認識 Git

你需要版本控制（version control）。每個軟體專案都源自於一個想法，並在原始碼中實踐此想法。這些檔案就是讓應用程式能運作的魔法，所以得好好的照顧這些檔案的版本，要確保檔案安全無虞、保留歷史變更紀錄，或致謝（或甚至是抱怨）原作者。同時也要確保眾多團隊成員之間的合作能夠無縫接軌。

但也希望這些東西存在一個工具內，不會擋到我們的路，只有需要的時候才會啟動。

有這樣的魔幻工具嗎？既然你正在讀這本書，你可能已經猜到答案了，答案就是 Git！世界各地的軟體開發人員和單位都很喜歡 Git。到底 Git 為什麼會風靡全球呢？

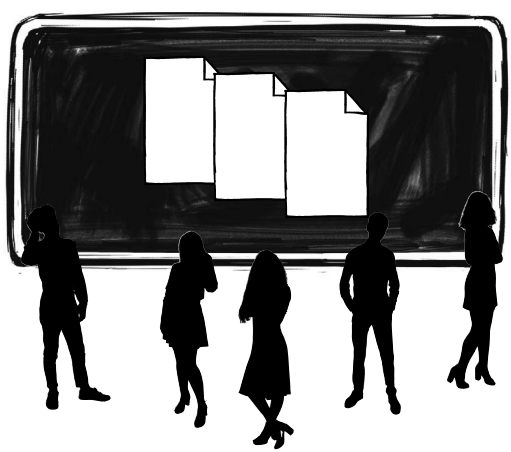
使用版本控制的原因

你可能有玩過需要玩很久才能打完的電玩。一步一步地完成遊戲進度，有輸有贏，獲得武器或裝備，有時候你可能同一關想要再挑戰一次，很多遊戲都可以儲存你目前的遊戲進度。如果你打敗火龍，下一步就是要一路過關斬將獲得巨大的祕寶。

確保一切順利，你打算先儲存遊戲進度再繼續冒險。儲存時會幫遊戲拍下一張「快照」記錄當前的進度。好處就是如果不小心遇到會噴出強酸的蜥蜴死了，就不需要再回頭走比較輕鬆的路線。只要重新載入之前的快照並再挑戰一次。火龍，別擋路！

版本控制能達到一樣的功能，可以儲存你目前的進度。稍微動一下手、儲存目前進度、繼續努力。這種「快照」功能可以記錄一系列的變更，即使專案中的好幾個檔案都有變動，全部都存在這一張快照內。

如果不小心弄錯了或是不喜歡目前的成果，都可以復原到之前快照記錄的情況，如果很喜歡目前的成果，只要再拍一張快照後繼續下去。



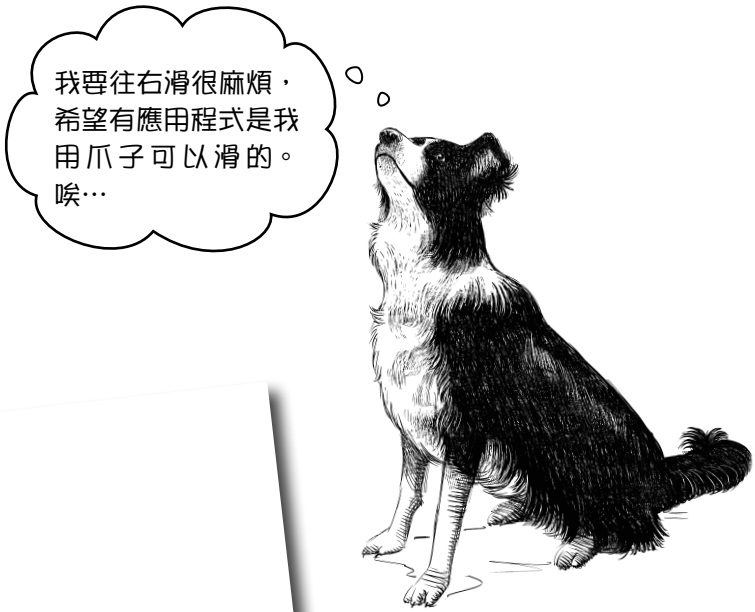
還不只這樣，Git 等版本控制系統可以讓你安心與其他開發人員一起同時處理同一批檔案，不會越俎代庖。後面的章節會再深入探討這功能，這裡講這些應該就夠了！

你可以把 Git 當成自己的記憶銀行、安全網路、合作平台，三者合一的系統。

認識了 Git 的版本控制，也就了解了這系統的強大之處和對我們工作的影響，能夠幫助我們變得更有效率。

恭喜！

你的公司剛剛拿到「熱狗交友」（Hawt Dawg）的合約，要幫人類最好的毛朋友打造史上第一個專屬他們的交友軟體，但在這個狗咬狗的世界，單身狗互相競爭，所剩時間不多了！



