

Branch: master Learn-Git-in-30-days / zh-tw / 13.md

Find file Copy path

doggy8088 微調第13天的一些文字 by #38

10fc2da on 18 Sep

2 contributors

371 lines (266 sloc) 15.7 KB

第 13 天：暫存工作目錄與索引的變更狀態

有沒有遇過這種情境，某個系統開發寫到一半，結果被老闆或客戶「插單」，被要求緊急修正一個現有系統的 Bug 或添加一個功能，眼前的程式即將完成，老闆的「急件」又不能拖，一個未完成的軟體開發狀態外加緊急調整的需求，這簡直是軟體品質的一大考驗。如果你有這種困擾，那麼 Git 可以漂亮的幫你完成任務。

認識 git stash 指令

我們知道使用 Git 版控的時候，有區分「工作目錄」與「索引」。工作目錄裡面會有你改到一半還沒改完的檔案(尚未加入索引)，也有新增檔案但還沒加入的檔案(尚未加入索引)。而放在索引的資料，則是你打算透過 `git commit` 建立版本(建立 commit 物件)的內容。

當你功能開發到一半，被緊急插單一定手忙腳亂，尤其是手邊正改寫到一半的那些程式碼不知該如何是好。在 Git 裡有個 `git stash` 指令，可以自動幫你把改寫到一半的那些檔案建立一個「特殊的版本」(也是一個 commit 物件)，我們稱這些版本為 stash 版本，或你可以直接稱他為「暫存版」。

建立暫存版本

我們手邊改到一半的檔案，可能會有以下狀態：

- 新增檔案 (尚未列入追蹤的檔案) (untracked files)
- 新增檔案 (已經加入索引的檔案) (tracked/staged files)
- 修改檔案 (尚未加入索引的檔案) (tracked/unstaged files)
- 修改檔案 (已經加入索引的檔案) (tracked/staged files)
- 刪除檔案 (尚未加入索引的檔案) (tracked/unstaged files)
- 刪除檔案 (已經加入索引的檔案) (tracked/staged files)

若要將這些開發到一半的檔案建立一個「暫存版」，你有兩個選擇：

- `git stash` 會將所有已列入追蹤(tracked)的檔案建立暫存版
- `git stash -u` 會包括所有已追蹤或未追蹤的檔案，全部都建立成暫存版

註: `git stash` 也可以寫成 `git stash save`，兩個指令的結果是一樣的，只是 `save` 參數可以忽略不打而已。

我們來看看一個簡單的例子。我們先透過以下指令快速建立一個擁有兩個版本的 Git 儲存庫與工作目錄：

```
mkdir git-stash-demo
cd git-stash-demo
git init

echo > a.txt
git add .
git commit -m "Initial commit"

echo 1 > a.txt
git add .
git commit -m "a.txt: set 1 as content"
```

目前的「工作目錄」是乾淨的，沒有任何更新到一半的檔案：

```
C:\git-stash-demo>git log
commit 95eff6b19a9494667985ed5da37427bb08b8cdd7
Author: Will <doggy.huang@gmail.com>
Date:   Fri Oct 11 08:17:15 2013 +0800

    a.txt: set 1 as content

commit 346fadefdd6ed2c562201b5ca37d1e4d26b26d54
Author: Will <doggy.huang@gmail.com>
Date:   Fri Oct 11 08:17:14 2013 +0800

    Initial commit

C:\git-stash-demo>git status
# On branch master
nothing to commit, working directory clean

C:\git-stash-demo>dir
磁碟區 C 中的磁碟是 System
磁碟區序號: 361C-6BD6

C:\git-stash-demo 的目錄

2013/10/11 上午 08:17 <DIR>      .
2013/10/11 上午 08:17 <DIR>      ..
2013/10/11 上午 08:17          4 a.txt
                1 個檔案          4 位元組
                2 個目錄    9,800,470,528 位元組可用
```

