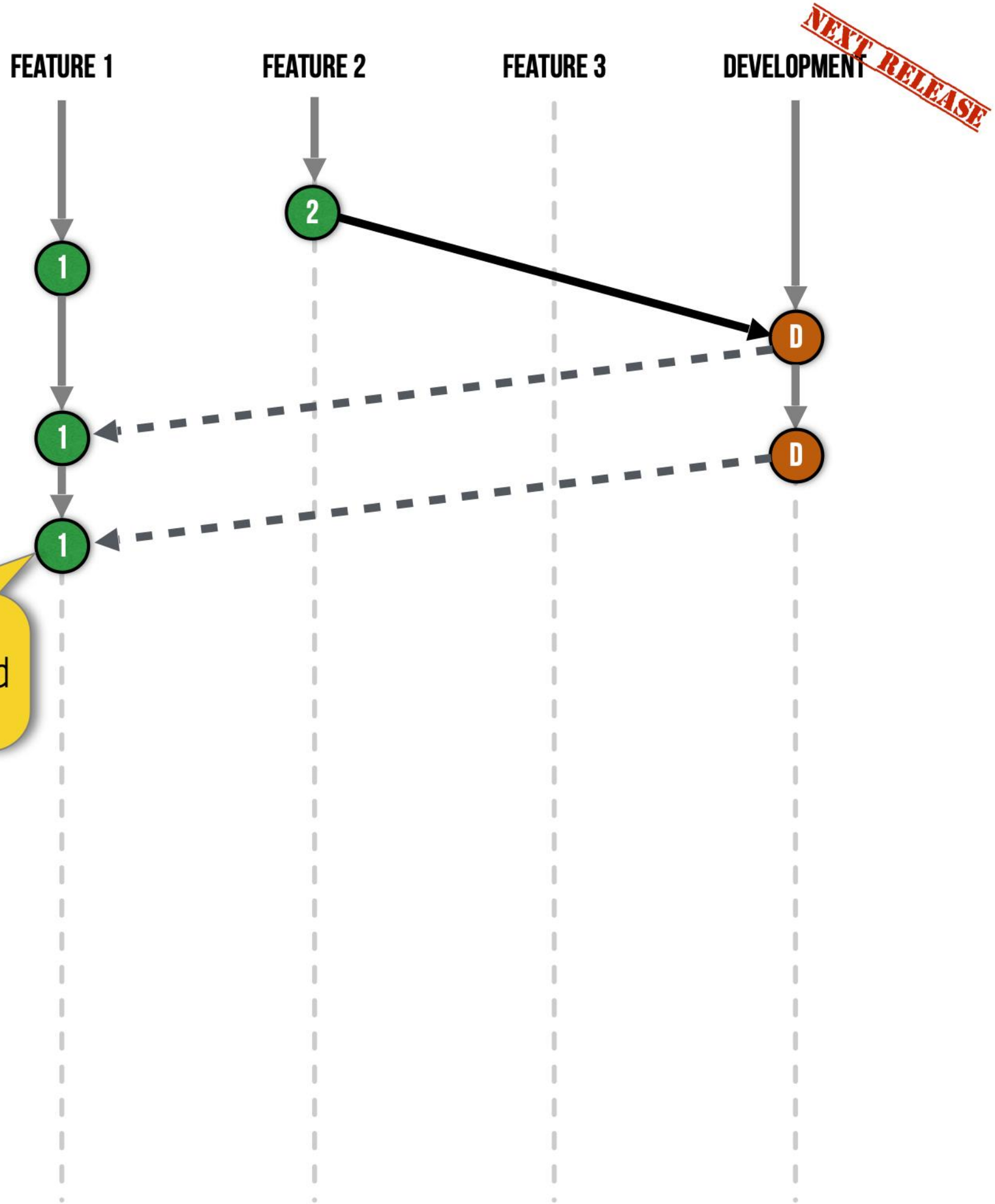
FEATURE DEVELOPMENT



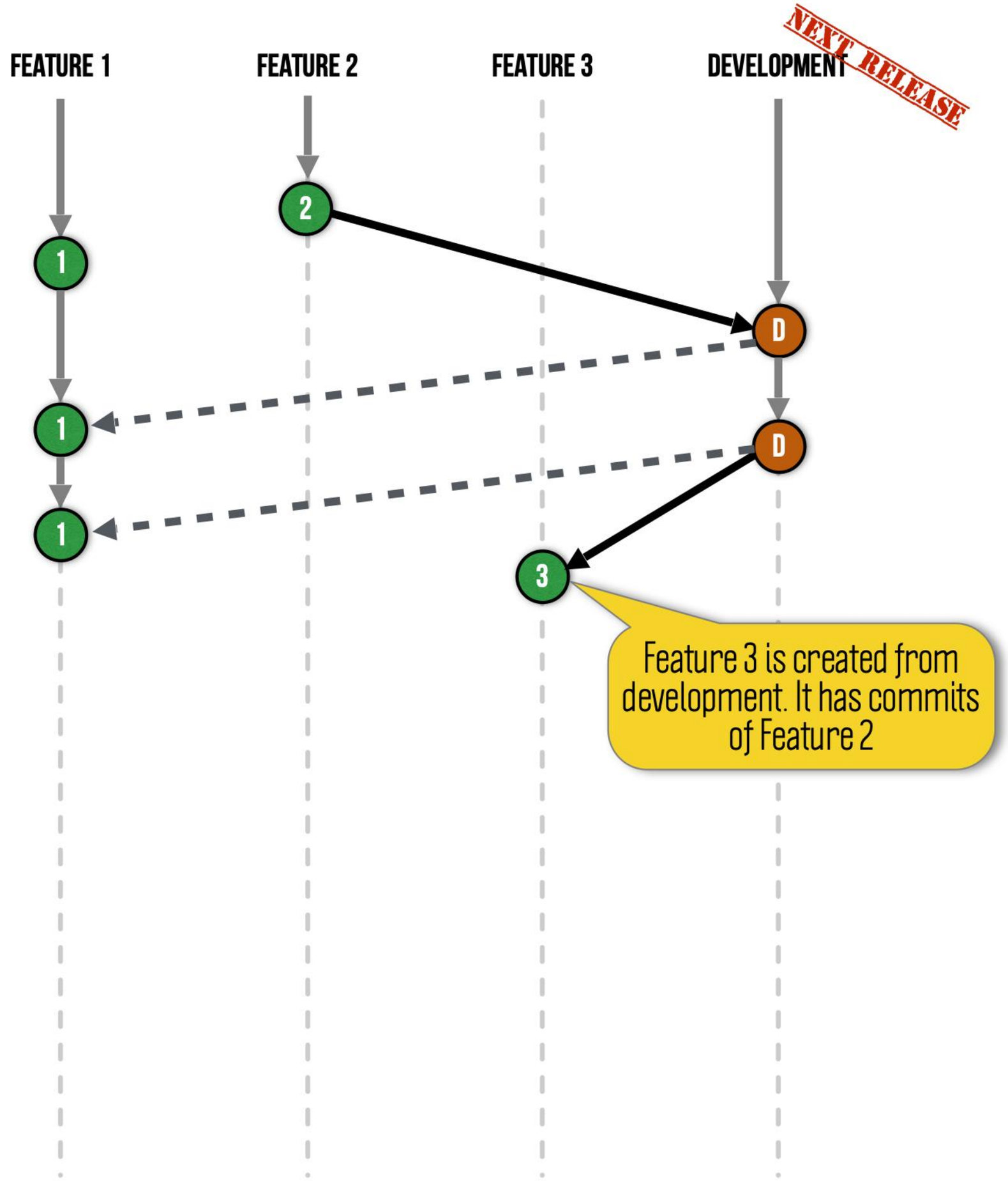
FEATURE DEVELOPMENT



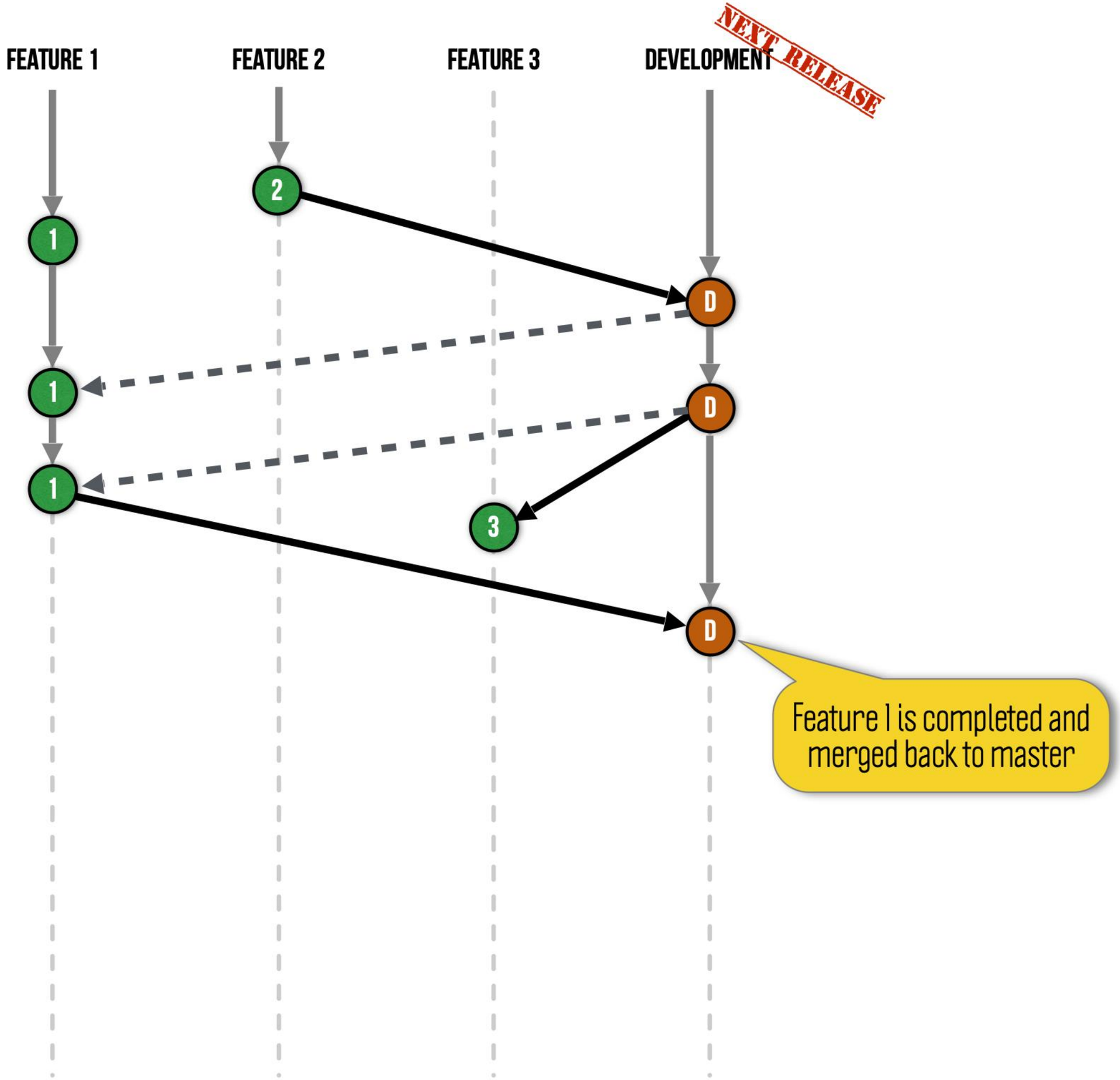
FEATURE DEVELOPMENT



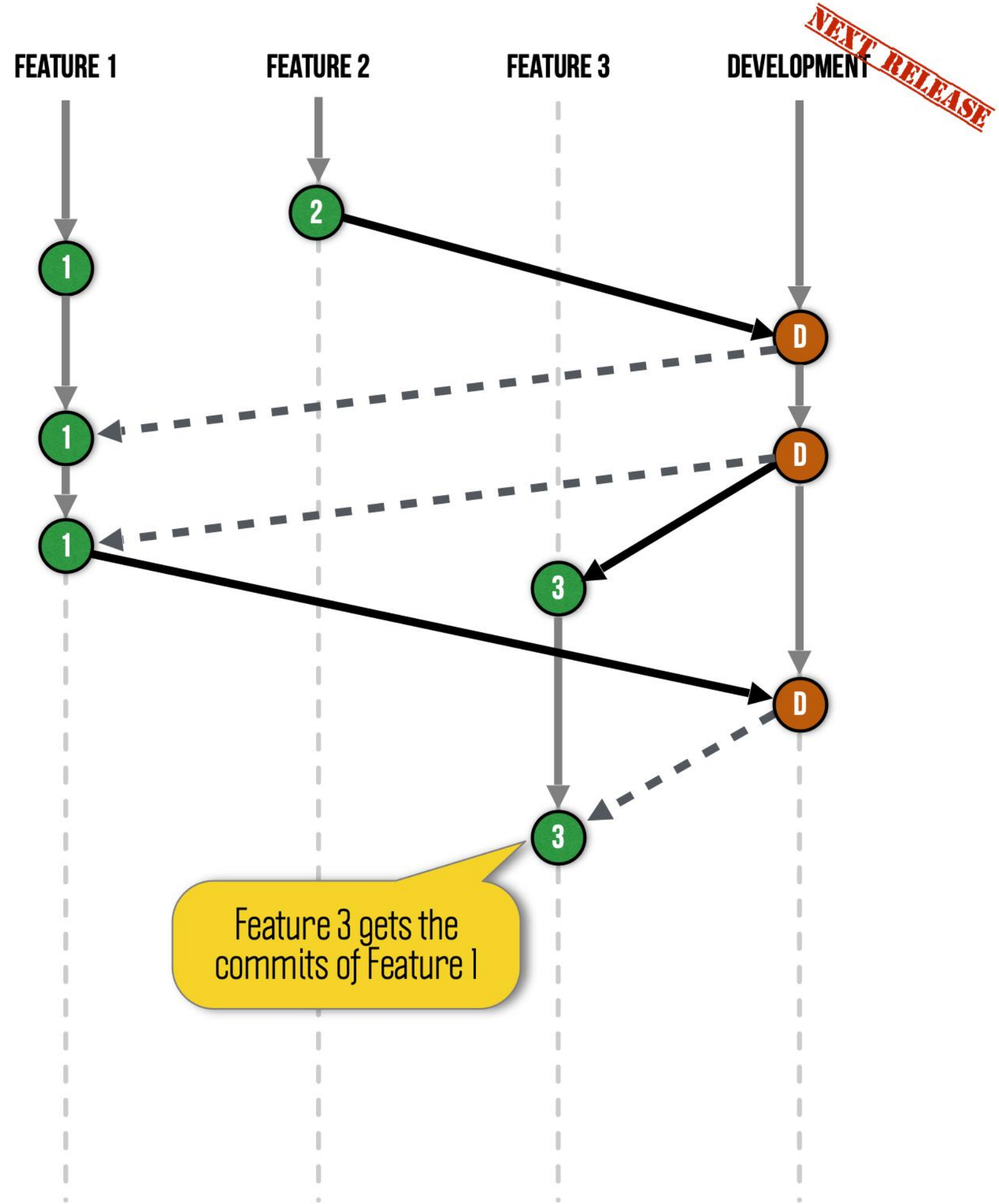
FEATURE DEVELOPMENT



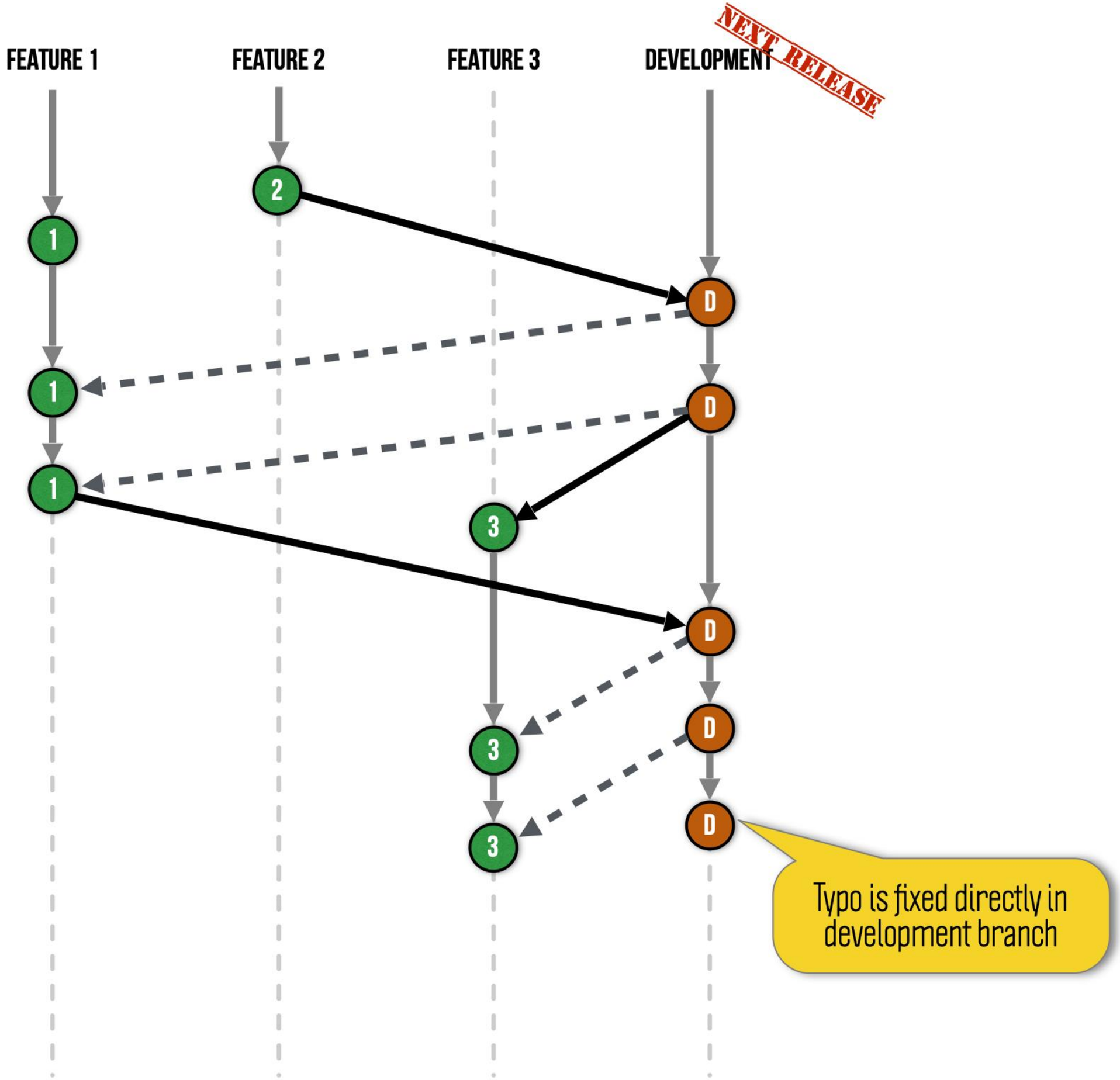