巴哥醫師診所
45021 美國俄亥俄州
狗窩山多佛街 100 號

工作說明書

恭喜貴公司獲選為本公司打造一款獨一無二的手機應用程式 ——「熱狗交友」。

此應用程式能提供人類最好的毛朋友拓展社交網路的機會、結交朋友，甚至能找到人生的伴侶！透過機器學習的最新技術、與專為狗狗需求所打造的直覺性介面，本公司希望能在短時間內成為產業界龍頭。

本公司相信現在是正確的時機，但我們也了解目前市場上的競爭，不過這次的想法是空前的嘗試，所以動作要快，也要準備好測試我們的想法。本公司會與貴公司的開發人員密切合作，持續迭代直到應用程式首次發布。

期待貴公司的初步設計與 alpha 版本的應用程式。

敬祝

安康

Johnny Grunt

Johnny Grunt 執行長

辦公室對話

瑪吉：我們想想看要不要用一個版本控制系統。

桑吉塔：我有聽過版本控制系統，不過一直沒機會實際用用看。不過時間不多了。

瑪吉：用 Git 很好入門。只要創建一個 Git 檔案庫（repository）就可以開始動手了。

桑吉塔：你說創一個什麼？

瑪吉：Git 檔案庫就是一個由 Git 所管理的資料夾。我先從頭開始說，你需要把這個專案的所有檔案都存在電腦的某個地方，對吧？

桑吉塔：我習慣把一個專案的所有相關檔案都放在同一個資料夾，包含來源檔案（source）、組建組態（build）、說明文件（documentation）。這樣比較好找。

瑪吉：這樣很好！一旦建好資料夾，就用 Git 在那個資料裡面設立一個檔案庫，就這麼簡單。

桑吉塔：這樣有什麼用？

瑪吉：就是你想要開始一個新的專案並用 Git 管理的話，執行一個 Git 指令叫 Git 把資料夾準備好，之後你就能在那個資料夾裡面使用其他的 Git 指令，可以想像成轉動插在車上的鑰匙來發動引擎，完成這第一步才能開車。

桑吉塔：嗯…了解。

瑪吉：只要一個指令，你的資料夾就已經「啟用 Git」，就像啟動車子的引擎，接下來就能順利地開始進行你的專案了！

桑吉塔：喔！我懂了。

瑪吉：需要幫忙再跟我說，我隨時都可以幫你一把。

懂 Git 了嗎？

如果你還沒安裝 Git 就沒辦法繼續下去了。如果還沒安裝 Git，現在趕快去吧！翻回去前面的介紹「你得安裝 Git」開始安裝吧！

就算 Git 安裝好了，最好記得是安裝最新版本的 Git，這樣才會跟本書的內容一致。

啓動引擎…

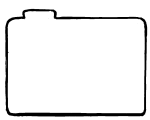
回想一下你以前進行過的專案，通常都會有超過一個以上的檔案 —— 可能是原始碼檔案、軟體文件、組建組態腳本等。如果我們想要用 Git 管理這些檔案，第一步就是建立一個 Git 檔案庫。

所以 Git 檔案庫到底是什麼？回想一下，使用版本控制系統其中一個原因，就是我們可以定期把我們的工作快照儲存起來。當然，Git 需要有個地方儲存這些快照，這個地方就是在 Git 檔案庫內。

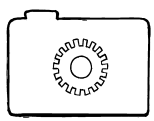
下一個問題就是 —— 這個檔案庫在哪裡呢？一般來說，我們會把一個專案中所有檔案都放到一個資料夾內。如果要用 Git 來當那個版本控制系統，我們首先要再那個資料夾裡面建立一個檔案庫，這樣一來 Git 就有地方可以儲存我們的快照。要建立一個 Git 檔案庫，需要在你的專案最上層資料夾執行 **`git init`** 指令。

針對這個我們稍後就會深入探討，但現在你只需要知道，沒有建立 Git 檔案庫，Git 能做到的功能真的很有限。

不論你的專案有多大（換言之，不論你的專案有多少檔案或次目錄），你的專案的**最上層**（根）資料夾需要執行 **`git init`** 才能開始使用 Git 進行你的專案。



