剖析內部設計

http://ihower.tw

2014/11

我是誰

- 張文鈿 a.k.a. ihower
  - http://ihower.tw
- Instructor at ALPHA Camp
  - http://alphacamp.tw
- Git user since 2008

Git 的內部設計

小測驗

如何用 Git 底層指令，不用 git add 和 git commit 指令進行 commit 動作?

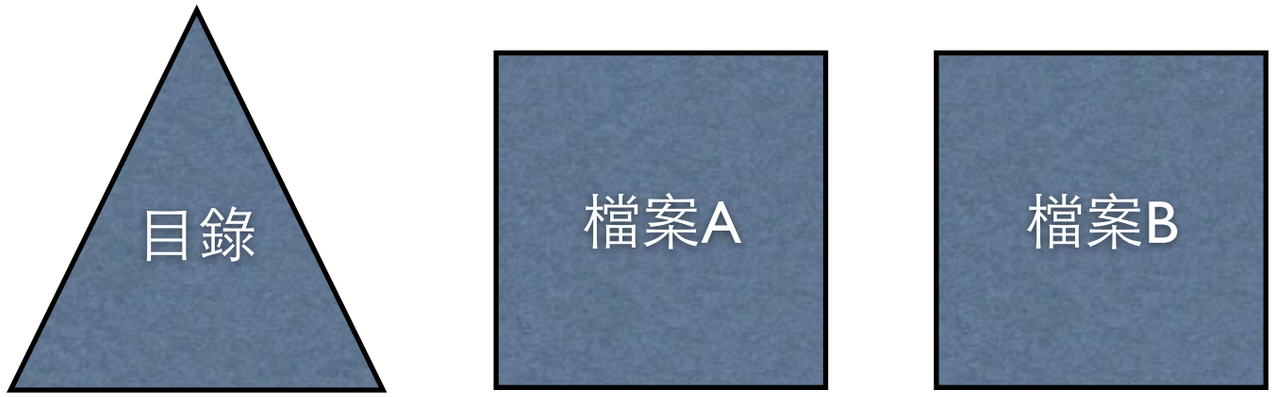
用 Graph 概念理解

@KentBeck
Kent Beck

finally figuring out that git commands are strangely named graph manipulation commands--creating/deleting nodes, moving pointers around

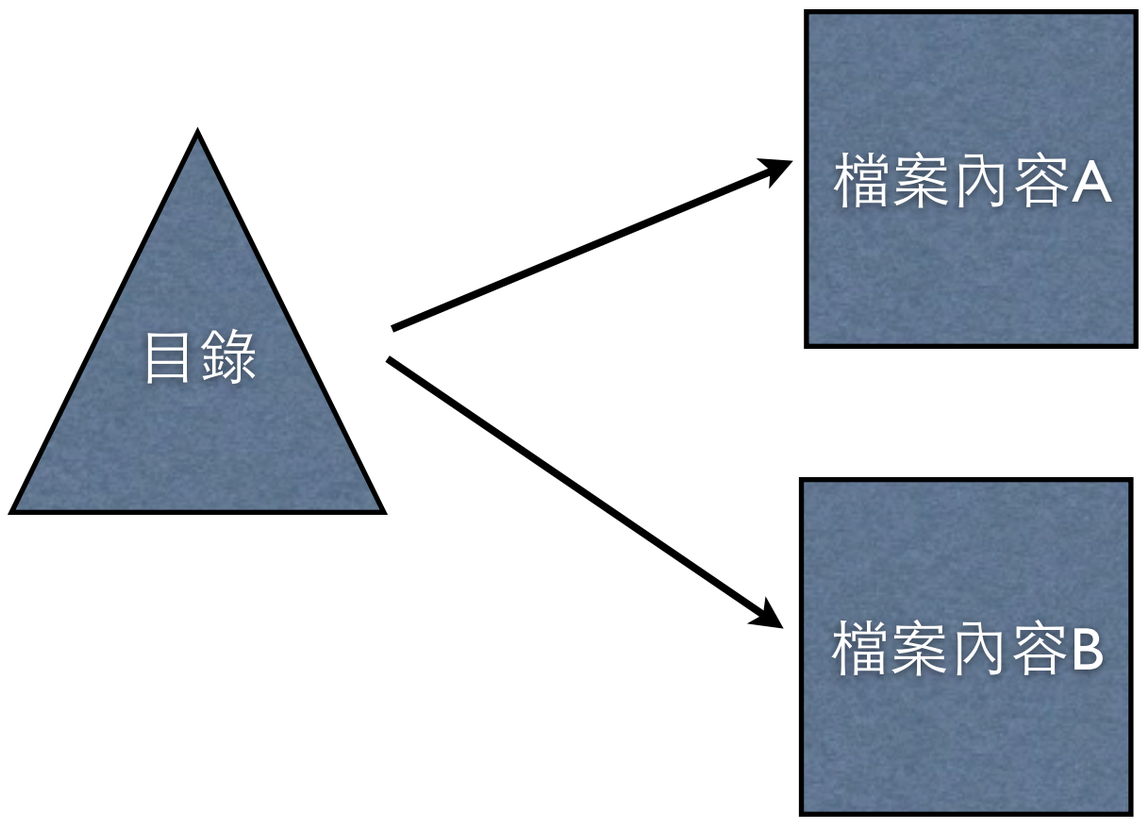
1 Mar via TweetDeck ⭐ Unfavorite ↻ Retweet ↩ Reply

working area



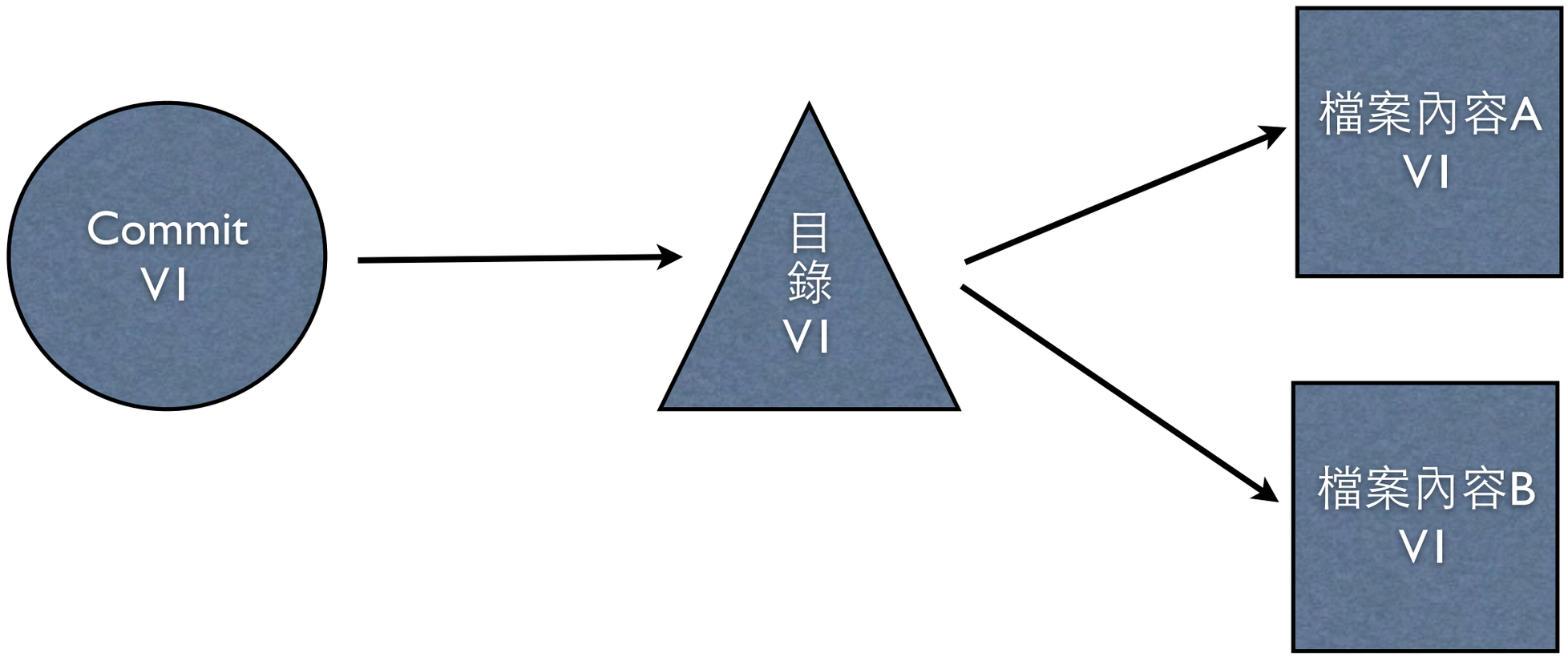
git add .