接著我新增一個 b.txt，再將 a.txt 的內容改成 2，如下：

```
C:\git-stash-demo>type a.txt
1

C:\git-stash-demo>echo 2 > a.txt

C:\git-stash-demo>type a.txt
2

C:\git-stash-demo>echo TEST > b.txt

C:\git-stash-demo>dir
磁碟區 C 中的磁碟是 System
磁碟區序號: 361C-6BD6

C:\git-stash-demo 的目錄

2013/10/11 上午 08:55 <DIR>      .
2013/10/11 上午 08:55 <DIR>      ..
2013/10/11 上午 08:54          4 a.txt
2013/10/11 上午 08:55          7 b.txt
                2 個檔案          11 位元組
                2 個目錄    9,704,288,256 位元組可用

C:\git-stash-demo>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:   a.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       b.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

現在我們用 `git status` 得出我們有兩個檔案有變更，一個是 a.txt 處於 "not staged" 狀態，而 b.txt 則是 "untracked" 狀態。

這時，我們利用 `git stash -u` 即可將目前這些變更全部儲存起來 (包含 untracked 檔案)，儲存完畢後，這些變更全部都會被重置，新增的檔案會被刪除、修改的檔案會被還原、刪除的檔案會被加回去，讓我們目前在工作目錄中所做的變更全部回復到 HEAD 狀態。這就是 Stash 幫我們做的事。如下所示：

```
C:\git-stash-demo>git stash -u
Saved working directory and index state WIP on master: 95eff6b a.txt: set 1 as content
HEAD is now at 95eff6b a.txt: set 1 as content

C:\git-stash-demo>git status
# On branch master
nothing to commit, working directory clean
```

在建立完成「暫存版」之後，Git 會順便幫我們建立一個暫存版的「參考名稱」，而且是「一般參考」，在 `.git\refs\stash` 儲存的是一個 commit 物件的「絕對名稱」：

```
C:\git-stash-demo>dir .git\refs\
磁碟區 C 中的磁碟是 System
磁碟區序號: 361C-6BD6

C:\git-stash-demo\.git\refs 的目錄

2013/10/11 上午 08:57 <DIR> .
2013/10/11 上午 08:57 <DIR> ..
2013/10/11 上午 08:57 <DIR> heads
2013/10/11 上午 08:57 41 stash
2013/10/11 上午 08:17 <DIR> tags
1 個檔案 41 位元組
4 個目錄 9,701,650,432 位元組可用
```

我們用 `git cat-file -p stash` 即可查出該物件的內容，這時你可以發現它其實就是個具有三個 parent (上層 commit 物件) 的 commit 物件：

```
C:\git-stash-demo>git cat-file -p stash
tree 86cf41ab650d8d0ce5fdd003bb7b722a917438a2
parent 95eff6b19a9494667985ed5da37427bb08b8cdd7
parent b79c4650e72ad4627d691a2d6cfb192626e24e94
parent 9b4e4a100776783dc76d16c3872235e6314d15e3
author Will <doggy.huang@gmail.com> 1381453062 +0800
committer Will <doggy.huang@gmail.com> 1381453062 +0800

WIP on master: 95eff6b a.txt: set 1 as content
```

有三個 parent commit 物件的意義就在於，這個特殊的暫存版是從另外三個版本合併進來的，然而這三個版本的內容，我們一樣可以透過相同的指令顯示其內容：

```
C:\git-stash-demo>git cat-file -p 95ef
tree eba2ef4205738a5015fc47d9cfe634d7d5eae466
parent 346fade9dd6ed2c562201b5ca37d1e4d26b26d54
author Will <doggy.huang@gmail.com> 1381450635 +0800
committer Will <doggy.huang@gmail.com> 1381450635 +0800

a.txt: set 1 as content

C:\git-stash-demo>git cat-file -p b79c
tree eba2ef4205738a5015fc47d9cfe634d7d5eae466
parent 95eff6b19a9494667985ed5da37427bb08b8cdd7
author Will <doggy.huang@gmail.com> 1381453061 +0800
committer Will <doggy.huang@gmail.com> 1381453061 +0800

index on master: 95eff6b a.txt: set 1 as content

C:\git-stash-demo>git cat-file -p 9b4e
tree b583bfe854b66756dd0f8ee96cab0c898193b5fd
author Will <doggy.huang@gmail.com> 1381453062 +0800
committer Will <doggy.huang@gmail.com> 1381453062 +0800