專案資料夾



工作目錄

這樣包含 Git 檔案庫的資料夾的稱呼，聽起來比較厲害。

❶ 建立一個專案資料夾。

❷ 啟動 Git。

```
File  Edit  Window  Help
git init
```

在下一節我們會談到這部分。

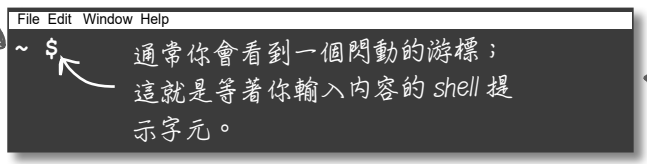
❸ 啟動資料夾內的 Git 檔案庫，會讓你的專案擁有超能力。你會聽到很多人都把這個稱為「工作目錄」。

命令列快速導覽：用 pwd 指令讓你知道自己身在何處

如果你對這不熟，可以翻回去緒論看一下。我們在「你得安裝 Git」附上一些說明供你參考。

在進行本書內的練習時，你會常常使用到命令列，所以我們先花一點時間熟悉命令列。一開始先像在緒論內提到的方式開啟一個終端機視窗，然後定位到你的硬碟裡面的一個位置。提醒一下，如果是 Mac 電腦，可以在**應用程式**（**Applications**）**> 工具程式**（**Utilities**）裡面找到**終端機 .app**（**Terminal.app**）。如果是 Windows 作業系統，按下開始按鈕，就能看到 Git 選單下面有 Git Bash。一開始就會看到提示字元，這告訴你終端機已經準備好可以接受指令。

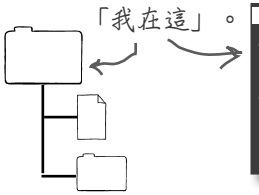
依據個人的終端機設定，畫面可能會不一樣。

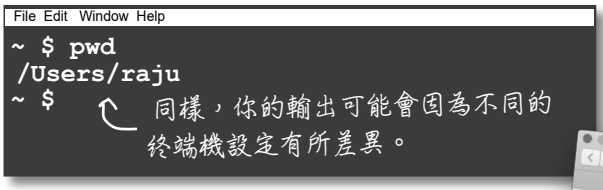


Windows 用戶：這就是 Git Bash 視窗。

我們先從簡單的開始。輸入 **pwd** 再按下確認鍵（Enter 鍵或 Return 鍵）；pwd 代表「列印工作目錄」，然後終端機目前正在執行的目錄路徑就會顯示。換言之，如果你想要建立一個新檔案或新目錄，就會出現在這個目錄中。

pwd 代表「我在這」。





有在視窗頂端看到這個路徑嗎？這等同於檔案總管（macOS 的 Finder）裡面的 pwd。

每一章最後有附上「削尖你的鉛筆」的答案。

Windows 用戶：當我們說「終端機」，指的就是…Git Bash！！

該動起來了！開啟終端機並練習使用 pwd 指令。把你看到的輸出內容寫在這：

太棒了！如果這是你第一次使用終端機或你對終端機其實不太熟，那可能會有點難，但是要記得：我們會在一路上陪伴你，不只是這個練習而已，本書所有的練習都會陪著你。

答案在第 44 頁。

指令列的其他功能：用 mkdir 創建新的目錄

這就是相當於 *mkdir* 的檔案總管畫面。

用 pwd 得知目前終端機目錄的位置很好用，因為幾乎你所有的行為都跟目前的目錄有關係，包含創建新資料夾。提到新資料夾，用來創建新資料夾的指令就是 mkdir，代表的就是「創建目錄」（make directory）。

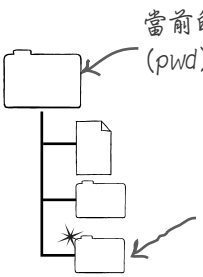
跟 pwd 不同的是，pwd 只會告訴你當前目錄的路徑，而 mkdir 則會接受一個引數（*argument*），這引數就是你想要創建的目錄名稱：

先別這樣做。等等我們會有給你練習的題目。

```
File  Edit  Window  Help
~ $ mkdir created-using-the-command-line
~ $
```

這就是引數，也就是新目錄的名稱。

如果一切順利的話，你就會看到另一個提示字元。



一定要記得把你的答案和我們在章末提供的解答比對一下。

如果你打算創建一個目錄但名稱已被使用，mkdir 指令會出現錯誤。

如果你試圖創建一個目錄，但新目錄名稱已被當前目錄中既有的目錄所使用，mkdir 指令會直接告訴你檔案已存在（File exists），不會執行任何動作。而且別被「檔案已存在」裡面的「檔案」二字搞混，這是指「資料夾」。