(將目錄節點和檔案內容節點關聯起來)



git commit

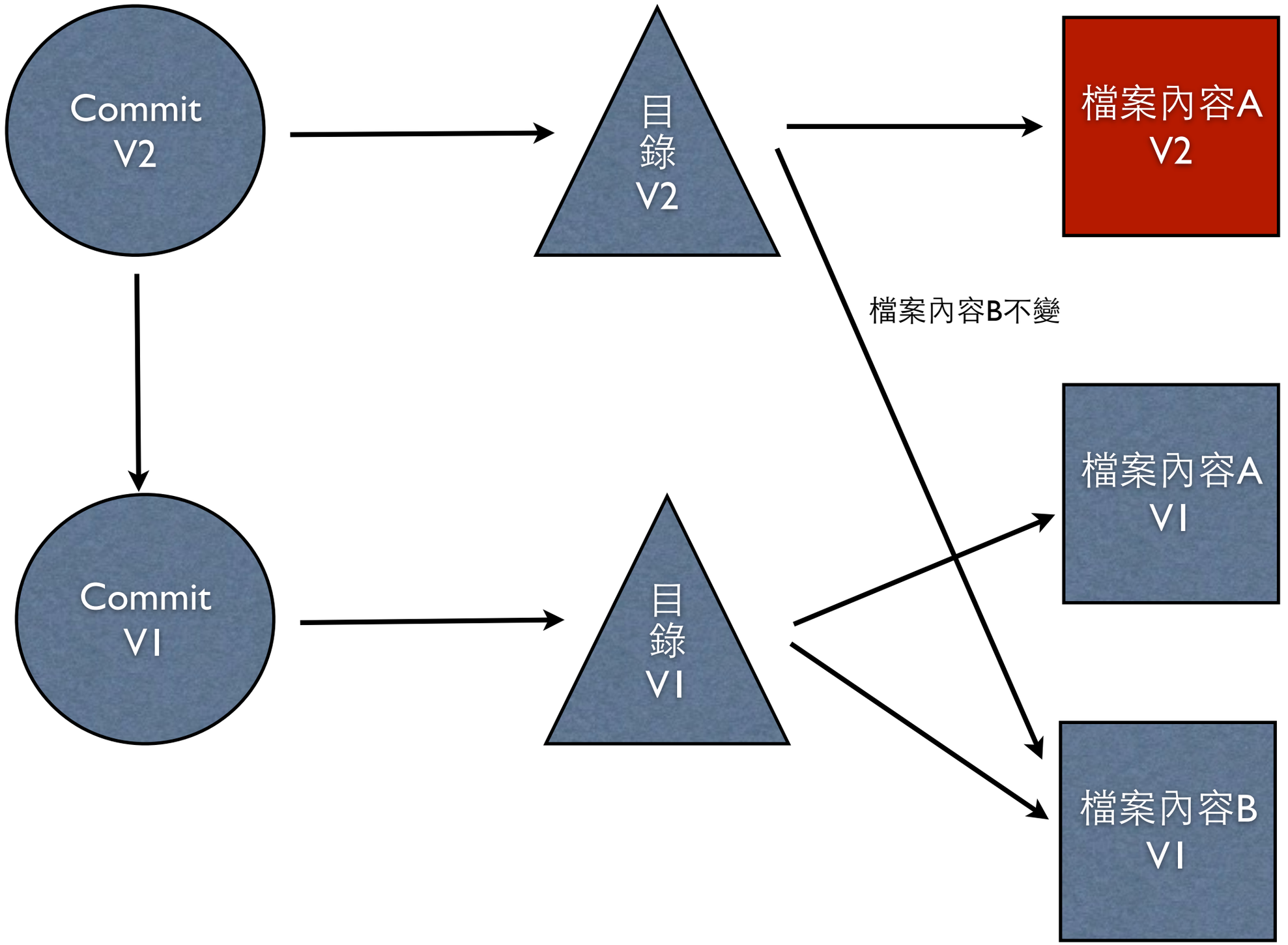
(產生commit節點，指向目錄節點)



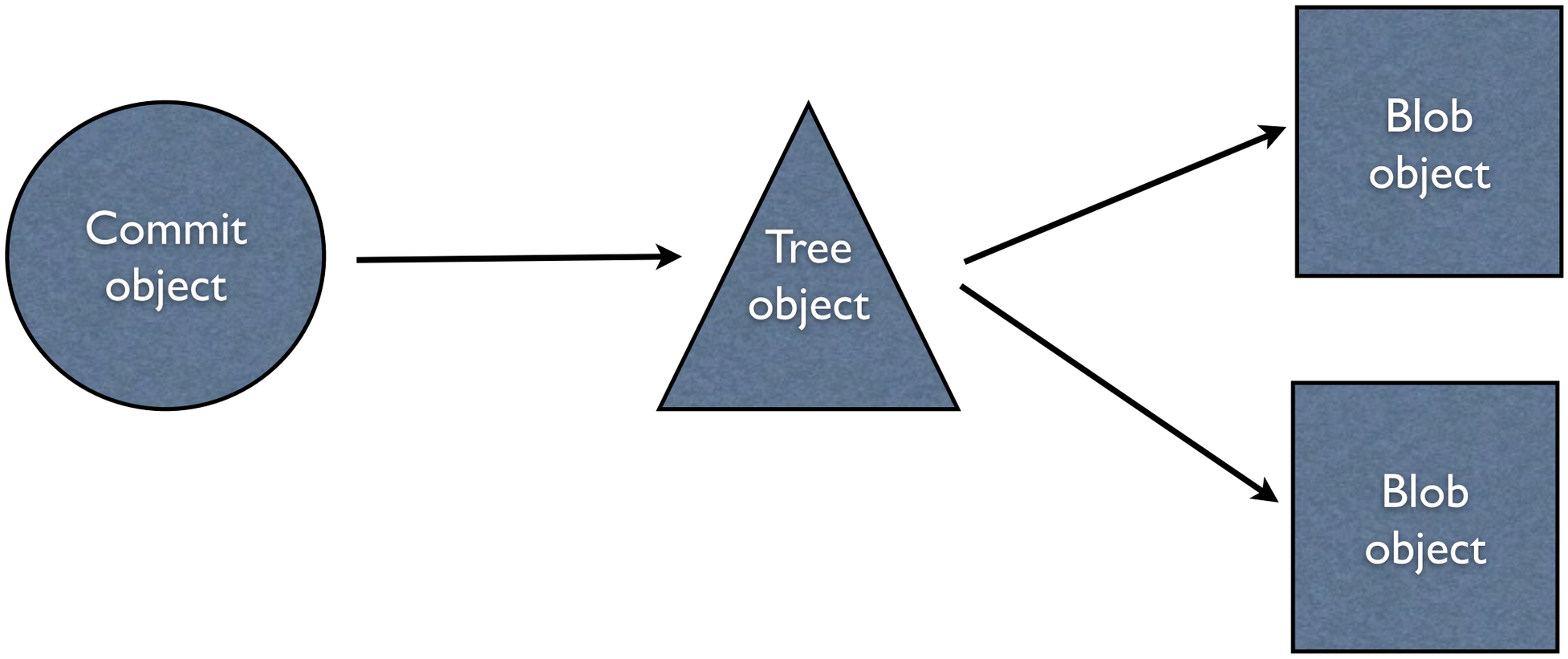
接下來我們修改檔案 A 成為 V2 版本，檔案B不變

git commit (cont.)

(產生commit V2節點，指向parent commit節點)



Git is objects database



儲存內容(demo)

- git init
- echo hello > hello.txt
- git add .
- tree .git
- 存在 .git/objects/ce/013625030ba8dba906f756967f9e9ca394464a
- 這是 hello 內容的 SHA1
  - printf "blob 6\x00hello\n" | shasum
  - echo "hello" | git hash-object --stdin
- git cat-file -p ce0136

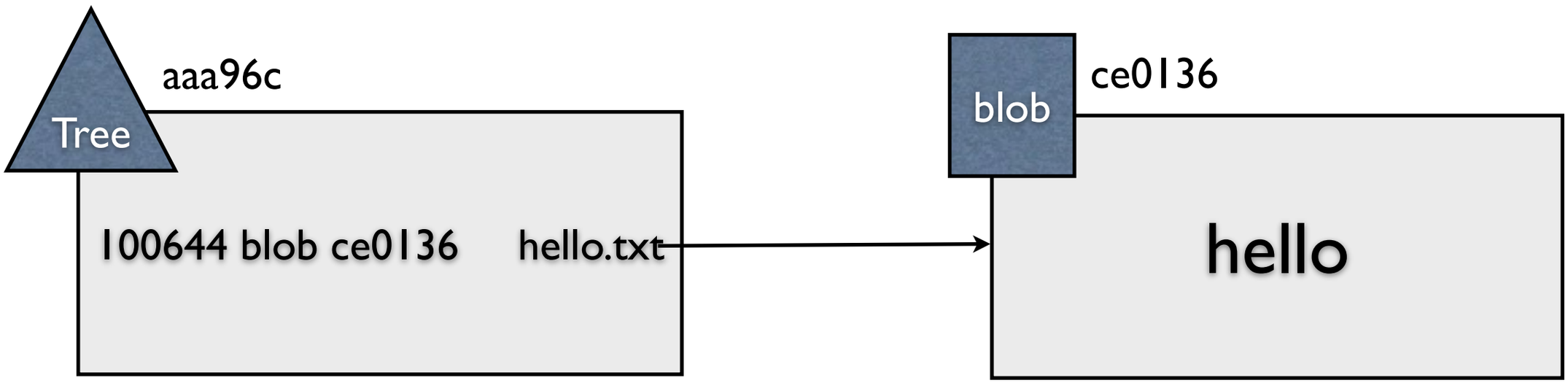
blob object 的實際檔案名稱
.git/objects/ce/013625030ba8dba906f756967f9e9ca394464a