untracked files on master: 95eff6b a.txt: set 1 as content
```

從上述執行結果你應該可以從「訊息紀錄」的地方清楚看出這三個版本分別代表那些內容：

1. 原本工作目錄的 HEAD 版本
2. 原本工作目錄裡所有追蹤中的內容 (在索引中的內容)

3. 原本工作目錄裡所有未追蹤的內容 (不在索引中的內容)

也就是說，他把「原本工作目錄的 HEAD 版本」先建立兩個暫時的分支，這兩個分支分別就是「原本工作目錄裡所有追蹤中的內容」與「原本工作目錄裡所有未追蹤的內容」之用，並在個別分支建立了一個版本以產生 `commit` 物件並且給予預設的 `log` 內容。最後把這三個分支，合併到一個「參照名稱」為 `stash` 的版本（這也是個 `commit` 物件）。不僅如此，他還把整個「工作目錄」強迫重置為 `HEAD` 版本，把這些變更與新增的檔案都給還原，多的檔案也會被移除。

取回暫存版本

由於「工作目錄」已經被重置，所以變更都儲存到 `stash` 這裡，哪天如果你想要把這個暫存檔案取回，就可以透過 `git stash pop` 重新「合併」回來。如下所示：

```
C:\git-stash-demo>git status
# On branch master
nothing to commit, working directory clean

C:\git-stash-demo>git stash pop
# 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:   a.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       b.txt
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (0e5b72c96fcf693e0402c40cd58f980bb3ff7efd)
```

執行完畢後，所有當初的工作目錄狀態與索引狀態都會被還原。事實上 `Git` 骨子裡是透過「合併」的功能把這個名為 `stash` 的版本給合併回目前分支而已。最後，它還會自動將這個 `stash` 分支給刪除，所以稱它為【暫存版】非常貼切！

建立多重暫存版

`Git` 的 `stash` 暫存版可以不只一份，你也可以建立多份暫存檔，以供後續使用。不過，在正常的開發情境下，通常不會有太多暫存版才對，會有這種情況發生，主要有兩種可能：

1. 你的開發習慣太差，導致累積一堆可能用不到的暫存版。
2. 你老闆或客戶「插單」的問題十分嚴重，經常改到一半就被迫插單。(這就是身為 `IT` 人的 `BI` 啊~~~XD) (`BI` = `Business Intelligence` 或另一層意思... Well, you know....)

我們延續上一個例子，目前工作目錄的狀態應該是有兩個檔案有變化，我們用 `git status -s` 取得工作目錄的狀態(其中 `-s` 代表顯示精簡版的狀態)：

```
C:\git-stash-demo>git status -s
M a.txt
?? b.txt
```

現在，我們先建立第一個 `stash` 暫存版：

```
C:\git-stash-demo>git stash save -u
Saved working directory and index state WIP on master: 95eff6b a.txt: set 1 as content
HEAD is now at 95eff6b a.txt: set 1 as content
```

然後透過 `git stash list` 列出目前所有的 `stash` 清單，目前僅一份暫存版：

```
C:\git-stash-demo>git stash list
stash@{0}: WIP on master: 95eff6b a.txt: set 1 as content
```

而且你可以看到建立暫存版之後，工作目錄是乾淨的。此時我們在建立另一個 `new.txt` 檔案，並且再次建立暫存版：

```
C:\git-stash-demo>git status -s
```

```
C:\git-stash-demo>echo 1 > new.txt

C:\git-stash-demo>git status -s
?? new.txt

C:\git-stash-demo>git stash save -u
Saved working directory and index state WIP on master: 95eff6b a.txt: set 1 as content
HEAD is now at 95eff6b a.txt: set 1 as content
```

我們在再一次 `git stash list` 就可以看到目前有兩個版本：

```
C:\git-stash-demo>git stash list
stash@{0}: WIP on master: 95eff6b a.txt: set 1 as content
stash@{1}: WIP on master: 95eff6b a.txt: set 1 as content
```

你應該會很納悶，都沒有自訂的註解，過了幾天不就忘記這兩個暫存檔各自的修改項目嗎？沒錯，所以你可以自訂「暫存版」的紀錄訊息。我們透過 `git stash save -u <message>` 指令，就可自訂暫存版的註解：