削尖你的鉛筆

要記得 *Windows* 裡面終端機就是 *Git Bash*。

換你練習了，在你開好的終端機視窗，直接使用 mkdir 來創建一個新目錄，命名為 my-first-commandline-directory。

把你使用的指令和引數寫在這裡。

接下來再次在同個目錄中執行該指令，把你看到的錯誤訊息寫在這：

錯誤就寫在這。

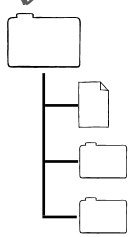
答案在第 44 頁。

指令列還有更多功能：用 ls 列出所有檔案

如果 mkdir 指令的輸出結果不是很理想，但只要沒有出現錯誤訊息，這指令就已經完成了它的功能。如果要確認是否真的如預期的一樣，你可以把當前目錄中所有的檔案列出來。列出所有檔案的指令就是 ls（list 的縮寫）。

你可以用 Mac 的 Finder 或 Windows 的檔案總管定位到目前的目錄，然後就可以使用這方式檢視。

在這執行 ls 指令代表列出當前目錄（pwd）中所有檔案與資料夾。



```
File Edit Window Help
~ $ ls
Applications    Movies
Desktop         Music
Documents       Pictures
Downloads       bin
Library         created-using-the-command-line
```

為求簡潔，我們截短了輸出的內容。

預設 ls 只會列出一般的檔案與資料夾。有時候你也會想要查看隱藏的檔案和資料夾（我們很快就會需要這個功能）。要達到這個功能，你可以在 ls 前面加上一個旗標（*flag*）。旗標和引數不一樣，旗標前面要加上一個連字號 -（hyphen）（藉此才能跟引數有所區隔）。要看到「所有」的檔案和資料夾（包含隱藏檔案和資料夾），我們可以使用「A」旗標（沒錯，就是大寫的「A」），例如：

這是「所有」的旗標。注意要有連字號。

這是指令。

注意大小寫。連字號後面接著的是一個大寫的「A」。

```
ls -A
```

```
File Edit Window Help
~ $ ls -A
.bash_history
.bash_profile
.bashrc
Applications
Desktop
Documents
Downloads
Library
```

名稱前面有「.」符號的就是隱藏的檔案和資料夾。

這就是輸出結果，你的看起來可能會有所不同。

跟前面一樣，為求簡潔，我們截短了輸出的內容。

削尖你的鉛筆

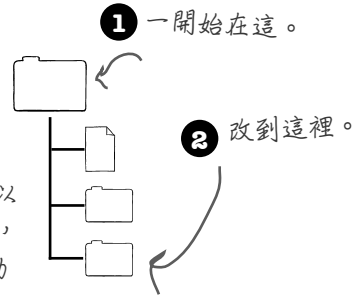
使用終端機把當前目錄中所有的檔案列出來。看看你是否能找到你最近剛創建的 my-first-commandline-directory。

然後使用 -A 的旗標來查看當前目錄中是否有隱藏的資料夾。

答案在第 45 頁。

指令列還有一些功能：用 cd 變更目錄

接下來繼續探索終端機。我們已經創建新的目錄，但我們要怎麼找到這個新目錄呢？為此有 cd 指令，cd 代表「變更目錄」（change directory）。一旦變更目錄後，我們就可以使用 pwd 來確認我們的確有改變目前的位置了。



大多的終端機都有自動完成功能，所以你可以只輸入目錄名稱的前幾個字母，然後點擊「Tab」鍵，終端機就會自動幫你把剩下的部分打完。

我們絕對不能把名字弄錯了。

```
File Edit Window Help
~ $ cd created-using-the-command-line
~/using-the-command-line $ pwd
~/using-the-command-line $ /Users/raju/created-using-the-command-line
```

根據終端機不同，這裡可能會顯示該目錄的完整路徑或簡短版本。不論是哪種方式，你隨時都可以用 pwd 來確認目前的位置。

cd 指令可以定位到當前目錄之下的次目錄。我們也可以使用 cd 回到上層目錄，例如：

cd 和兩點（..）之間需有一空格。

```
cd ..
```

```
File Edit Window Help
~/using-the-command-line $ cd ..
~ $
```

兩點代表「上層目錄」。

這就是那兩點。

一定要隨時（用 pwd）注意自己的工作目錄——指令列的多數操作都和這個目錄有關。

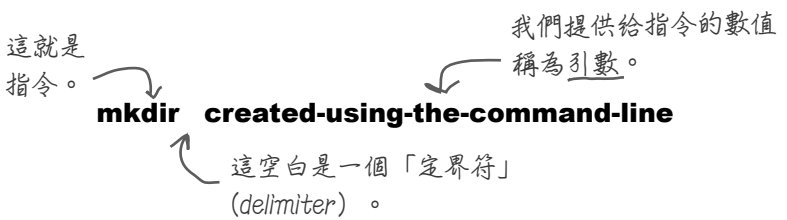
習題

直接試試看變更目錄。使用 cd 跳到你新創建的 my-first-commandline-directory 資料夾，然後使用 pwd 來確認你的確變更了目錄，然後再使用 cd .. 回到上層目錄。把下面的空白當成筆記本來練習使用這些指令。

