

【冷知識】斷頭（ detached HEAD ）是怎麼一回事？

這個標題有點驚人，其實不是恐怖片的那種斷頭啦。在「[【冷知識】HEAD 是什麼東西？](#)」章節曾經介紹過，HEAD 是一個「指向某一個分支的指標」，你可以把 HEAD 當做「目前所在的分支」看待。

正常情況下，HEAD 會指向某一個分支，而分支會指向某一個 Commit。但 HEAD 偶爾會發生「沒有指到某個分支」的情況，這個狀態的 HEAD 便稱之「detached HEAD」。

可能發生這個狀態的原因有：

1. 使用 Checkout 指令直接跳到某個 Commit，而那個 Commit 剛好目前沒有分支指著它。
2. Rebase 的過程其實也是處於不斷的 detached HEAD 狀態。
3. 切換到某個遠端分支的時候。

在這個狀態可以做什麼？

其實這個狀態沒什麼特別的，就只是 HEAD 剛好指向某個沒有分支指著的 Commit 罷了，所以一樣可以跟平常一樣的操作 Git，一樣可以進行 Commit。這是原本的歷史紀錄：

All Branches ⇅ Hide Remote Branches ⇅ Ancestor Order ⇅				
Graph	Description	Commit	Author	
	master add dog 2	27f6ed6	Eddie Kao <ed	
	add dog 1	2bab3e7	Eddie Kao <ed	
	add 2 cats	ca40fc9	Eddie Kao <ed	
	add cat 2	1de2076	Eddie Kao <ed	
	add cat 1	cd82f29	Eddie Kao <ed	
	add database settings	382a2a5	Eddie Kao <ed	
	init commit	bb0c9c2	Eddie Kao <ed	

detached

我使用 Checkout 指令切換至 add cat 1 那個 Commmit：

```
$ git checkout cd82f29
Note: checking out 'cd82f29'.
```

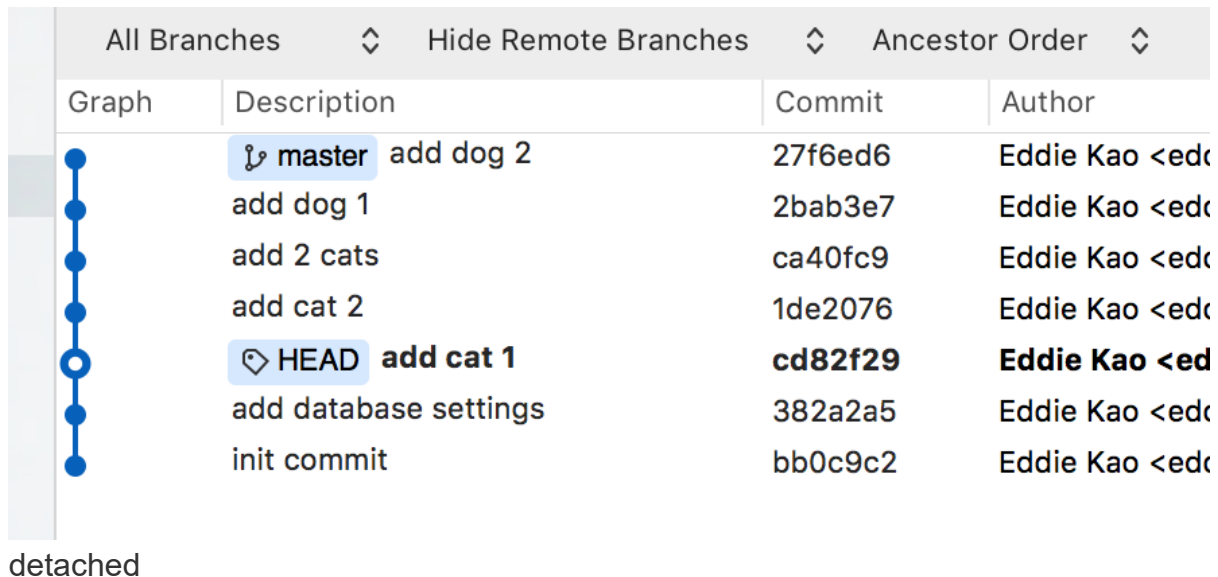
You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at cd82f29... add cat 1
```

或是直接在 SourceTree 的 Commit 上點兩下，也可以有一樣的效果。現在的狀態是這樣：



All Branches		Hide Remote Branches	Ancestor Order	
Graph	Description	Commit	Author	
●	master add dog 2	27f6ed6	Eddie Kao <edk...>	
●	add dog 1	2bab3e7	Eddie Kao <edk...>	
●	add 2 cats	ca40fc9	Eddie Kao <edk...>	
●	add cat 2	1de2076	Eddie Kao <edk...>	
○	HEAD add cat 1	cd82f29	Eddie Kao <edk...>	
●	add database settings	382a2a5	Eddie Kao <edk...>	
●	init commit	bb0c9c2	Eddie Kao <edk...>	

detached

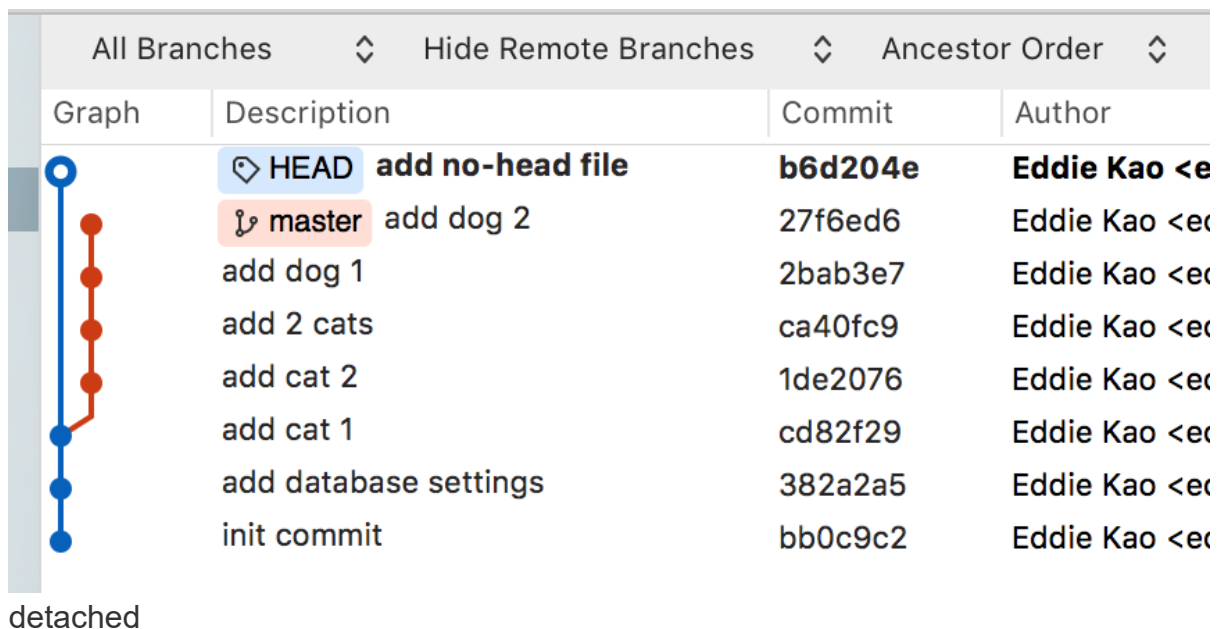
這時候我試著進行一次 Commit：