FEATURE DEVELOPMENT



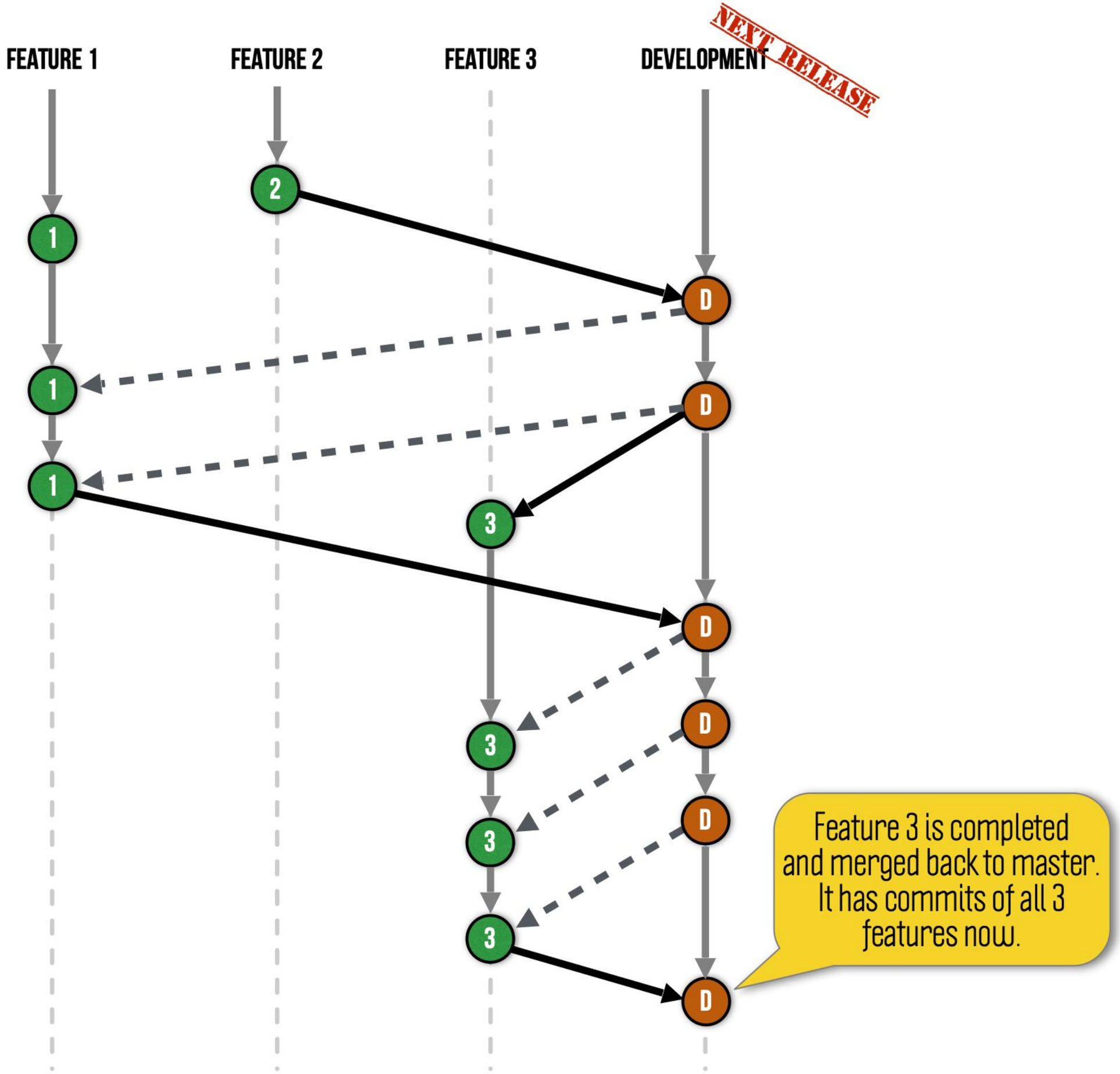
FEATURE DEVELOPMENT



FEATURE DEVELOPMENT



FEATURE DEVELOPMENT



RELEASE DEVELOPMENT

RELEASE BRANCH IS CREATED WHEN

ALL FEATURES ARE READY FOR THE NEXT RELEASE

sometimes we call it as feature freeze