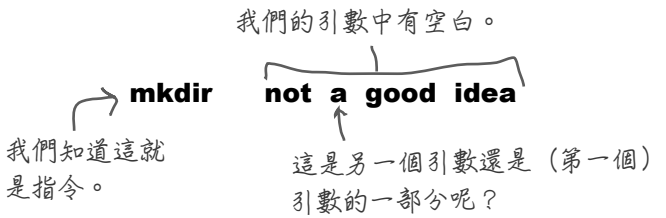
答案在第 45 頁。

這沒有引數

pwd 和 mkdir 等指令是我們正在使用的「指令」。mkdir、cd 等指令需要你告訴這些指令你想要創建什麼、你想要去哪，我們是透過使用「引數」（argument）來補充這些指令。



你可能會好奇為什麼我們會選擇用連字號，而不是使用空白。因為在引數中使用空白會讓情況更麻煩。你看看，指令列已經用空白把指令和引數區別開來。所以，如果你的引數中也有空白，指令列就會變得很容易搞混。



在指令列之中，空白的功能是一個分隔符號。但如果我們的引數中有空白，對於指令列來說要分辨你到底是輸入多個引數，還是一個有大於一個字的引數。

所以當你的引數中有空白而且你又想把它視為單一引數，你就得用引號。

```
mkdir    "this is how it is done"
```

現在這樣就很清楚是一個引數。

如你所見，你如果在引數中使用空白很容易會出問題。我們建議是什麼呢？檔名和路徑避免使用空白。

例如，C:\my-projects\ 相較於 C:\my projects\ 是比較好的路徑。

好問題。指令列其實不在乎你用雙引號還是單引號，但記得要一致。如果引數一開始是用單引號，最後也用單引號。雙引號也一樣。

一般來說，大多終端機的使用者都偏好使用雙引號，我們也是，但有一個情況你就一定要用雙引號，就是當你的引數裡面有單引號的時候。

請注意這個範例，`sangita's` 這個字裡面有用到單引號：

```
mkdir "sangita's_home-folder"
```

這裡用了單引號，引數前後就得用雙引號。

情況反過來也一樣，如果你的引數中得用一個雙引號，那引數的前後就得用單引號。

然而我們其實有說過了，最好的方式就是在引數中不要用空白，尤其是目錄和檔案名稱裡面。**你需要空白的時候，就用連字號或底線（_）**。當引數後需要補充資訊時，就能避免用到任何的引號。

連連看？

透過指令列，有各式各樣的指令和旗標。在這個指令配對遊戲中，請將每個指令配對到對應的描述。

cd	顯示當前目錄的路徑。
pwd	創建一個新的目錄。
ls	回到上層目錄。
mkdir	變更目錄。
ls -A	列出當前目錄中的一般檔案。
cd ..	列出當前目錄中的**所有**檔案。

答案在第 46 頁。

大掃除

目前你已完成這一節，我們建議你要把練習時創建的資料夾清理一下，如 `my-first-commandline-directory` 等。只要用檔案總管（或 Finder）刪除即可。雖然指令列也有可以達成此功能的方式，但用指令列刪除檔案通常會跳過垃圾桶。換言之，如果不小心刪錯資料夾就很難復原。

之後等你更熟悉指令列的時候，或許你就可以用適當的指令來刪除檔案，但就目前而言，我們選安全一點的方式。

創建你的第一個檔案庫

我們來花點時間多熟悉 Git 吧。你已經安裝好 Git，所以接下來就可以確認一切準備就緒，並了解要怎麼創建 Git 檔案庫。為了創建檔案庫，你會用到終端機視窗。沒錯！

一開始先打開終端機視窗，就跟之前的練習一樣。為了讓一切都方便管理，我們建議你創建一個名為 **headfirst-git-samples** 的資料夾，把本書中**所有**範例都放在裡面。在那個資料夾裡面，直接創建一個資料夾放第 1 章的所有練習，命名為 **ch01_01**。

如果你對指令列不太熟悉，你可以用 Windows 作業系統的檔案總管或 Mac 的 Finder 來創建一個資料夾。但我們會常常用到指令列，所以你對指令列會越來越熟悉。

```
File Edit Window Help
headfirst-git-samples $ mkdir ch01_01
headfirst-git-samples $ cd ch01_01
ch01_01 $
```