```
$ touch no-head.html

$ git add no-head.html

$ git commit -m "add a no-head file"
[detached HEAD b6d204e] add no-head file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 no-head.html
```

這時候的狀態變這樣：



有覺得有點眼熟嗎？是的，這就跟「[【狀況題】我可以從過去的某個 Commit 再長一個新的分支出來嗎？](#)」章節介紹的差不多，只是，現在這個 Commit `b6d204e` 還「沒有名字」，更正確的說是還沒有分支指向它，目前僅有 HEAD 指著它而已。

這有什麼影響嗎？影響就是當我的 HEAD 回到其它分支之後，這個 Commit 就不容易被找到（除非你有記下這個 Commit 的 SHA-1 值），如果一直沒人來找它，過久了之後就會被 Git 啟動的資源回收機制給收掉了。

所以，如果還想留下這個 Commit，就給它一個分支指著它就行了。如果你剛好就在這個 Commit 上的話：

```
$ git branch tiger
```

或是明確的跟 Git 說幫你建立一個分支指向某個 Commit：

```
$ git branch tiger b6d204e
```

雖然剛建完分支，當下的狀態還是處於 detached HEAD，不過不用太擔心，這個 Commit 以後就可以透過 `tiger` 這個分支來找到它了。你也可以使用 Checkout 指令配合 `-b` 參數，建立分支後直接切換：

```
$ git checkout -b tiger b6d204e
Switched to a new branch 'tiger'
```

為什麼切換到遠端的分支也會是這個狀態？

前面提到當 HEAD 沒有指到某個分支的時候，它會呈現 **detached** 狀態。事實上，更正確的說，應該是說「當 HEAD 沒有指到某個『本地』的分支」就會呈現這個狀態。舉例來說，有一個我剛從自己的 GitHub 帳號 Clone 下來的專案，我使用 `git branch` 指令檢視目前的分支：

```
$ git branch --remote
origin/HEAD -> origin/master
origin/master
origin/refactoring
```

使用 `--remote` 或 `-r` 參數可以顯示遠端的分支，當我試著切換到 `origin/refactoring` 這個分支的時候：

```
$ git checkout origin/refactoring
Note: checking out 'origin/refactoring'.
```

```
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
```

```
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
```

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 1ec82d7... refactored
```

OOPS！變成 detached HEAD 狀態了。要切換到遠端分支而不呈現 detached HEAD 狀態，可以加上 `--track` 或 `-t` 參數：

```
$ git checkout -t origin/refactoring
Branch refactoring set up to track remote branch refactoring from origin.
Switched to a new branch 'refactoring'
```

這樣就切過去了，那個 `-t` 參數是指會在本機建立一個名為追蹤分支 (tracking branch) 的東西。或是簡單一點直接把前面的 `origin` 拿掉：

```
$ git checkout refactoring
Branch refactoring set up to track remote branch refactoring from origin.
Switched to a new branch 'refactoring'
```

也會有一樣的效果。關於遠端分支的內容，會在「遠端共同協作」的相關章節再說明。

怎麼離開 detached HEAD 狀態？

既然已經知道所謂的 **detached HEAD** 狀態只是 **HEAD** 沒有指向任何分支造成的，要脫離這個狀態，只要讓 **HEAD** 有任何分支可以指就行了，例如讓它回到 **master** 分支：

```
$ git checkout master  
Switched to branch 'master'
```

這樣一來 **HEAD** 就解除 **detached** 狀態了。

資料來源

- <https://gitbook.tw/chapters/faq/detached-head.html>