RELEASE DEVELOPMENT

RELEASE BRANCH IS DELETED WHEN

THE RELEASE COMPLETES

we will tag the released code, no need to keep the branch..

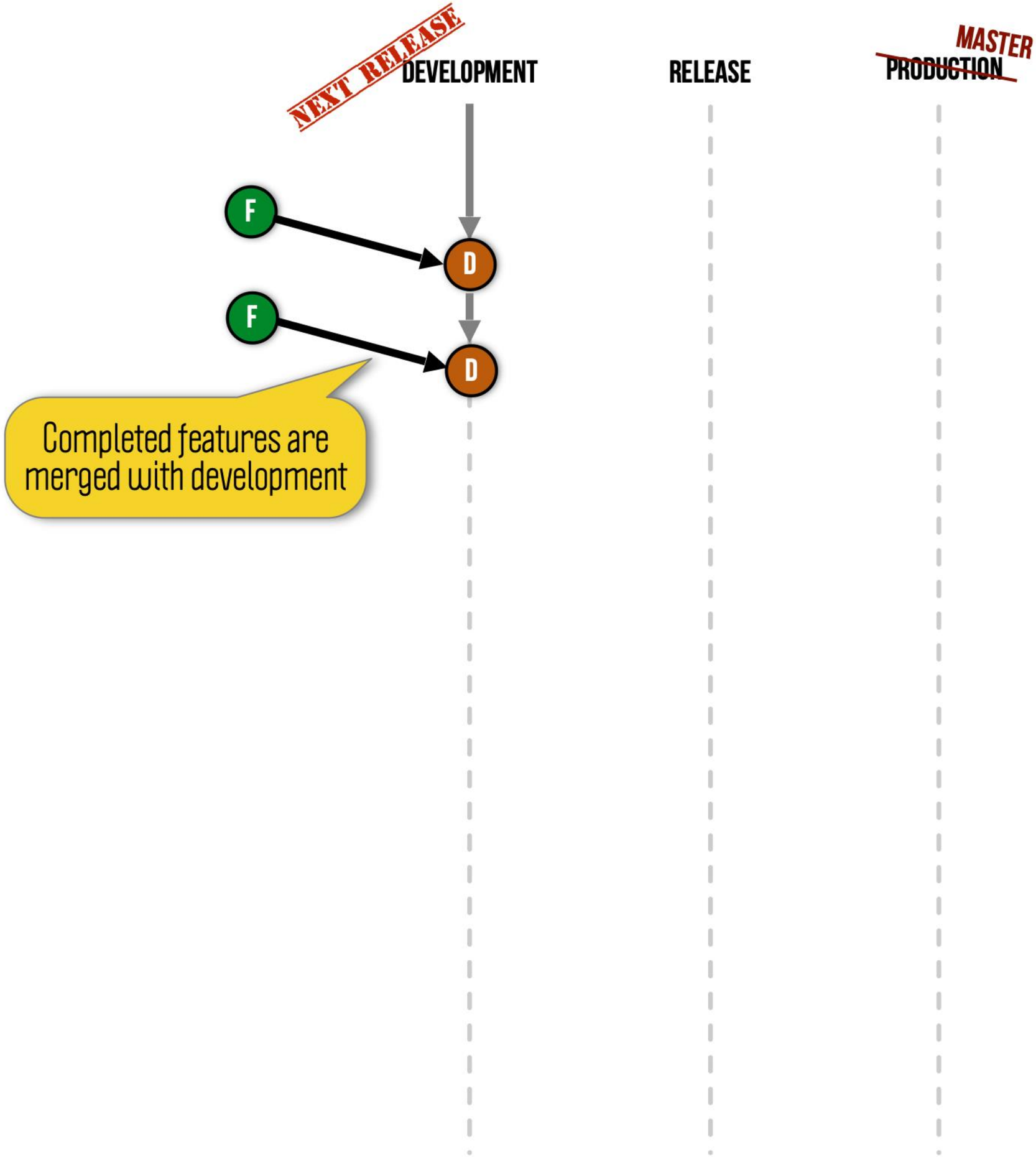
RELEASE BRANCH

- Utilizzati per lavorare su specifiche release del codice
  - Utile per team molto grandi