Blob object

- Git 是 Content-addressable filesystem
- Blob 沒有 metadata，沒有檔名資訊
- Blob object 的儲存檔名，是根據內容產生的 SHA1
- 內容一樣的檔案，根據 SHA1 演算法只會存成同一份檔案，不會浪費空間

儲存目錄 (demo)

- git write-tree
  (根據 staging area 產生 Tree object)
- git cat-file -p aaa96c

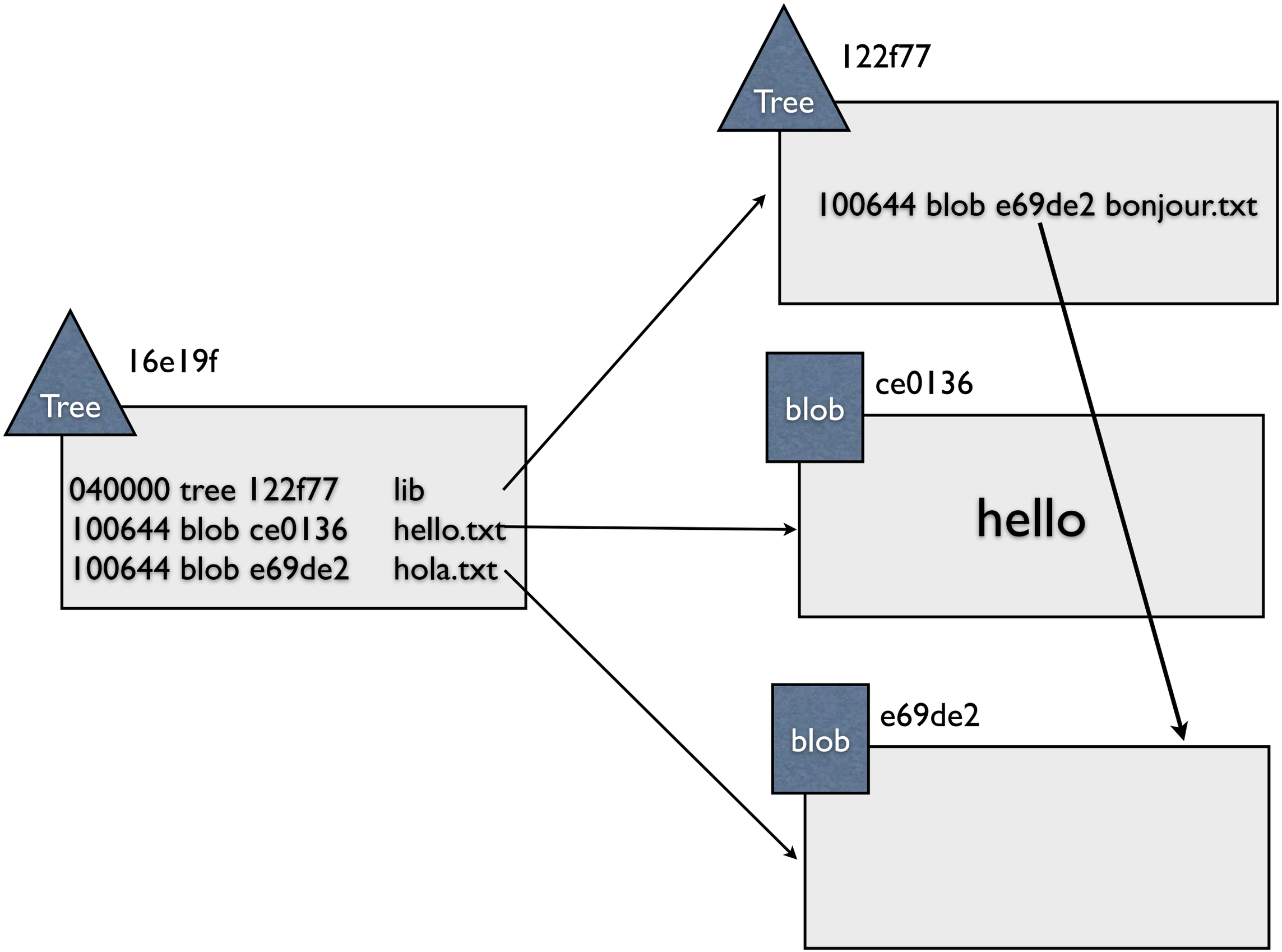


Tree object 的實際檔案名稱
.git/objects/aa/a96ced2d9a1c8e72c56b253a0e2fe78393feb7

其中 100644 為檔案模式，表示這是一個普通檔案
100755 表示可執行檔，120000 表示 symbolic link

儲存目錄 (cont.)

- 新增兩個空白檔案和子目錄
  - touch hola.txt & mkdir lib & touch lib/bonjour.txt
  - git add .
  - git write-tree
  - git cat-file -p 16e19f (觀察這個 tree)
  - git cat-file -p e69de2 (觀察其中的 lib tree)


122f77
Tree
100644 blob e69de2 bonjour.txt
16e19f
Tree
040000 tree 122f77 lib
100644 blob ce0136 hello.txt
100644 blob e69de2 hola.txt
blob
ce0136
hello
blob
e69de2