一開始先創建名為 ch01_01 的目錄。

然後變更成這一個。

cd 代表的是「變更目錄」。

別忘了 mkdir 指的是「創建目錄」

既然已經位於一個全新的目錄中，我們來創建我們的第一個 Git 檔案庫。為了達成此功能，只要在我們新創建的資料夾裡面執行 `git init` 指令。

執行 init 指令。

要確認大小寫正確。Git 指令一定是小寫。

```
File Edit Window Help
ch01_01 $ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:  git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:  git branch -m <name>
Initialized empty Git repository in ~/headfirst-git-samples/ch01_01/.git/
ch01_01 $
```

先忽略這邊的提示（hint），下一章會探討到。

Git 表示一切順利。

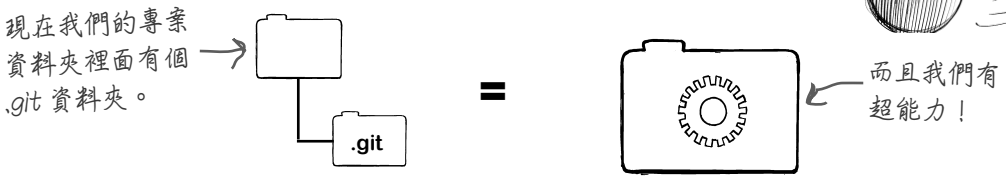
滿簡單的，對吧？你已經創建好了 —— 你的第一個檔案庫。

認識 init 指令

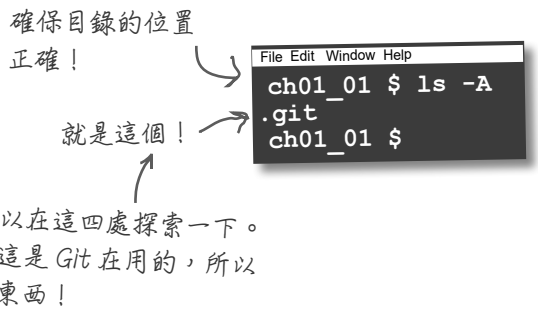
所以我們剛剛到底完成了什麼？ `git init` 指令看起來功能很簡單，但其實功能很多。讓我們回到一開始來看看這指令到底做了什麼。

首先我們先從全新、空白的目錄開始。

我們使用終端機定位到資料夾的位置，然後使用魔法咒語 `git init`，`init` 是**啟動（initialize）**的縮寫。Git 知道我們叫它在這個位置創建一個檔案庫，然後它會創建一個隱藏資料夾，名為 `.git`，裡面會有一些設定檔和一個子資料夾，這資料夾裡面會存放我們請 Git 拍快照時的檔案。



其中一個確認的方式就是用我們的終端機把所有檔案列出來，例如：



這個隱藏資料夾代表 Git 檔案庫，它的功用就是儲存任何與你的專案有關的檔案，包含所有提交檔案、專案歷史紀錄、設定檔等。也會儲存任何擬針對此特定專案所設定的特定 Git 配置與設定。

沒有蠢問題

問：我用電腦時比較習慣用我自己的檔案總管，我可以用自己的檔案總管看到 .git 資料夾嗎？

答：當然！大多作業系統預設的檔案總管不會顯示隱藏檔案和資料夾。要看一下你的偏好設定確定自己可以看到隱藏檔案和資料夾。

問：如果我不小心刪掉這個目錄怎麼辦？

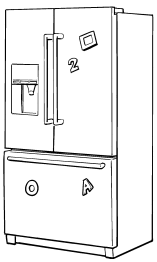
答：首先，最好不要發生。再者，這個資料夾就是個「保險庫」，裡面 Git 儲存所有的資訊——包含完整專案歷史紀錄和其他 Git 打理需要的其他檔案，還有一些我們可以用來擁有客製化 Git 體驗的設定檔。這代表如果你刪掉這檔案，你會弄丟所有的專案歷史紀錄，但專案資料夾裡面的其他檔案不會受到影響。

問：如果我在同個資料夾不小心執行了 git init 指令一次以上怎麼辦？

答：好問題。這完全沒問題。Git 只會告訴你它在重新啟動 Git 檔案庫，但你不會遺失任何資料，也不會造成任何傷害。事實上，你可以在 ch01_01 裡面試試看。我們才剛開始，而且學習的最佳方法就是去嘗試。有什麼好失去的呢？

問：我用過的其他版本控制系統有伺服器，我們在這不需要嗎？

答：Git 一開始真的超簡單。git init 會創建一個 Git 檔案庫，之後就可以開始工作。最後你會需要一個機制來和團隊成員分享工作進度，而且我們保證很快就會學到。就目前而言，你一切都準備好了。