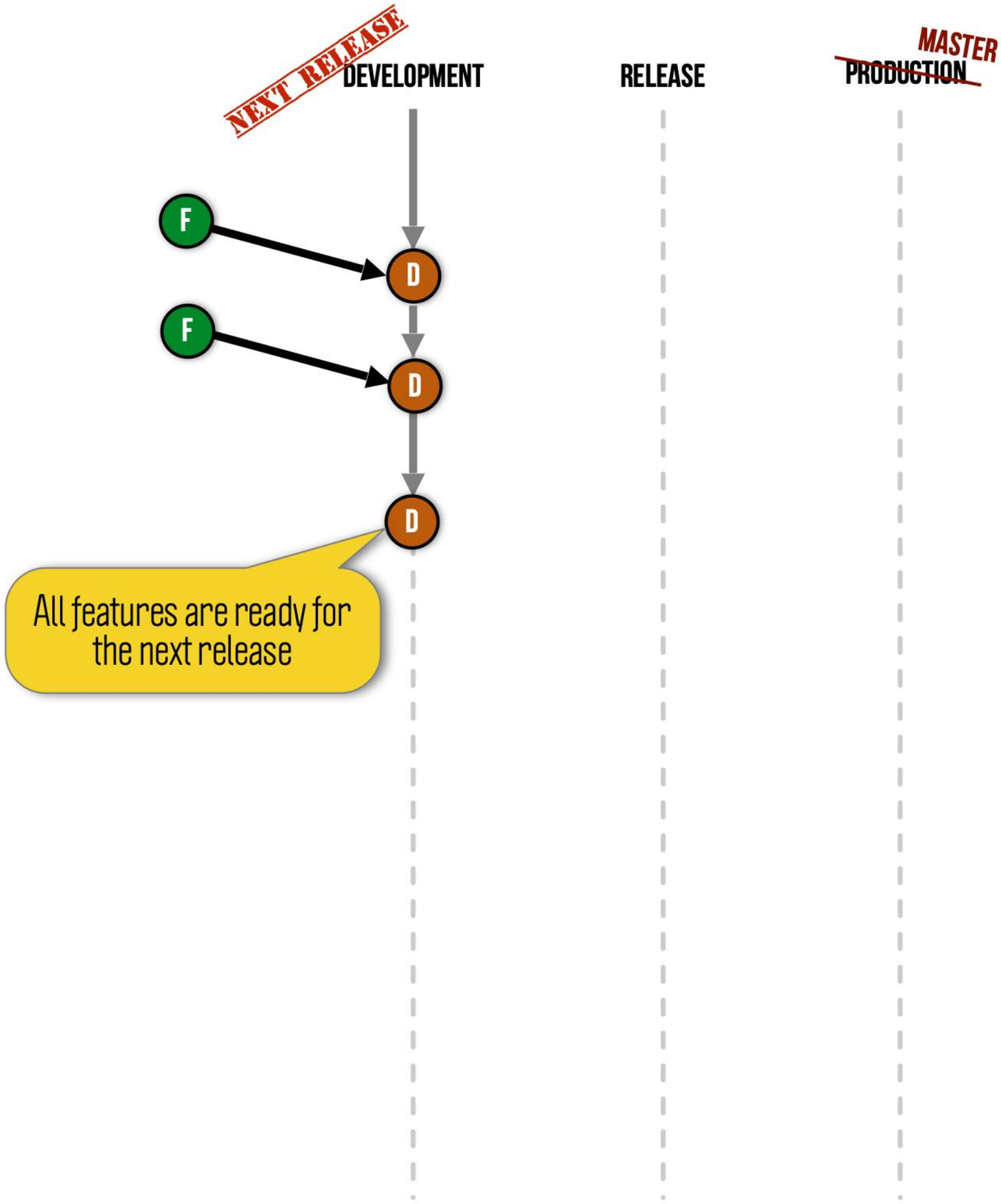
- Congela il codice in uno stato “stabile”, pronto per gli ultimi test prima del rilascio in produzione.
  - Solo piccole modifiche sono accettate su questo branch

- **Proviene da**: development
- **Torna a**: development e master.