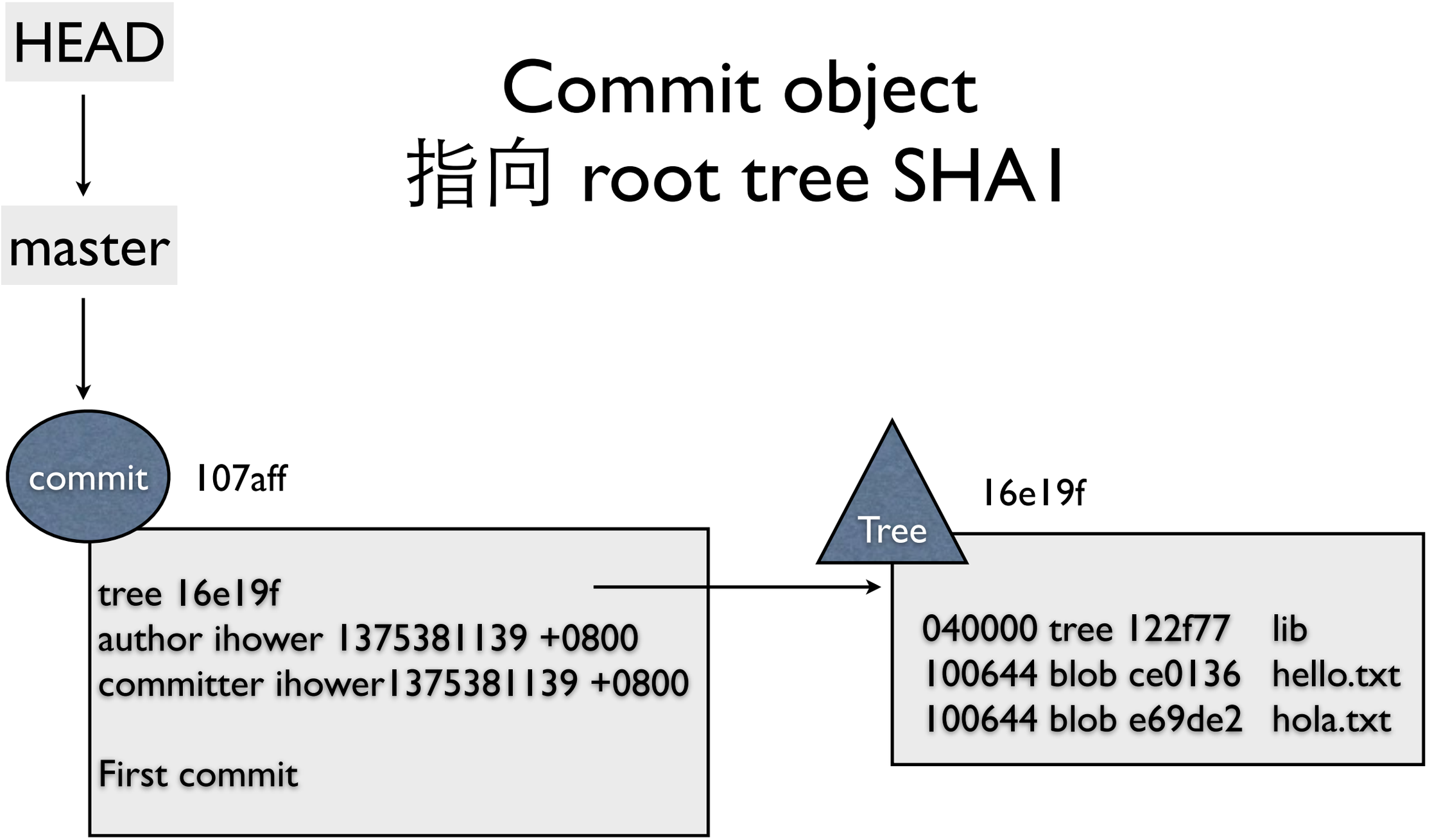
Tree object

- Git 用 Tree object 把 Blob object 組織起來，包括檔案命名和目錄結構
  - Blob object 並沒有包含檔案名稱和目錄結構
  - Tree object 裡面還可以有 Tree object 子目錄
- Tree object 的檔名，一樣是根據內容產生 SHA1

遞交 Commit (demo)

- git commit-tree 16e19f -m “First commit”
- git cat-file -p 107aff
- cat .git/HEAD
- cat .git/refs/heads/master
- git update-ref refs/heads/master 107aff
- git rev-parse HEAD

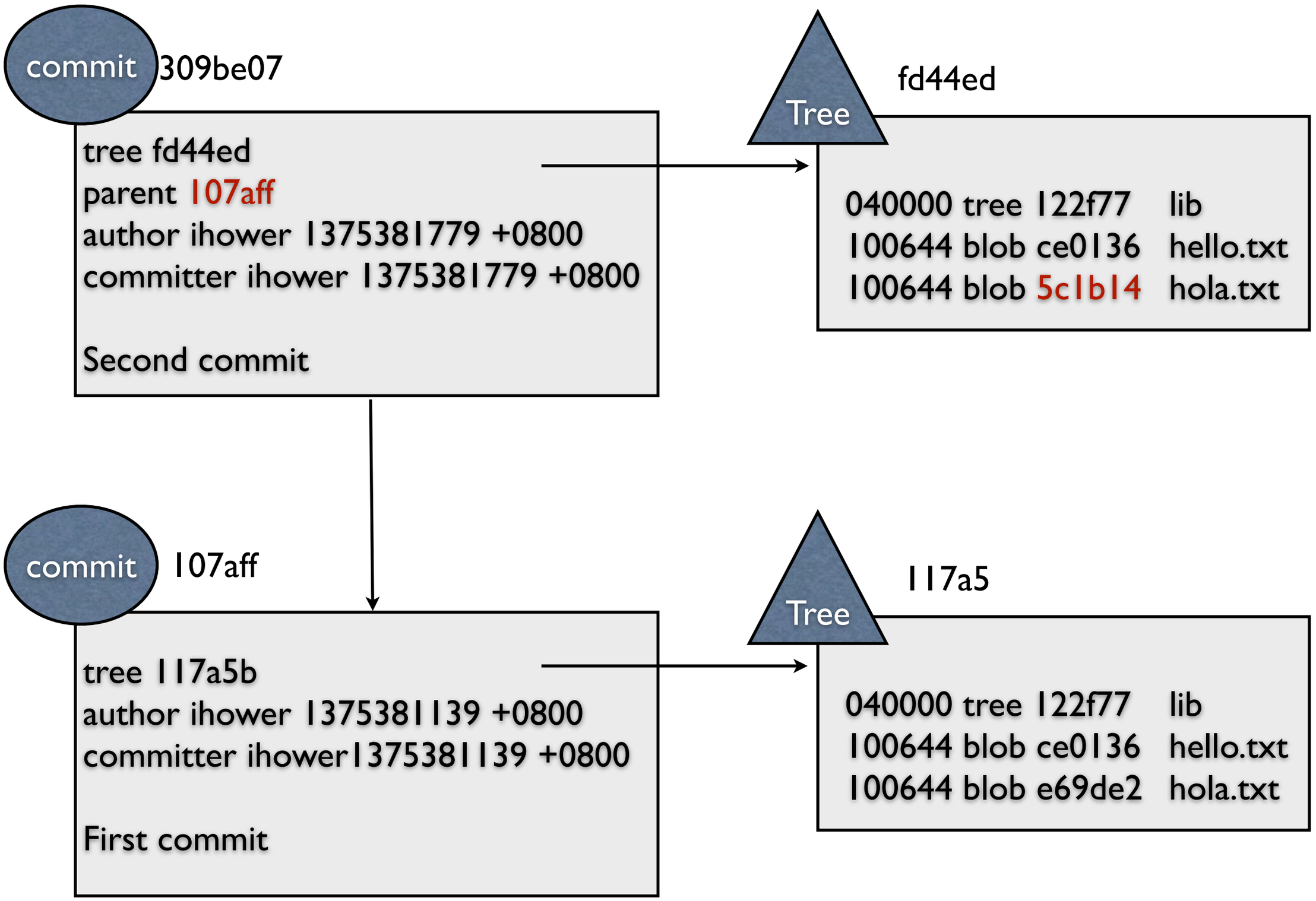
Commit object
指向 root tree SHA1



再次遞交 Commit (demo)

- 修改 hola.txt 檔案，加入 hola 字串
- git commit -am “Second commit”
- git cat-file -p 309be07

Commit object
指向 parent commit SHA1



Commit object

- 紀錄 root tree SHA1
- 紀錄 parent commit SHA1
- 紀錄作者、時間和 commit message 資訊
- Commit object 的檔名，一樣是根據內容產生 SHA1

Git commit 動作流程

- 用內容產生 blob object
- 寫入 file mode, blob SHA1, file name 到 staging area
- 根據 staging area 產生 Tree object
- 用 root tree SHA1 和 parent commit SHA1 產生 commit object
- 用 commit SHA1 更新 master 參考

測驗解答

如何不用 git add 和 git commit 指令進行 commit 動作?

git add

```
echo "hola" | git hash-object -w --stdin
git update-index --add --cacheinfo \
    100644 5c1b14949828006ed75a3e8858957f86a2f7e2eb hola.txt
```

---

git commit

```
git write-tree
git commit-tree 27b9d5 -m "Second commit" -p 30b060
git update-ref refs/heads/master 97b806c9e5561a08e0df1f1a60857baad3a1f02e
```

https://gist.github.com/ihower/6132576

Tag object

(Tag 分兩種：annotated tag 才會產生 object)

- git tag -a release
- git rev-parse release
- git cat-file -p 2450f3


tag
2450f3
object 309be0
type commit
tag release
tagger ihower 1375383070 +0800
Release!
commit
309be0

小結論:

Git 有四種 Objects

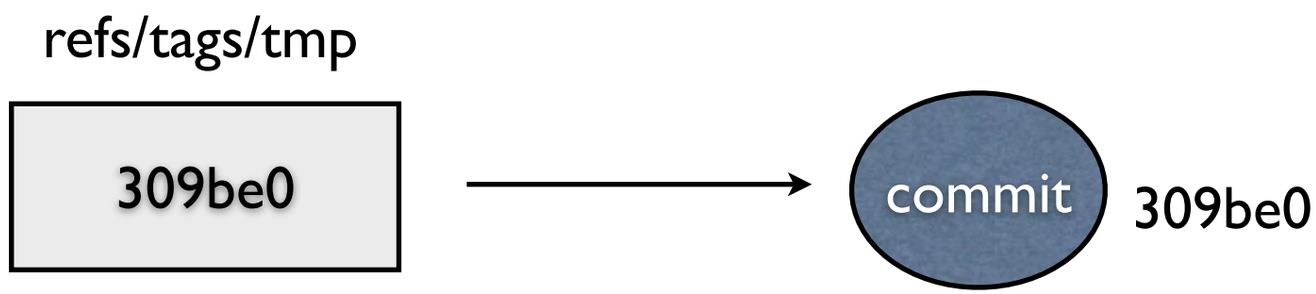
- Blob
- Tree
- Commit
- Tag

References 參照

- 單純一個檔案紀錄一個 SHA1 參照
- Tag reference
- Branch reference
- HEAD reference (指向目前所在的 branch)

Tag reference

- git tag tmp
- cat .git/refs/tags/tmp
- 不像 object 資訊豐富，reference 內容只有 Commit object SHA1



Branch 和 HEAD reference

- 每次 commit 就會變動 reference
- HEAD 指向目前在哪一個 branch
- cat .git/HEAD
- cat .git/refs/heads/master