```
C:\git-stash-demo>git stash -h
usage: git core\git-stash list [<options>]
or: git core\git-stash show [<stash>]
or: git core\git-stash drop [-q|--quiet] [<stash>]
or: git core\git-stash ( pop | apply ) [--index] [-q|--quiet] [<stash>]
or: git core\git-stash branch <branchname> [<stash>]
or: git core\git-stash [save [--patch] [-k|--[no-]keep-index] [-q|--quiet]
                        [-u|--include-untracked] [-a|--all] [<message>]]
or: git core\git-stash clear

C:\git-stash-demo>git stash pop
Already up-to-date!
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       new.txt
nothing added to commit but untracked files present (use "git add" to track)
Dropped refs/stash@{0} (5800f37937aea5fb6a1aba0d5a1598a940e70c96)

C:\git-stash-demo>git stash save -u "新增 new.txt 檔案"
Saved working directory and index state On master: 新增 new.txt 檔案
HEAD is now at 95eff6b a.txt: set 1 as content

C:\git-stash-demo>git stash list
stash@{0}: On master: 新增 new.txt 檔案
stash@{1}: WIP on master: 95eff6b a.txt: set 1 as content
```

這時，如果你直接執行 `git stash pop` 的話，他會取回最近的一筆暫存版，也就是上述例子的 `stash@{0}` 這一項，並且把這一筆刪除。另一種取回暫存版的方法是透過 `git stash apply` 指令，唯一差別則是取回該版本 (其實是執行合併動作) 後，該暫存版還會留在 `stash` 清單上。

如果你想取回「特定一個暫存版」，你就必須在最後指名 `stash id`，例如 `stash@{1}` 這樣的格式。例如如下範例，我使用 `git stash apply "stash@{1}"` 取回前一個暫存版，但保留這版在 `stash` 清單裡：

```
C:\git-stash-demo>git stash list
stash@{0}: On master: 新增 new.txt 檔案
stash@{1}: WIP on master: 95eff6b a.txt: set 1 as content

C:\git-stash-demo>git stash apply "stash@{1}"
# 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:   a.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       b.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

```
C:\git-stash-demo>git stash list
stash@{0}: On master: 新增 new.txt 檔案
stash@{1}: WIP on master: 95eff6b a.txt: set 1 as content
```

如果確定合併正確，你想刪除 `stash@{1}` 的話，可以透過 `git stash drop "stash@{1}"` 將特定暫存版刪除。

```
C:\git-stash-demo>git stash drop "stash@{1}"
Dropped stash@{1} (118cb8a7c0b763c1343599027d79f7b20df57ebf)
```

```
C:\git-stash-demo>git stash list
stash@{0}: On master: 新增 new.txt 檔案
```

如果想清理掉所有的暫存版，直接下達 `git stash clear` 即可全部刪除。

```
C:\git-stash-demo>git stash list
stash@{0}: On master: 新增 new.txt 檔案
```

```
C:\git-stash-demo>git stash clear
```

```
C:\git-stash-demo>git stash list
```

今日小結

Git 的 `stash` (暫存版) 機制非常非常的實用，尤其是在 IT 業界插單嚴重的工作環境下 (不只台灣這樣，世界各地的 IT 業界應該也差不多)，這功能完全為我們量身打造，非常的貼心。在 `Subversion` 裡就沒有像 `Git` 這麼簡單，一個指令就可以把工作目錄與索引的狀態全部存起來。

本篇文章也試圖透過指令了解 `stash` 的核心機制，其實就是簡單的「分支」與「合併」而已，由此可知，整套 `Git` 版本控管機制，大多是以「分支」與「合併」的架構在運作。

我重新整理一下本日學到的 `Git` 指令與參數：

- `git stash`
- `git stash -u`
- `git stash save`
- `git stash save -u`
- `git stash list`
- `git stash pop`
- `git stash apply`
- `git stash pop "stash@{id}"`
- `git stash apply "stash@{id}"`
- `git stash drop "stash@{id}"`
- `git stash clear`

參考連結

- [BASIC SNAPSHOTTING](#)
- [HOME](#)
- [回目錄](#)
- [前一天：認識 Git 物件的相對名稱](#)
- [下一天：Git for Windows 選項設定](#)