程式碼重組磁貼

我們列出要創建新資料夾、變更、及創建新 Git 檔案庫的所有步驟。我們身為勤勞的軟體開發人員，要時常確保自己是在正確的目錄中。為了要幫助同事，我們把程式碼都列出來用磁貼貼在冰箱上，但它們全都掉到地上了。你的工作就是要把這些磁貼放回去。注意有些磁貼可以使用一次以上。

在這裡重新整理磁貼。

答案在第 46 頁。

向 Git 自我介紹

我們要開始用 Git 和 Git 檔案庫之前還有一個步驟。Git 希望你向它自我介紹一下。這樣當你拍下一張「快照」，Git 就知道是誰拍的。而且我們等等就會開始教你怎麼拍快照，所以我們先把這個處理好。這個步驟只要做一次，這個設定適用於你在電腦上做的所有專案。

我們會先從熟悉的好朋友開始——終端機，之後再繼續下去。**記得用你自己的名字和電子郵件，別用我們的！**（我們知道你很愛我們，但是我們不希望搶走你的功勞！）一開始先開啟一個新的終端機視窗，不用擔心要變更目錄的事情——這部分的設定在哪裡執行都沒關係。

啟動你的終端機再跟著我們的步驟做。

❶ 一開始先告訴 Git 我們的全名。

```
File  Edit  Window  Help
~ $ git config --global user.name "Raju Gandhi"
```

你可以在任何目錄中執行這個功能。

執行 config 指令。

❷ 然後告訴 Git 我們的電子郵件。這裡可以使用你的個人電子郵件，之後隨時都可以改。

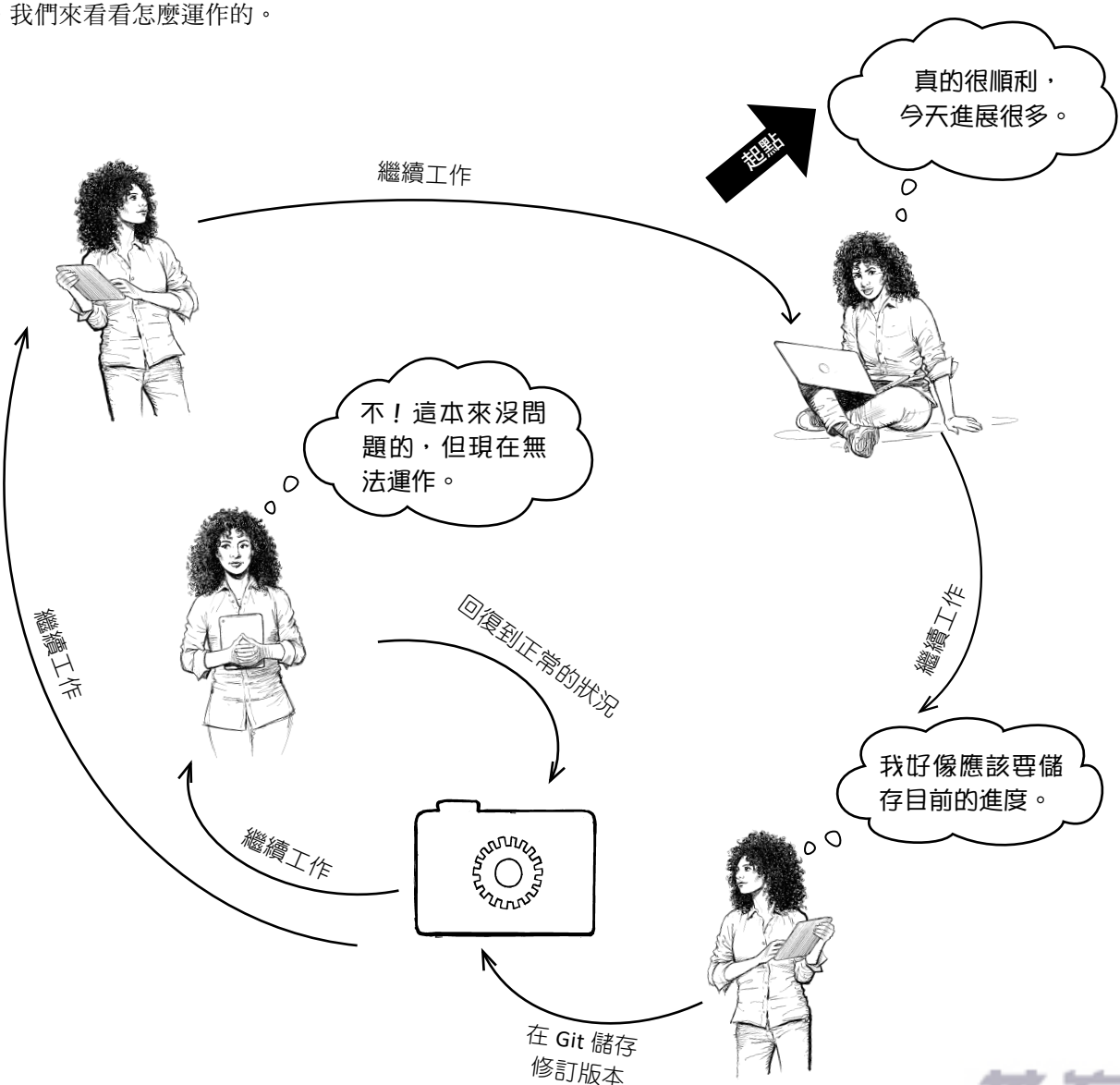
```
File  Edit  Window  Help
~ $ git config --global user.email "me@i-love-git.com"
```