HEAD

ref: refs/heads/
master

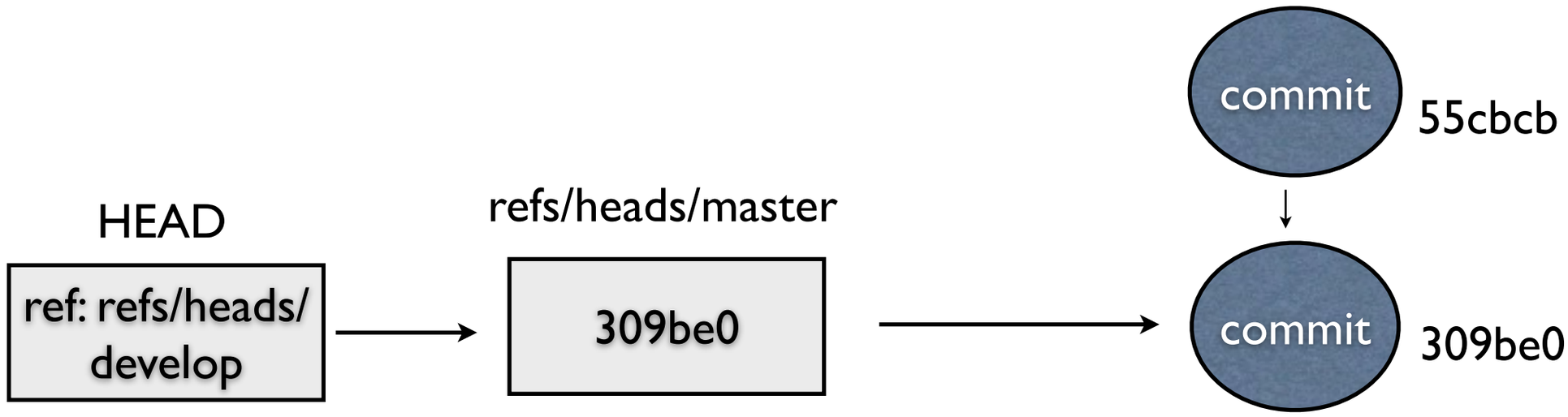
refs/heads/master

309be0

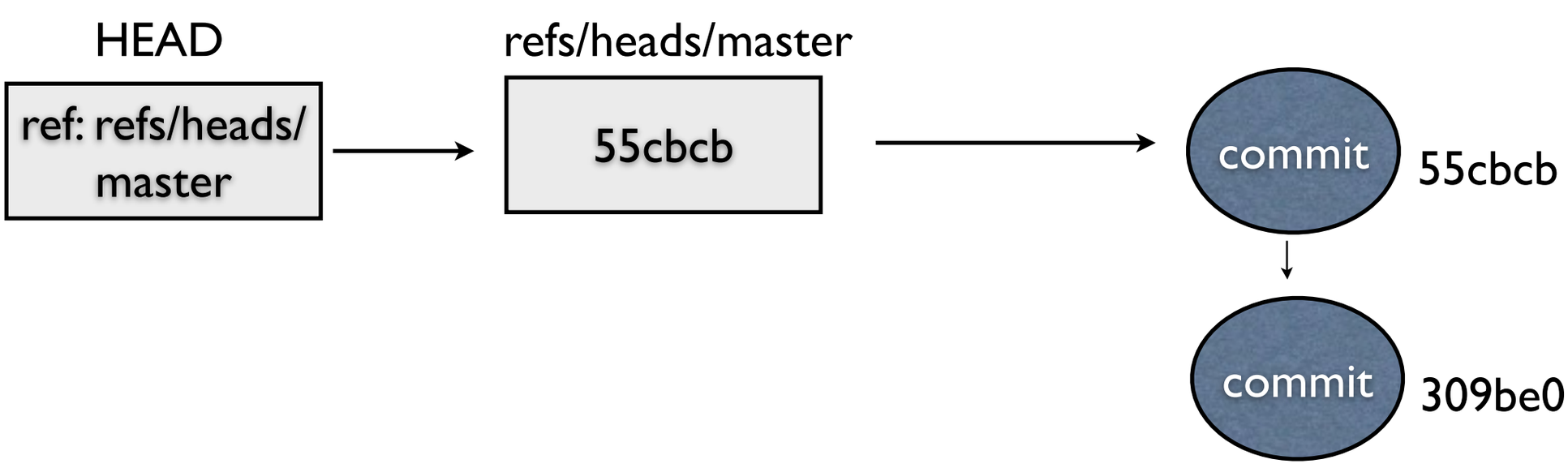
commit

309be0

如果在 Branch 上產生新 Commit...