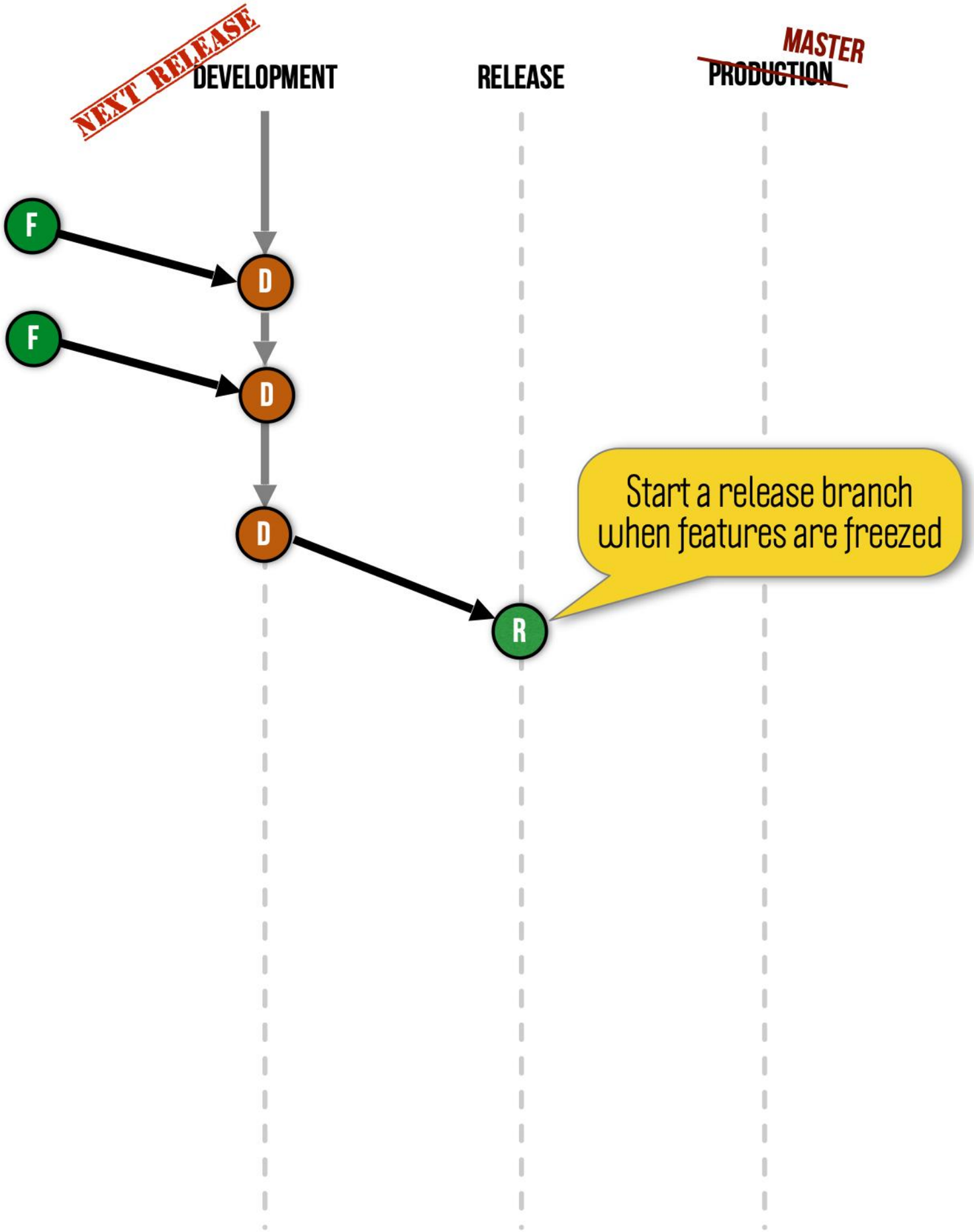
RELEASE DEVELOPMENT



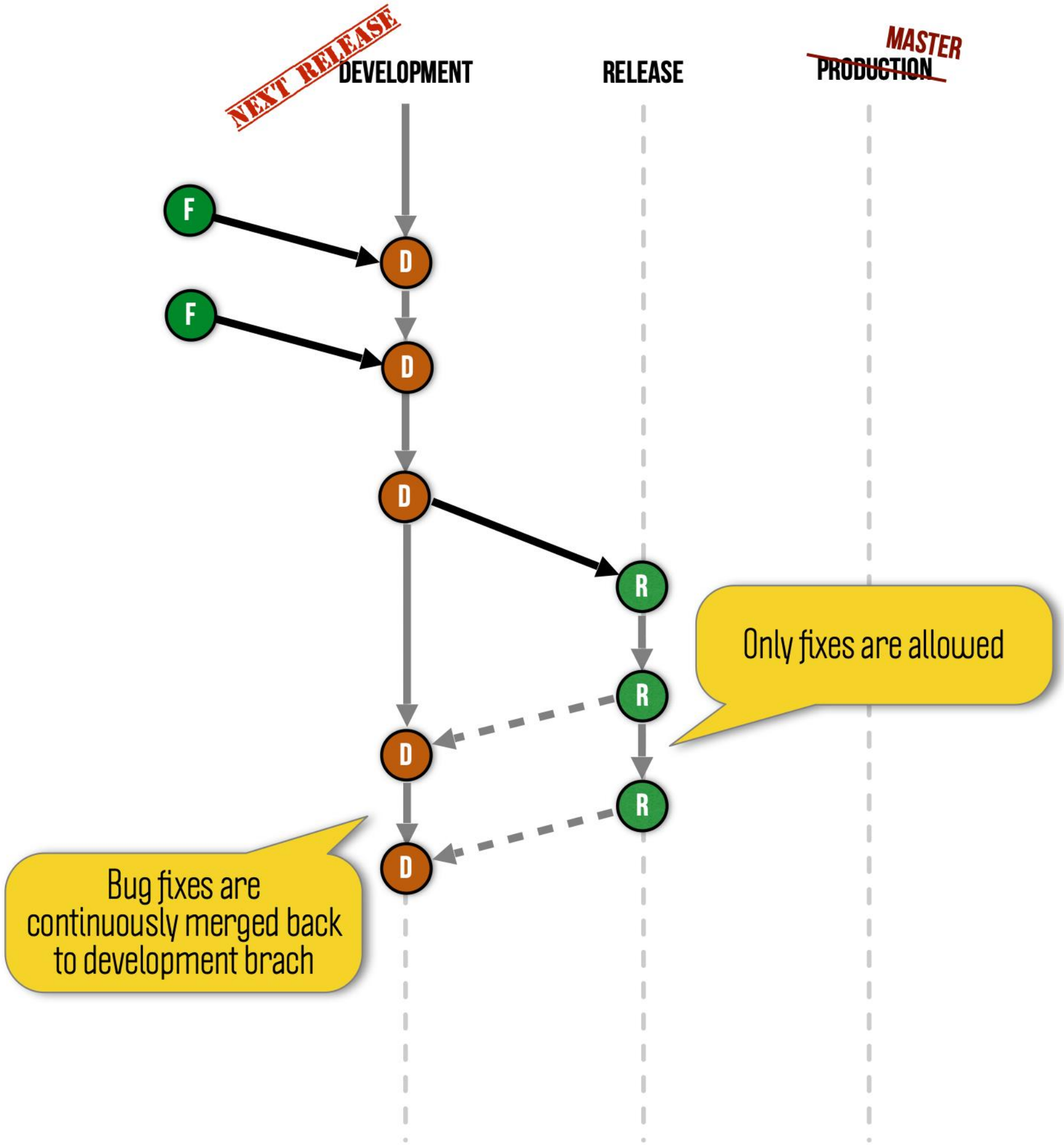
RELEASE DEVELOPMENT



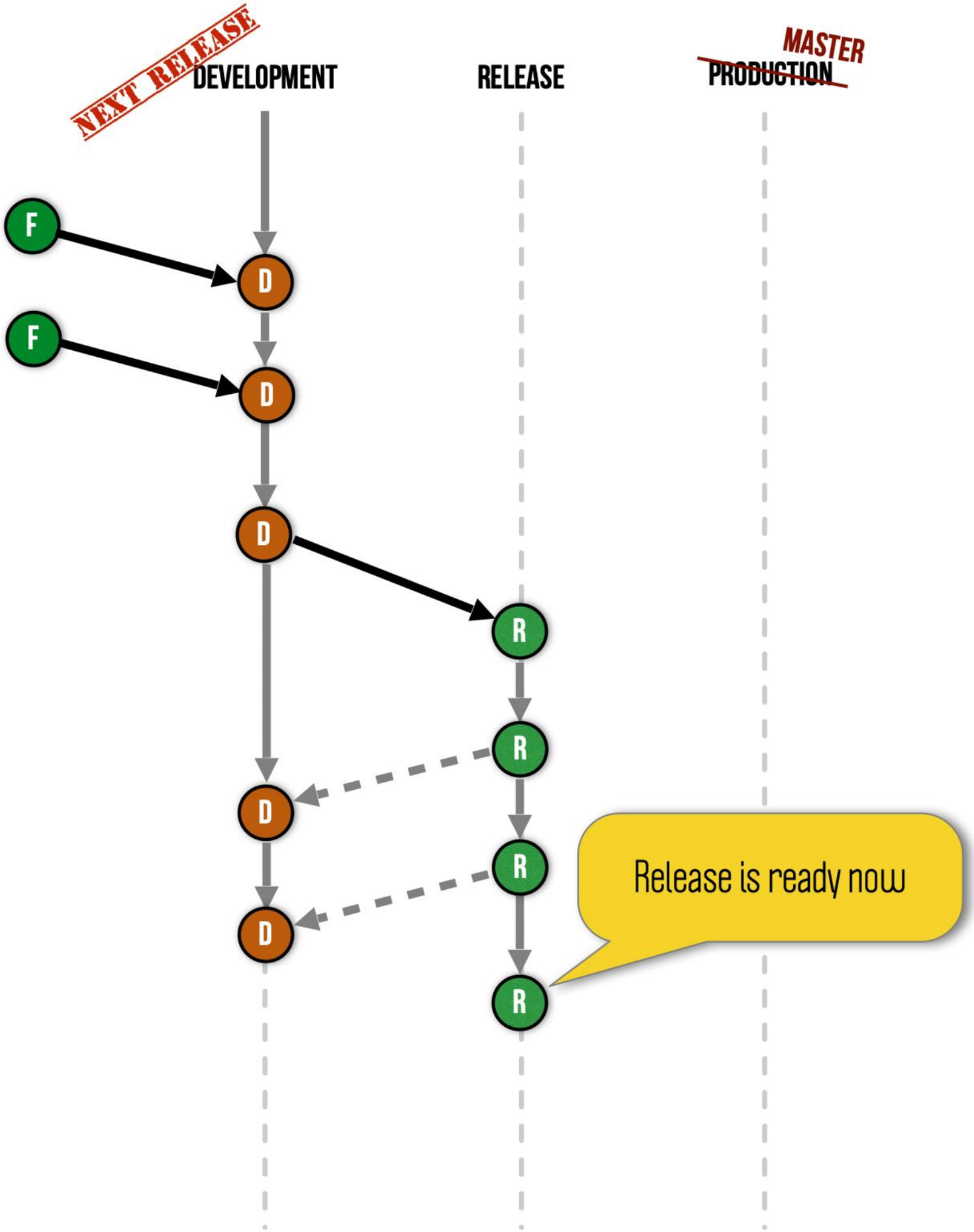
RELEASE DEVELOPMENT



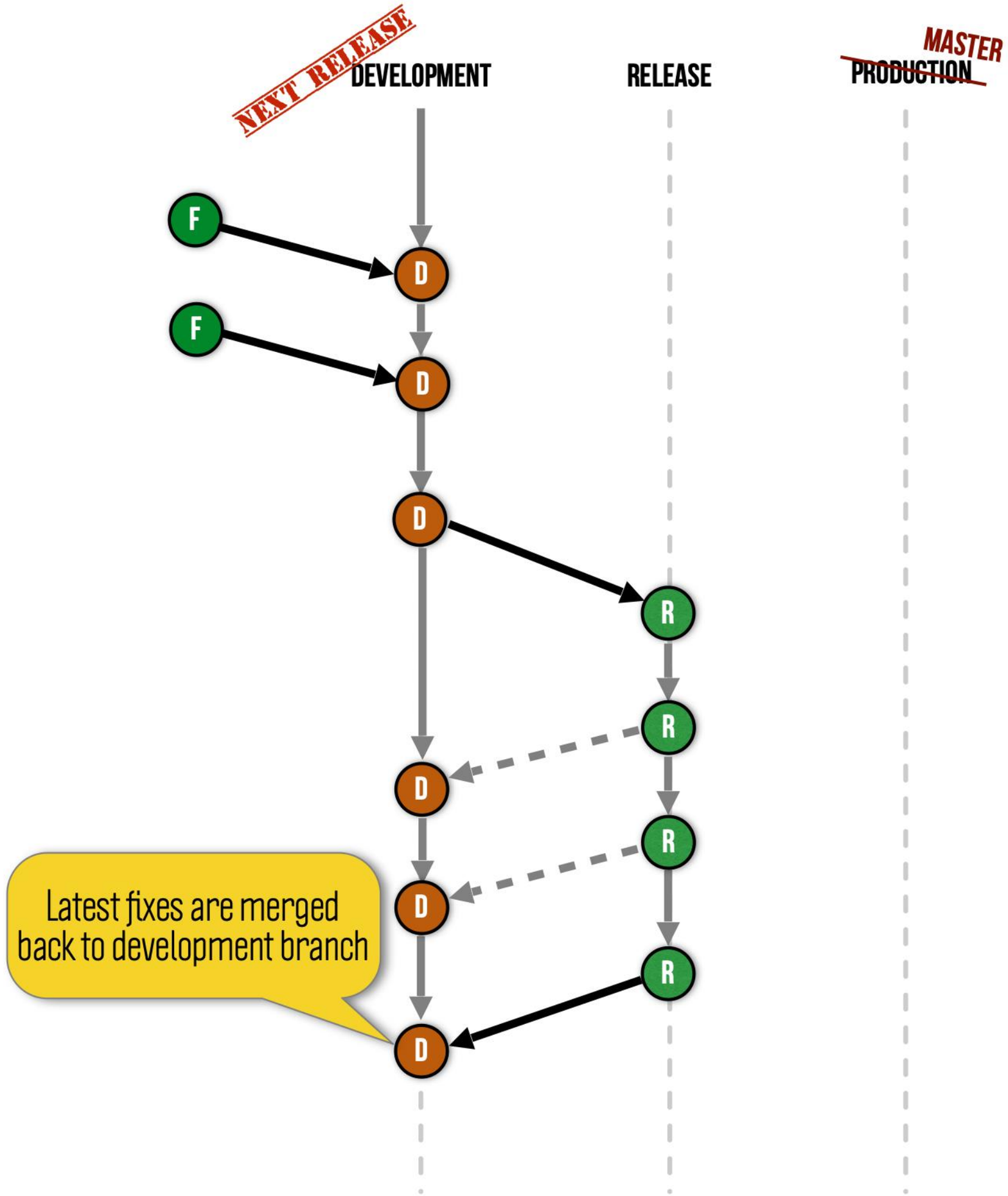
RELEASE DEVELOPMENT



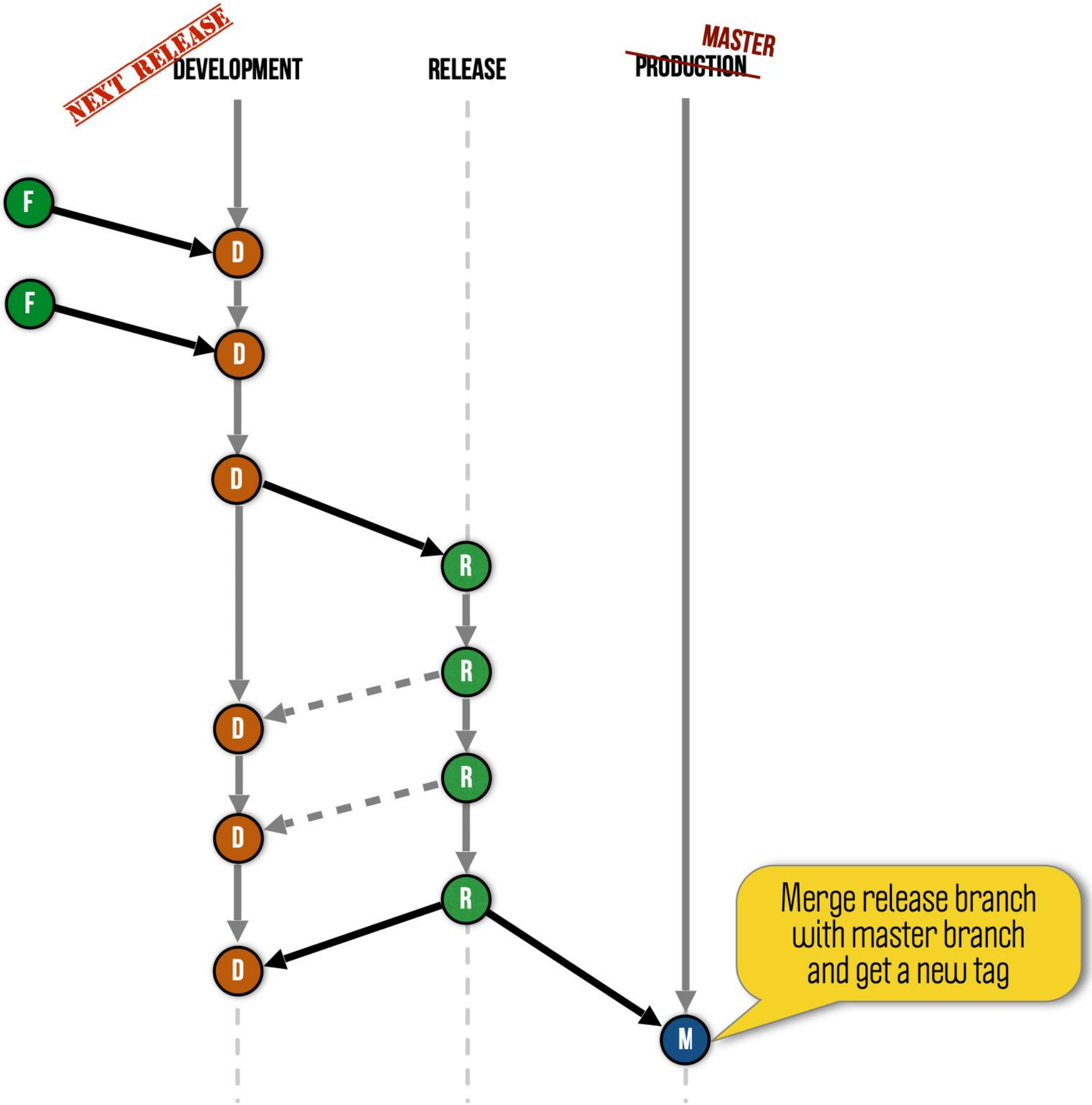
RELEASE DEVELOPMENT



RELEASE DEVELOPMENT



RELEASE DEVELOPMENT



HOT FIXES IN PRODUCTION

FIX BRANCHES ARE REQUIRED BECAUSE EVERY FIX SHOULD BE

WELL TESTED AND VERIFIED

if means, you may need to rollback what you've done in the fix

HOT-FIX BRANCH

- Utilizzati per risolvere problemi riscontrati nell’applicazione
  - Bug riscontrati in produzione (dopo il rilascio)

- Questi bug devono essere risolti il prima possibile.