鍵入你的電子郵件地址。

備註：之後隨時都可以修改這些資訊，只要執行同一個指令，輸入不同的數據即可。所以如果一旦你決定要用公務電子郵件，那就沒問題了，直接用就可以。你或許可以考慮把這頁用書籤夾起來。

怎麼用 Git

我們稍微了解一下一般會怎麼使用 Git。還記得我們有提過可以儲存進度的電動遊戲嗎？嗯，要請 Git「儲存你的進度」需要把自己的作品「提交」給 Git。基本上，這代表 Git 把你作品的修訂版本儲存起來。一旦你提交之後，就可以開心地繼續做下去，直到你覺得有必要再儲存另一個修訂版本，然後這個循環持續下去。我們來看看怎麼運作的。



把 Git 用在實際情境中

相信你一定迫不及待要開始了（我們就是知道！）。目前，已經啟動 Git 檔案庫、告訴 Git 我們的名字和電子郵件，也大概知道通常要怎麼使用 Git。如果把 Git 用在實際情境中呢？我們從簡單的開始，讓你們看看 Git 能做什麼 —— 我們先看 Git 如何透過創建「提交」來「拍快照」。

為了這個練習，我們假裝是一個全新的專案，通常會先完成一個檢核表，這樣就能確保該做的都有做。隨著專案進度持續邁進，我們還是要一直檢查各個項目（要持續激發多巴胺！），隨著對這個專案的了解越深，我們會持續增加項目。最後這檔案自然就會有版本控制，管控著專案中其他的所有檔案，我們就是透過 Git 達成這功能。

我們一步一步來看看該做什麼。

第一步：

創建一個全新的專案資料夾。

第二步：

在那個資料夾裡面啟動 Git 檔案庫。

你應該會對這兩個步驟頗熟悉。

第三步：

建立檢核表，包含一些一開始要檢核的項目。

第四步：

在 Git 裡面透過提交此檔案來儲存檢核表的快照。

這就是我們要看到的結果！

但我們先回頭處理熱狗交友服務

哈囉，歡迎來到這裡。我們真的得開始動手做熱狗交友 APP 了。現在有很多狗狗都在尋覓真愛。我建議一開始先把所有該做的事情都列入檢核表，這樣才不會忘了做什麼事。

熱狗 APP 專案經理。

因為我們才剛開始學 Git，你何不開一個終端機視窗跟著一起做呢？

首先要在 **headfirst-git-samples** 的上層資料夾內建立一個新的資料夾，要用 **pwd** 指令確保自己在正確的目錄中。如果你的終端機還在 `cd01_01` 目錄，你可以使用 **cd ..**（記得要有兩個點）回到上一層。

建立我們的熱狗目錄。

```
File  Edit  Window  Help
headfirst-git-samples $ mkdir HawtDawg
headfirst-git-samples $ cd HawtDawg
HawtDawg $
```

別忘記換到這裡！

對，我們知道這是重複的。但是這樣可以讓我們多用這些指令，就能加強印象。畢竟這本是「深入淺出 Git」。

接下來只要用我們很熟悉的 **git init** 在 **HawtDawg** 裡啟動一個新的檔案庫。

啟動這個 Git 檔案庫。

```
File  Edit  Window  Help
HawtDawg $ git init
Initialized empty Git repository in ~/headfirst-git-samples/HawtDawg/.git/
HawtDawg $
```

Git 親切地告知我們，它完成了我們的指令。

我們沒有把 git init 提供的提示顯示出來。你會看到提示，我們下一章會談到這個。