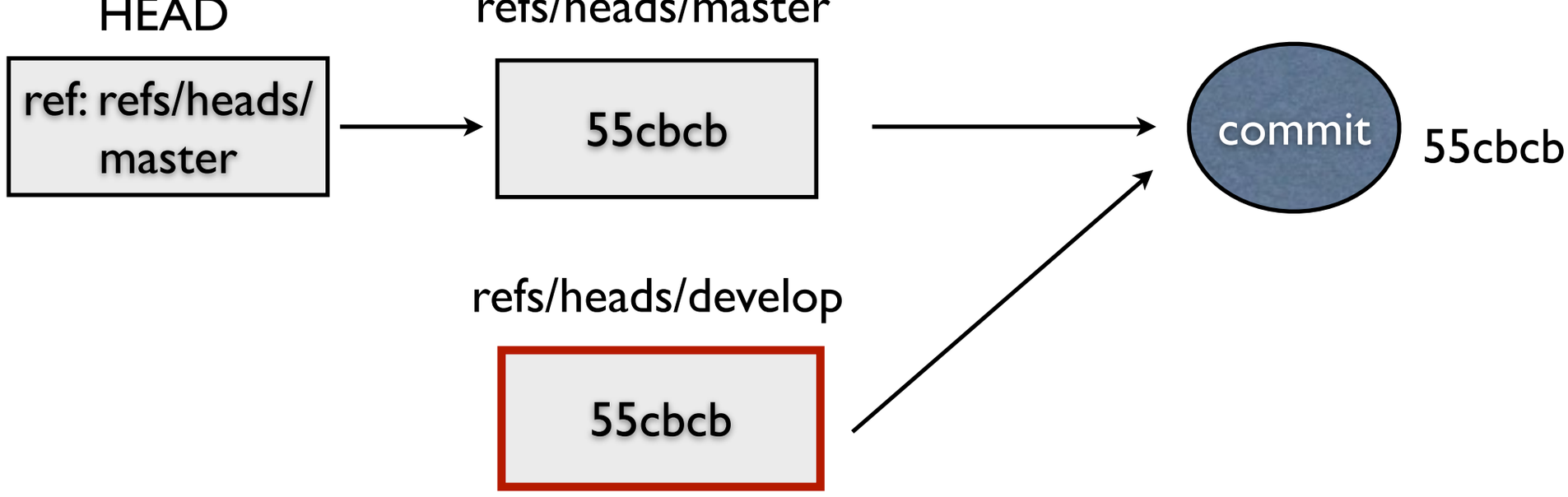


Branch reference 就會自動改指到新的 commit



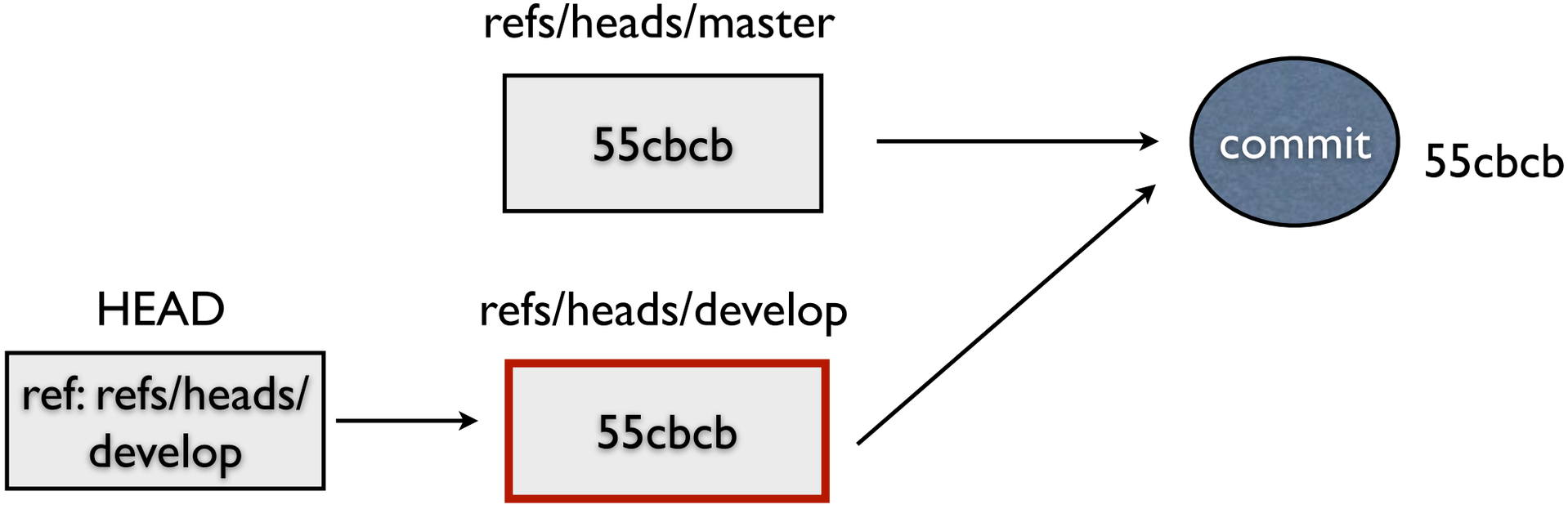
開新 Branch develop

git branch develop



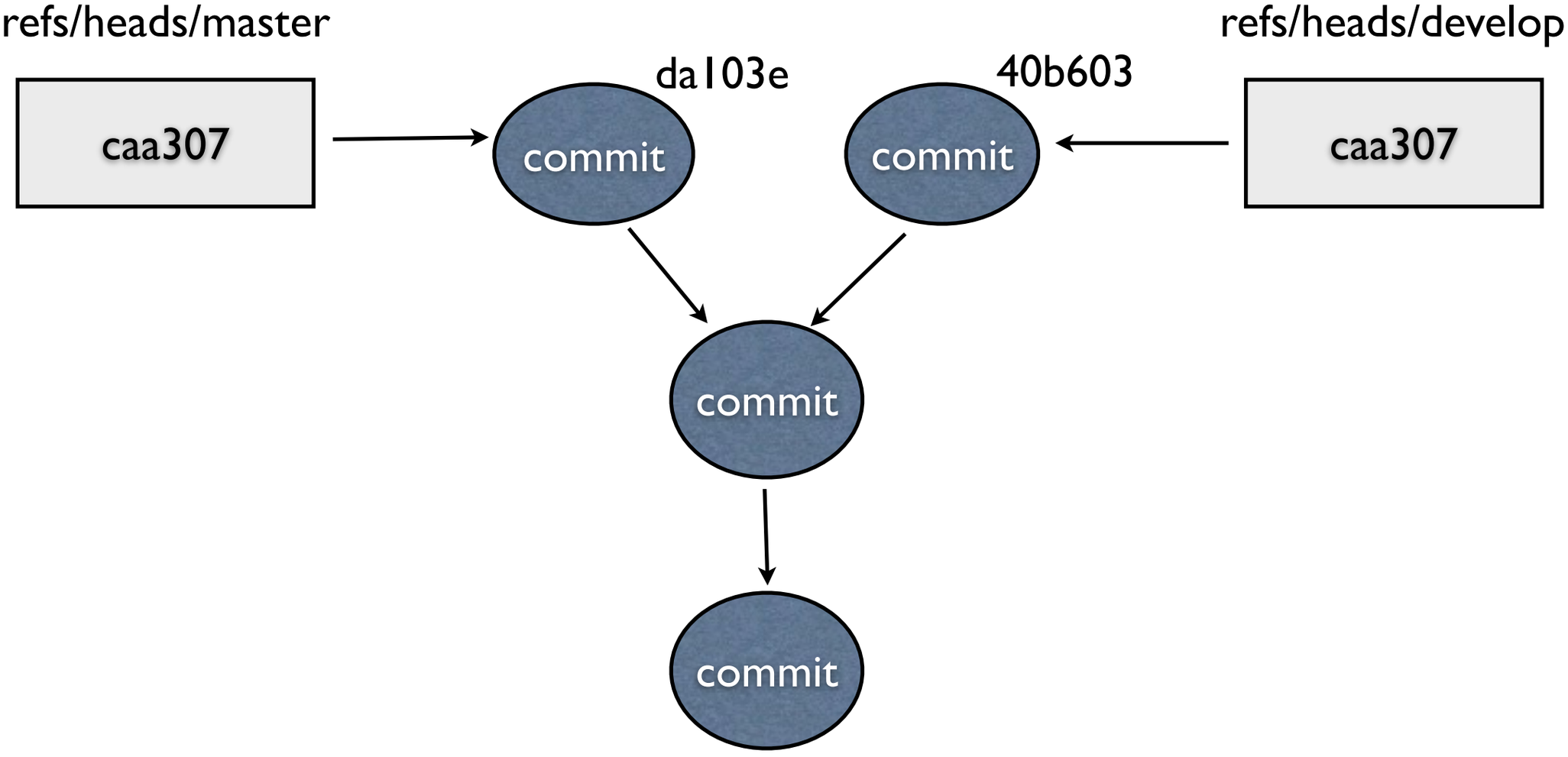
切換 Branch：改HEAD

git checkout develop