- **Proviene da**: master
- **Torna a**: development e master

HOT FIXES IN PRODUCTION

NEXT RELEASE

DEVELOPMENT

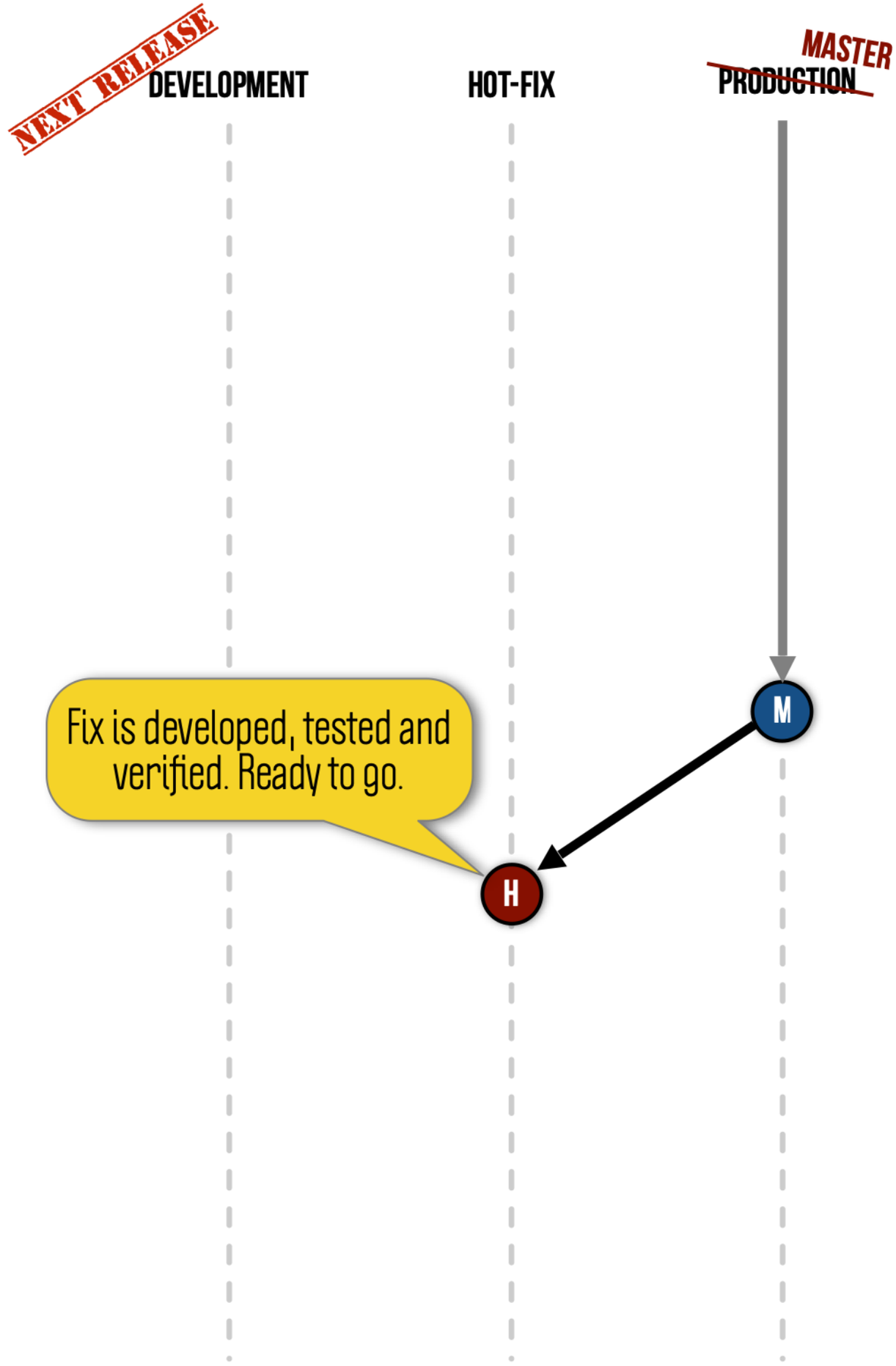
HOT-FIX

~~PRODUCTION~~ MASTER

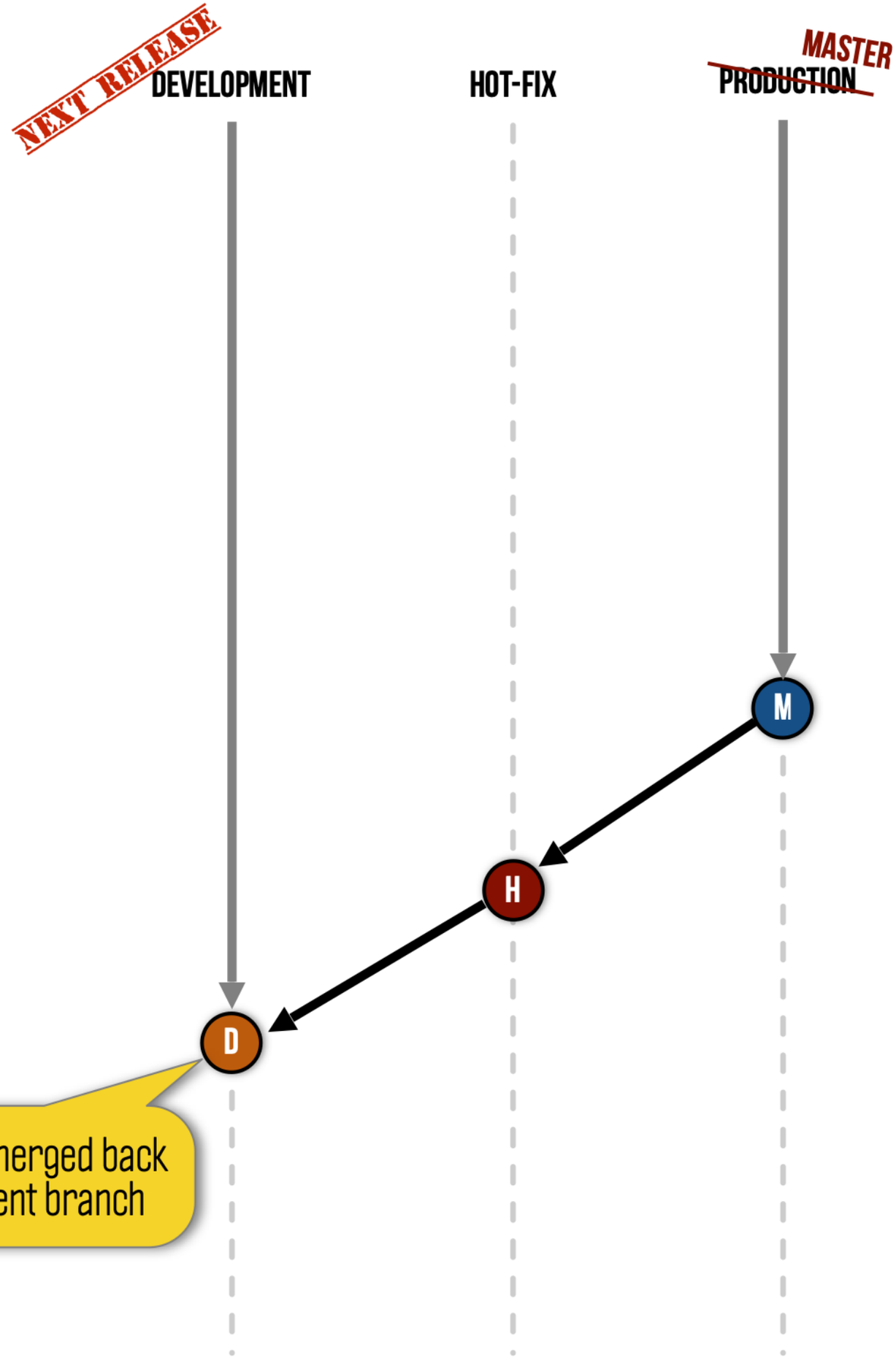
Hot fix is required! A new branch is created from production branch