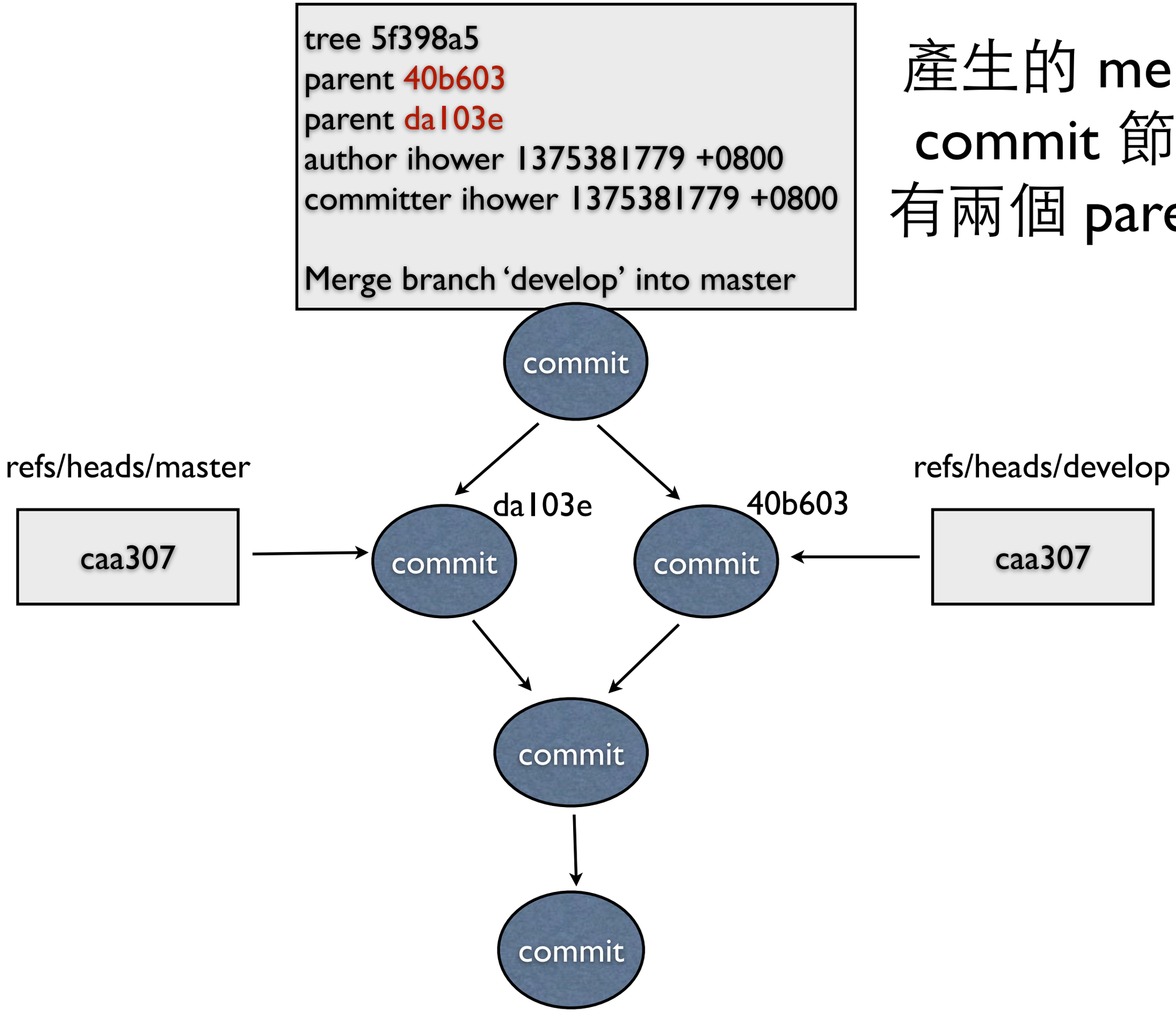


合併 Branch

git merge develop

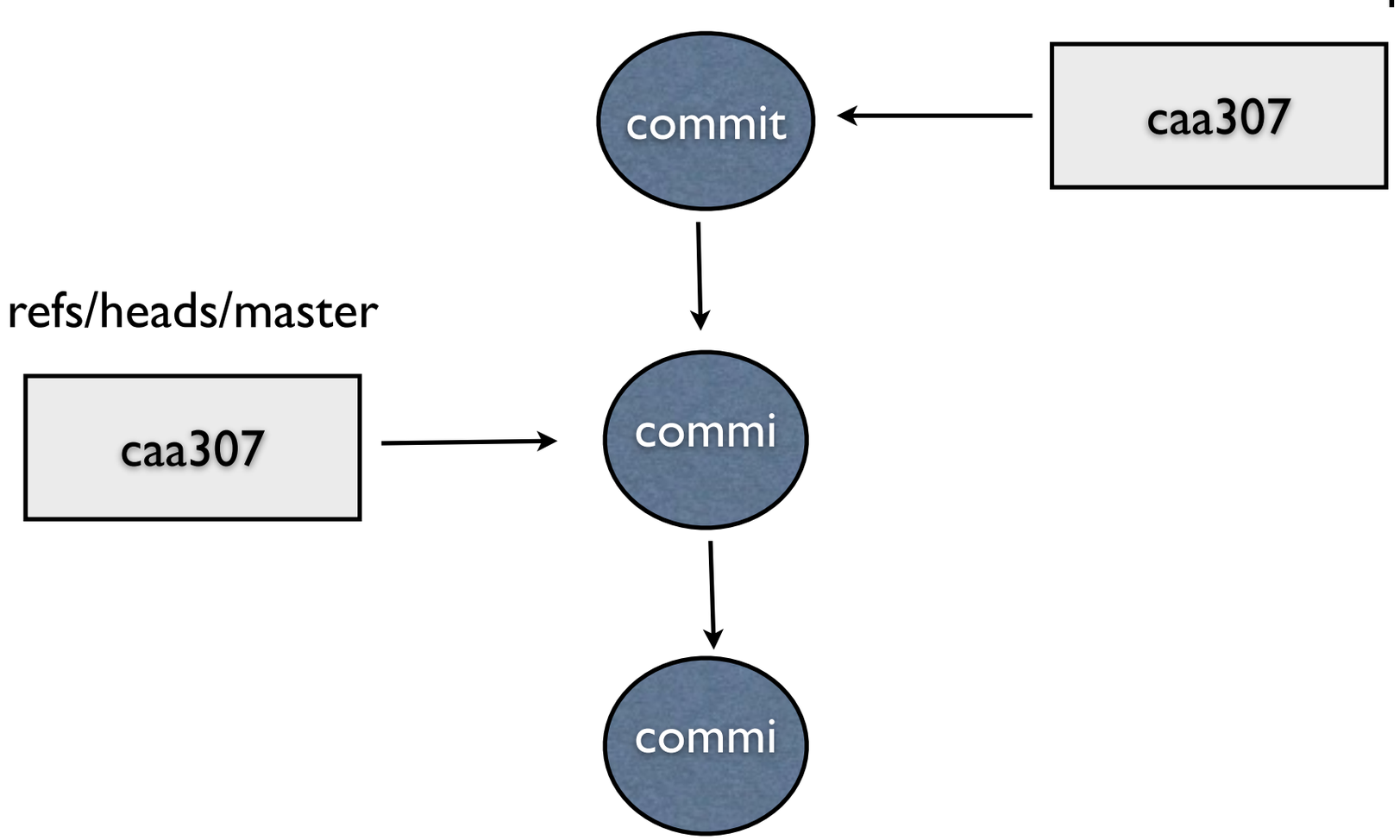


產生的 merge commit 節點有兩個 parents



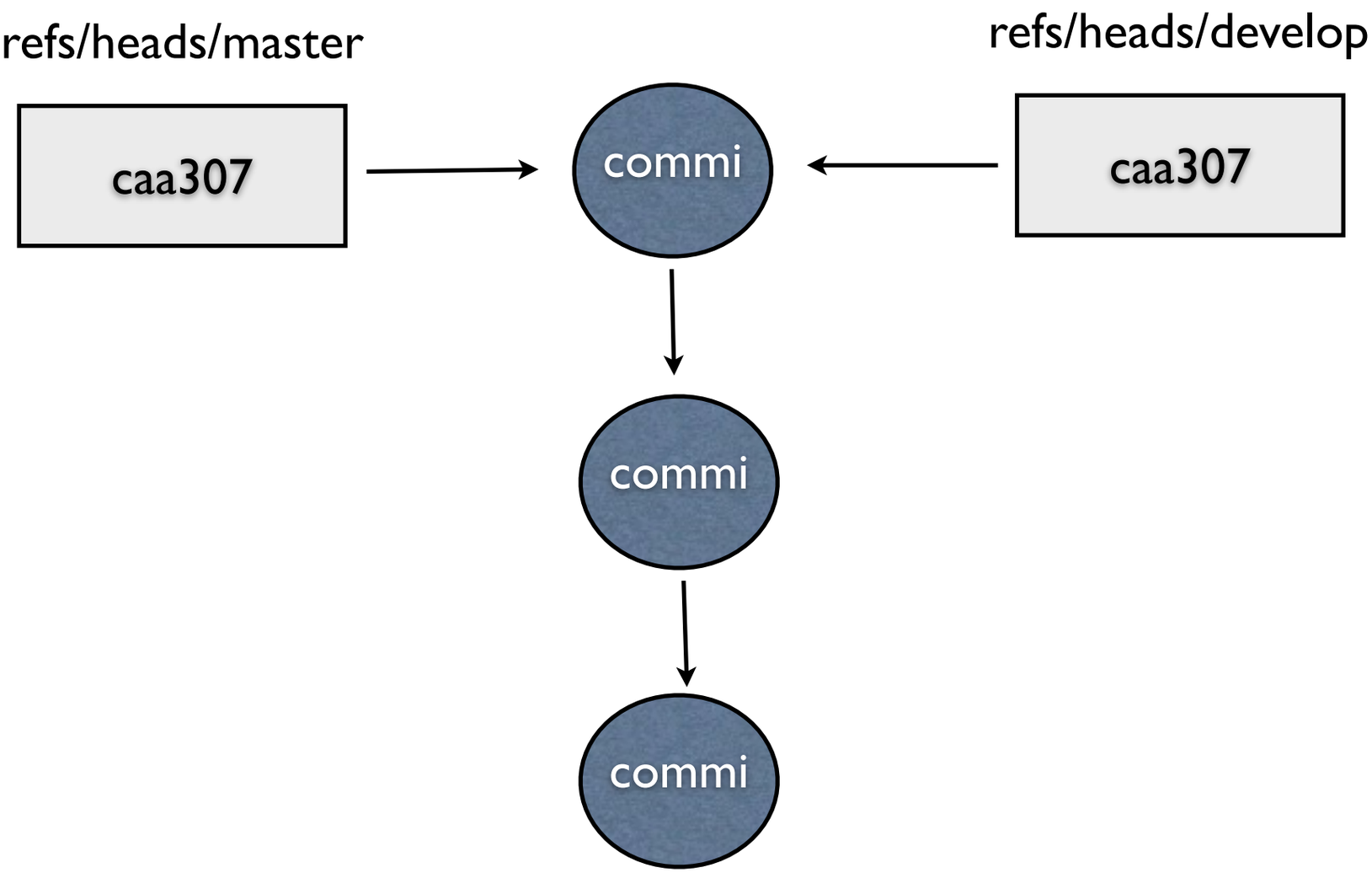
另一種合併情況 fast-forward

將 develop 合併進 master
(git merge develop)