M

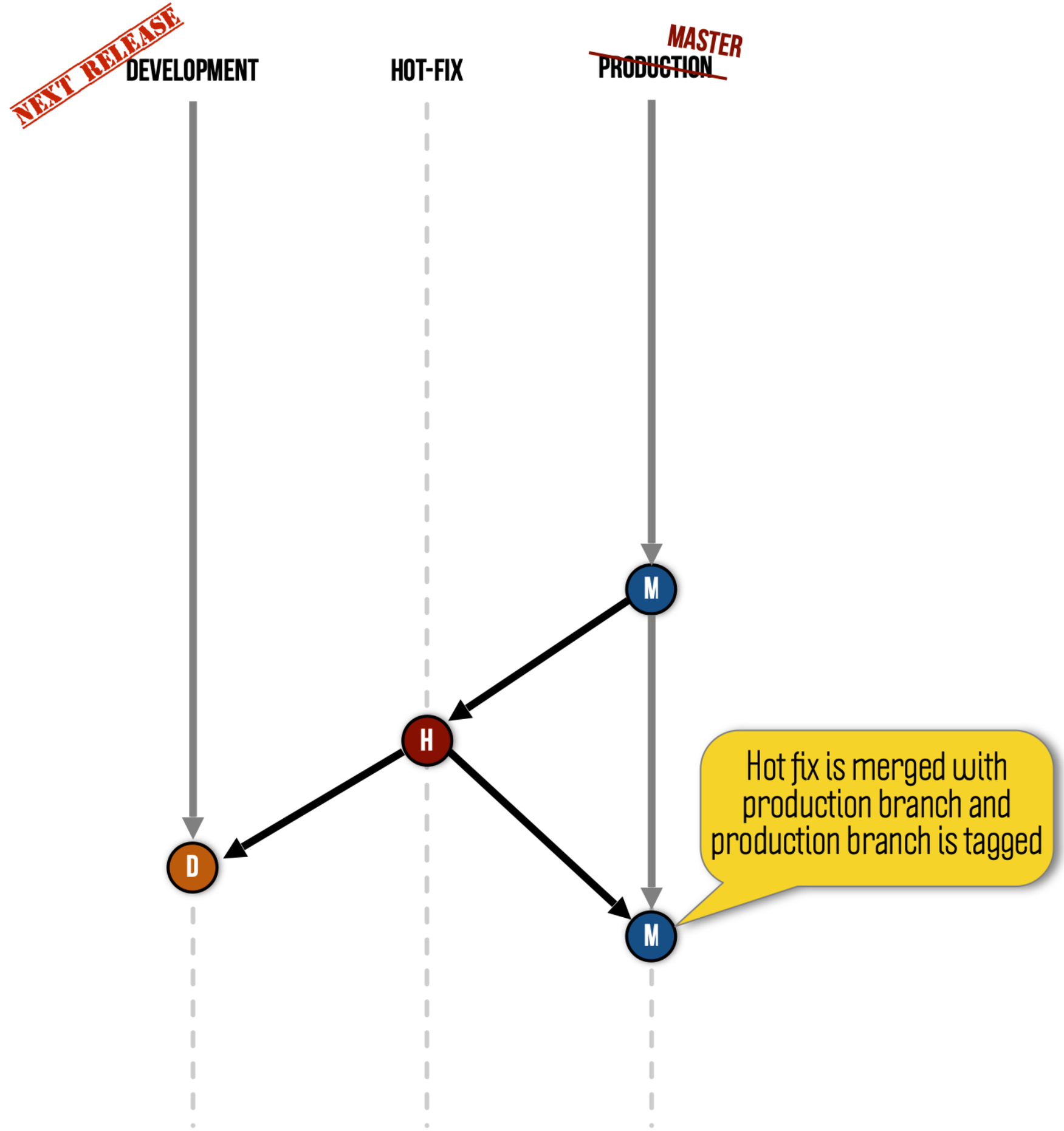
HOT FIXES IN PRODUCTION



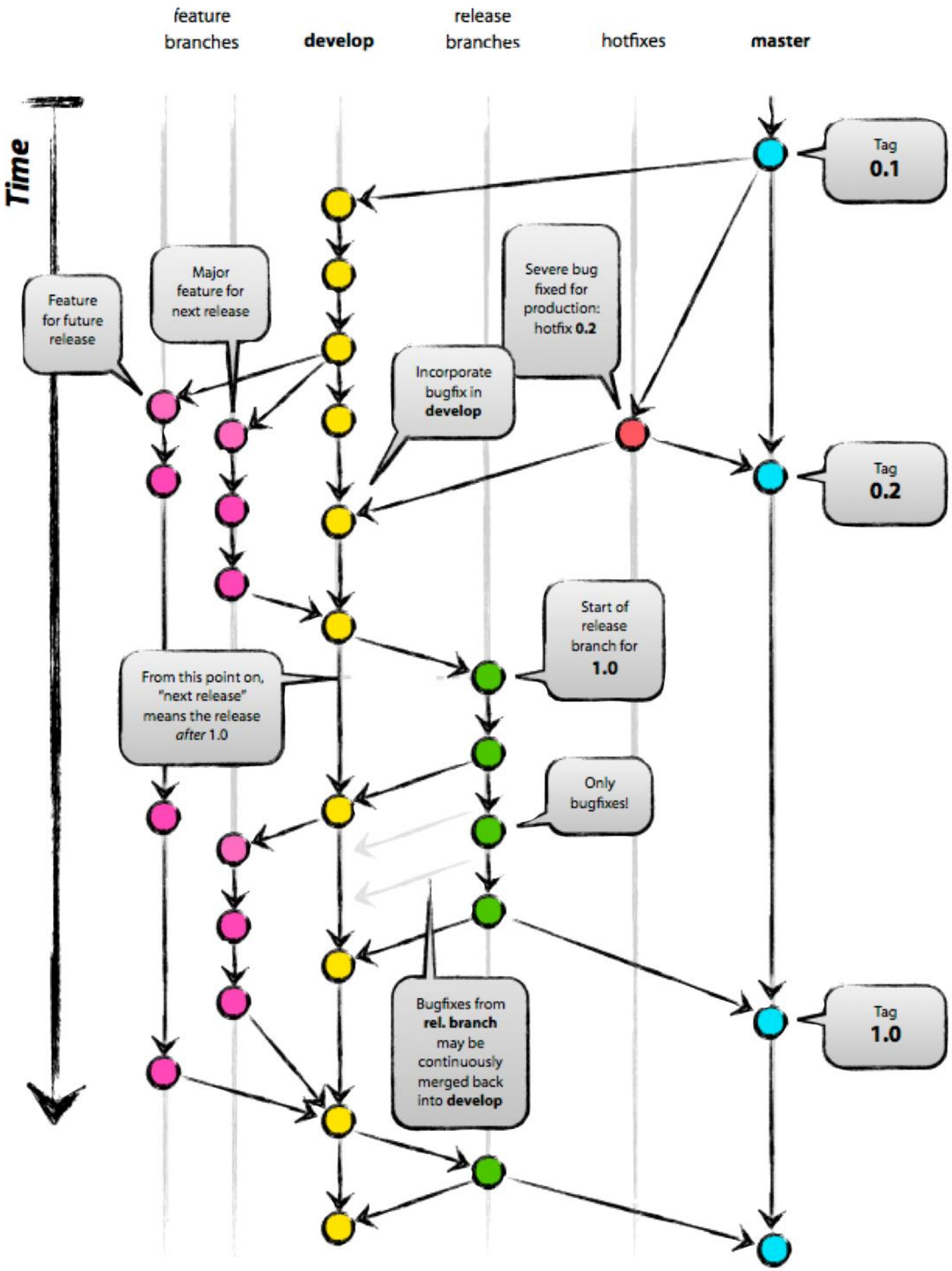
HOT FIXES IN PRODUCTION



HOT FIXES IN PRODUCTION



ALL FLOWS IN THE MODEL



THIS GRAPHIC IS TAKEN FROM VINCENT DRIESSEN'S ARTICLE CALLED "A SUCCESSFUL GIT BRANCHING MODEL"

RIASSUMENDO

- **Branch Master.**
  - Ogni commit rappresenta una versione funzionante del software
  - Pochi commit.

- **Development.**
  - Ogni commit rappresenta l’aggiunta di una feature

- **Feature.**
  - Diversi commit rappresentano l’avanzamento dello sviluppo di una feature.

- **Hot-fix.**
  - Risoluzione di bug nel codice

NAMING CONVENTIONS

- **Branch Master.**
  - Nome standard
- **Branch Development.**
  - Nome standard
- **Branch Feature.**
  - feature_nome-feature
  - ***`nome-feature`*** potrebbe essere un codice alfanumerico
  - ***`nome-feature`*** potrebbe contenere il nome dell’autore
- **Branch Hotfix.**
  - hotfix_bug-description
  - ***`description`*** potrebbe essere un codice a alfanumerico

PER IL PROGETTO

- **Branch Master.**
  - Necessario.

- **Development.**
  - Necessario

- **Feature.**
  - Caldamente consigliati.

FEATURE DEVELOPMENT

1

CREATE PERMANENT BRANCHES

IF THESE DOES NOT EXIST

```
➜git:(master) git checkout -b development
Switched to a new branch 'development'

➜git:(development) git push origin development
Total 0 (delta 0), reused 0 (delta 0)
To UberProject.git
* [new branch] development -> development

➜git:(development) git branch -a
* development
 master
 remotes/origin/development
 remotes/origin/master
```

it creates development branch from master and pushes to remote

2

PULL TO UPDATE LOCAL DEVELOPMENT BRANCH

BE SURE YOU RUN TESTS DIRECTLY AFTERWARDS

```
➜git:(master) git fetch origin
From UberProject
* [new branch] development -> origin/development
```

```
➜git:(master) git checkout development
Branch development set up to track remote branch development from origin.
Switched to a new branch 'development'

➜git:(development) git pull
remote: Counting objects: 19, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 17 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (17/17), done.
From UberProject
 3eb8b34..c827c84 development -> origin/development
Updating 3eb8b34..c827c84
Fast-forward
grails-app/controllers/org/example/BookController.groovy | 6 ++++++
grails-app/domain/org/example/Book.groovy | 7 +++++++
test/unit/org/example/BookControllerTests.groovy  | 17 ++++++++++++++++
test/unit/org/example/BookTests.groovy  | 17 ++++++++++++++++
4 files changed, 47 insertions(+)
create mode 100644 grails-app/controllers/org/example/BookController.groovy
create mode 100644 grails-app/domain/org/example/Book.groovy
create mode 100644 test/unit/org/example/BookControllerTests.groovy
create mode 100644 test/unit/org/example/BookTests.groovy
```

3

CREATE A FEATURE BRANCH FROM DEVELOPMENT

WITH A NAME HAVING STORY ID AND DESCRIPTIVE TITLE

```
➜git:(development) git checkout -b feature-1185-add-commenting
Switched to a new branch 'feature-1185-add-commenting'

➜git:(feature-1185-add-commenting)
```

the title should give hints about what's in it

4

WORK IN YOUR FEATURE BRANCH

COMMIT EARLY AND OFTEN

Be careful! it does not mean "push early and often"

```
➜git:(feature-1185-add-commenting) vi web-app/WEB-INF/applicationContext.xml
➜git:(feature-1185-add-commenting) ✗ git add .
➜git:(feature-1185-add-commenting) ✗ git commit -m "added comment for applicationContext.xml"
[feature-1185-add-commenting b6f68cc] added comment for applicationContext.xml
1 file changed, 2 insertions(+), 1 deletion(-)

➜git:(feature-1185-add-commenting) vi web-app/WEB-INF/sitemesh.xml
➜git:(feature-1185-add-commenting) ✗ git add .
➜git:(feature-1185-add-commenting) ✗ git commit -m "added comment for sitemesh.xml"
[feature-1185-add-commenting 0e74f89] added comment for sitemesh.xml
1 file changed, 3 insertions(+), 1 deletion(-)

➜git:(feature-1185-add-commenting) vi .application.properties
➜git:(feature-1185-add-commenting) ✗ git add .
➜git:(feature-1185-add-commenting) ✗ git commit -m "increased the application version"
[feature-1185-add-commenting 7ce1f07] increased the application version
1 file changed, 1 insertion(+), 1 deletion(-)
```

5

REBASE FREQUENTLY TO INCORPORATE UPSTREAM CHANGES

TO PREVENT YOUR BRANCH FROM DIVERGING SIGNIFICANTLY

```
➜ git:(feature-1185-add-commenting) git fetch origin development
From UberProject
* branch development -> FETCH_HEAD

➜ git:(feature-1185-add-commenting) git rebase origin/development
First, rewinding head to replay your work on top of it...
Fast-forwarded feature-1185-add-commenting to origin/development.
```

6

INTERACTIVE REBASE TO SQUASH YOUR COMMITS

BE SURE THAT WE ONLY DEAL WITH THE LOCAL COMMITS

```
➜ git:(feature-1185-add-commenting) git rebase -i origin/development
```

GIT WILL DISPLAY AN EDITOR WITH A LIST OF THE COMMITS TO BE MODIFIED

```
pick 4559eba updated application version
pick d1706ae added comments for applicationContext.xml
pick d3c2734 added comments for sitemesh.xml
```

NOW WE NEED TO TELL GIT WHAT TO DO. KEEP THE FIRST ONE AS IT IS AND CHANGE THE OTHERS AS “SQUASH”

```
pick 4559eba updated application version
squash d1706ae added comments for applicationContext.xml
squash d3c2734 added comments for sitemesh.xml
```

GIT SQUASHES THE COMMITS TOGETHER TO ONE COMMIT AND OPENS A NEW EDITOR TO ENTER COMMIT MESSAGE

```
updated application version, added comments to
applicationContext.xml and siteMesh.xml files
```

NOW THE LAST 3 COMMITS ARE SQUASHED INTO ONE COMMIT WITH A SPECIAL COMMIT MESSAGE

7

MERGE YOUR CHANGES WITH DEVELOPMENT BRANCH

IT'S TIME TO MERGE THE COMPLETED WORK

```
➜ git:(feature-1185-add-commenting) git checkout development
Switched to branch 'development'
Your branch is behind 'origin/development' by 3 commits, and can be fast-forwarded.
 (use "git pull" to update your local branch)

➜ git:(development) git pull
Updating c827c84..7761c9c
Fast-forward
application.properties | 2 +-
grails-app/controllers/org/example/ShelfController.groovy | 6 ++++++
grails-app/domain/org/example/Shelf.groovy | 7 +++++++
test/unit/org/example/ShelfControllerTests.groovy  | 17 ++++++++++++++++++
test/unit/org/example/ShelfTests.groovy  | 17 ++++++++++++++++++
5 files changed, 48 insertions(+), 1 deletion(-)
create mode 100644 grails-app/controllers/org/example/ShelfController.groovy
create mode 100644 grails-app/domain/org/example/Shelf.groovy
create mode 100644 test/unit/org/example/ShelfControllerTests.groovy
create mode 100644 test/unit/org/example/ShelfTests.groovy

➜ git:(development) git merge feature-1185-add-commenting
Updating 7761c9c..d55ee12
Fast-forward
application.properties  | 2 +-
web-app/WEB-INF/applicationContext.xml | 3 ++-
web-app/WEB-INF/sitemesh.xml  | 4 +++-
3 files changed, 6 insertions(+), 3 deletions(-)
```

8

PUSH YOUR CHANGES TO THE UPSTREAM

```
➜ git:(feature-1185-add-commenting) git push origin development
Counting objects: 13, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 759 bytes | 0 bytes/s, done.
Total 7 (delta 4), reused 0 (delta 0)
To UberProject.git
 7761c9c..d55ee12 development -> development
```