另一種合併情況 fast-forward

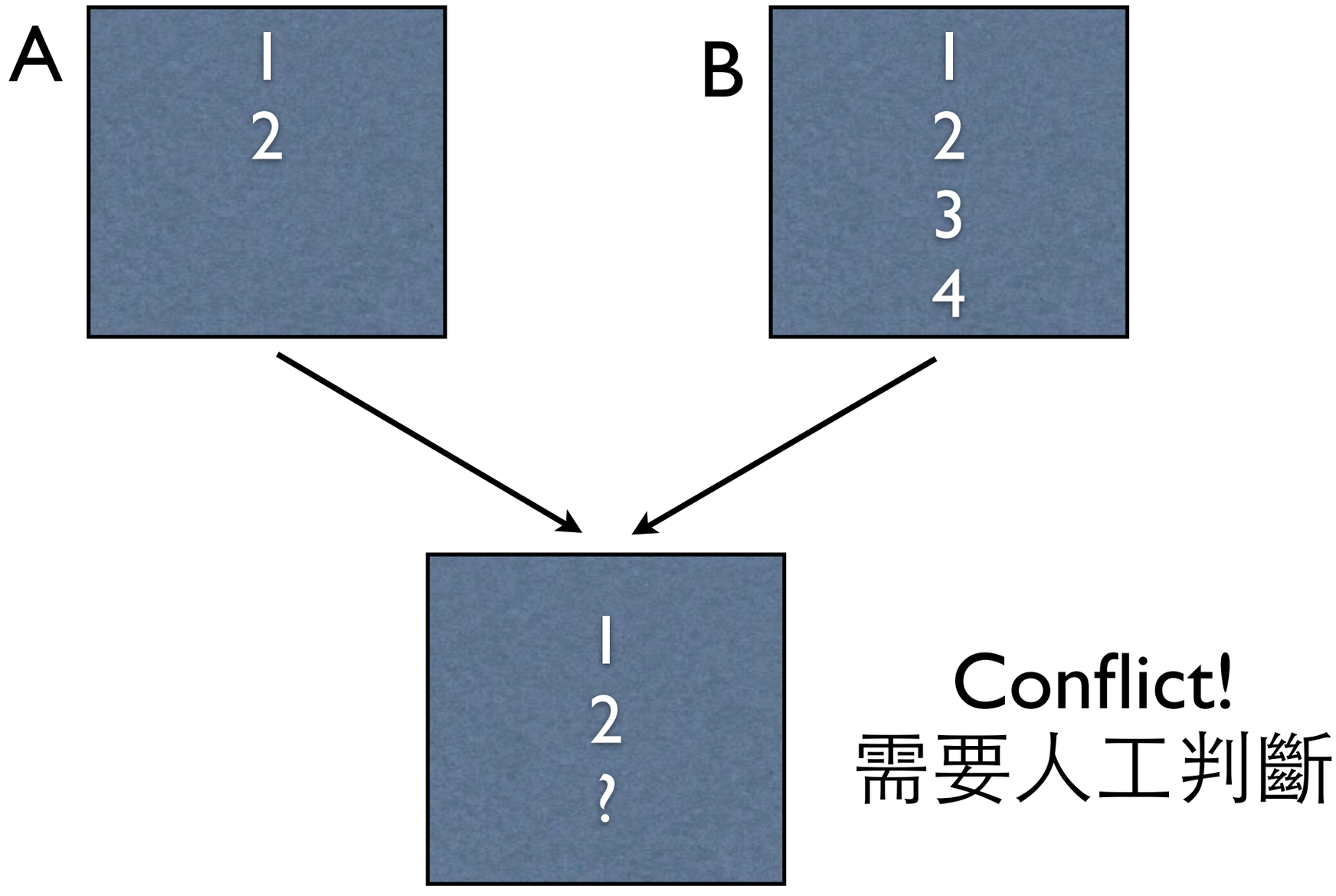
沒有產生 merge 節點，只是移動參考



Git 如何 Merge commits?

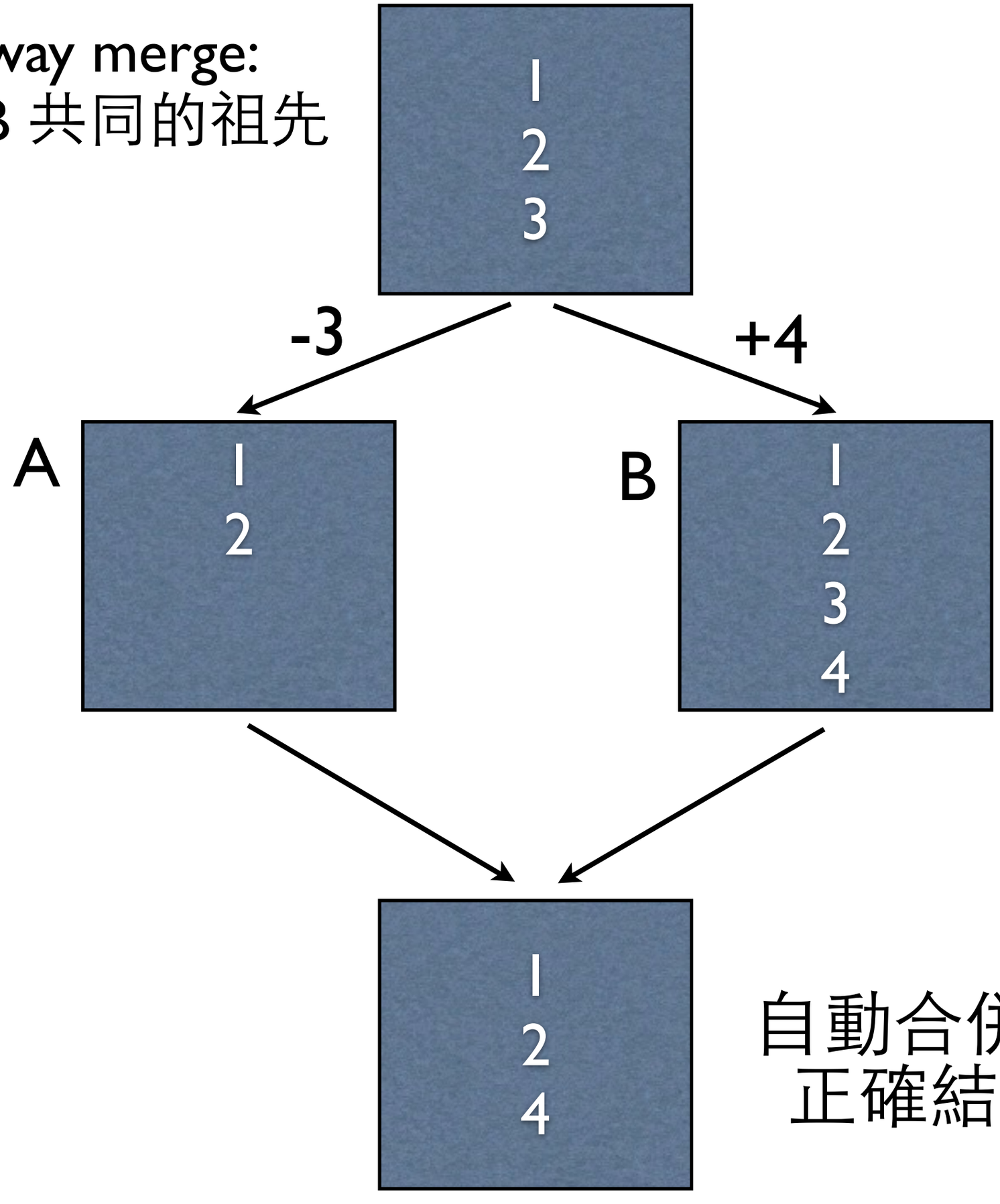
- Git 進行了一個 Three-way merge 的動作
- three-way merge 除了要合併的兩個檔案，還加上兩個檔案的共同祖先。如此可以大大減少人為處理 conflict 的情況。
- two-way merge 則只用兩個檔案進行合併 (svn預設即 two-way merge)

Two-way merge



Conflict!
需要人工判斷

Three-way merge:
先找出 AB 共同的祖先



自動合併出
正確結果

結論

additive

- 跟 Unix filesystem 有類似的結構，除了
  - Git filesystem 的設計是一直累加的，不會有東西被刪除
- Blob object 沒有 metadata

Reference is cheap

- 開新 branch 只是 refs 而已，直到 commit 前都沒有負擔。
- 不像有些VCS 開分支會複製一份原始碼，非常耗費資源。

Integrity

- SHA1 是內容的 checksum
- 如果檔案內容有損毀，就會發現跟SHA1不同。如果 tree 被偷改檔名，也會被發現。
- HEAD 指向的 SHA1，就是整個 repository 的 checksum
- 這在分散式系統非常重要：資料從一個開發者傳到另一個開發者時，確保資料沒有被修改。

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

- Linus Torvalds

謝謝，請多指教

http://ihower.tw

參考資料

- http://ihower.tw/blog/category/git
- http://pragprog.com/screencasts/v-jwsceasy/source-control-made-easy
- http://www.youtube.com/watch?v=4XpnKHJAok8 Linux 的演講
- http://www.softdevtube.com/2013/02/05/advanced-git/
- http://git-scm.com/book
- Git from the bottom up
  http://ftp.newartisans.com/pub/git.from.bottom.up.pdf
- Version Control with Git, O'Reilly
- http://nfarina.com/post/9868516270/git-is-simpler
- http://think-like-a-git.net/sections/graph-theory.html

- Git in Practice, Manning
- https://peepcode.com/products/git
- https://peepcode.com/products/advanced-git
- Git Internals, Peepcode
- Pragmatic Version Control Using Git, Pragmatic
- Pragmatic Guide to Git, Pragmatic
- Continuous Delivery Ch.14
- https://www.atlassian.com/git/tutorials/comparing-workflows
- https://guides.github.com/introduction/flow/index.html