系統安裝記錄 Documentation

Release 1.0

陳國華

October 22, 2014

CONTENTS

Contents:

CHAPTER

ONE

SPHINX-DOC 安裝

目前對於一般使用者或系統開發者在記錄自己的學習心得的方式會隨著不同環境及狀況而所使用的工具有所不同，當在網路尚未普及到家家户户都接ADSL時，可能大都是寫在記事本、Microsft Office系列的文書軟體，但隨著網路的普及化加上全球資訊網出現，陸續出現BLOG、WIKI，到現在的雲端服務，如evernote、google雲端文件、Microsft Office系列等…，不論在桌上型電腦、筆記型電腦、平板、手機，在不同的作業系統都可以存取，但這些都注重於快速記事，如果要把所記錄的東西整合後出版到網路上公開，發佈成網頁、pdf、epub等格式，在出版這塊或許office是一般使用者最常用的，也可以發佈成PDF，而EPUB可以透過其他的軟體轉換，如果是發佈成網頁或許少數幾頁的文件是沒問題，大不了網頁內容長了點，但如果是一本書或使用手冊呢？將數十頁或數百頁的內容發佈成一個網頁，這就不太好觀看及使用了，這樣還需要花費更多的人力去建置網頁，檔案格式又不同了，當這個時後書本、使用手冊要改版時將會要修改多個版本的內容，再加上章節內容抽換時會出現文件格式變動的問題等…。上述的問題，在以建立書本、使用手冊爲主，所要的需求如下：

- 編輯一份文件，即可同時發佈成網頁、PDF、EPUB等格式
- 在文件改版，可容易抽換章節內容
- 可在不同的文件編輯器上寫作
- 便於多人共筆寫作

綜合以上的需求，SPHINX-DOC可以幫我們完成這個任務 SPHINX-DOC有別於其他的文書出版軟體，SPHINX-DOC起初是爲了創建新的 PYTHON使用者手冊 而產生的，文件的內容是使用reStructuredText(reST)標記語言所編寫，reST是種跨平台及任何文字編號器都可以編輯，而且是個易讀的純文字標記語法，主要是透過Python中的Docutils元件將純文字轉換成不同的格式。

1.1 測試環境

- 作業系統：Ubuntu 14.04
- Python：3.4.0
- SPHINX-DOC：1.2.2
- Apache：2.4.7

1.2 安裝SPHINX-DOC

本系統使用python3的版本做架設，在ubuntu有sphinx-doc套件可以直接安裝，而且是最新的版本1.2.2的版本，套件有分Python2和Python3的版本，兩種版本所產生出來的內容經比對過後，主要差異在因爲中文使用Unicode，使用Python2的版本所產生出來的內容，如果有中文字，在前方會加一個''u''。

Python2和Python3中文差異

python2：project = u'中文專案'
python3：project = '中文專案'

安裝sphinx使用python3

```
sudo apt-get install python3-sphinx
```

安裝sphinx使用python2

```
sudo apt-get install python-sphinx
```

安裝執行畫面

1.3 建立SPHINX-DOC專案

建立專案執行畫面

1.4 匯出HTML網站

在Ubuntu下，不論是Python2或是Python3都是用「 make html 」指令建立HTML網站

匯出HTML指令

```
make html
```

執行畫面

```
allen@uServer:~/git/allen$ make html
sphinx-build -b html -d _build/doctrees   . _build/html
Making output directory...
Running Sphinx v1.2.2
loading pickled environment... done
building [html]: targets for 6 source files that are out of date
updating environment: 0 added, 0 changed, 0 removed
looking for now-outdated files... none found
preparing documents... done
writing output... [100%] index
writing additional files... genindex search
copying static files... done
copying extra files... done
dumping search index... done
dumping object inventory... done
build succeeded.

Build finished. The HTML pages are in _build/html.
```

1.5 匯出PDF

在全英文的內容下直接執行匯出PDF是沒有問題，但內容出現有中文的時後匯出就要做另外設定及中文字型的安裝與設定，如果Ubuntu是安裝中文版，文鼎字型會自動安裝好，英文版則需要自行安裝中文字型，安裝方式如下

安裝文鼎標楷體與細明體

```
sudo apt-get install fonts-arphic-ukai fonts-arphic-uming
```

- fonts-arphic-ukai - 「文鼎 PL UKai」 (AR PL UKai) 中文萬國碼 TrueType 楷體字型
- fonts-arphic-uming - 「文鼎 PL UMing」 (AR PL UMing) 中文萬國碼 TrueType 明體字型

檢查系統中已安裝的中文字型

```
fc-list :lang=zh
```

執行畫面

```
allen@uServer:~$ fc-list :lang=zh
/usr/share/fonts/truetype/arphic/uming.ttc: AR PL UMing TW MBE:style=Light
/usr/share/fonts/truetype/arphic/ukai.ttc: AR PL UKai CN:style=Book
/usr/share/fonts/truetype/arphic/ukai.ttc: AR PL UKai HK:style=Book
/usr/share/fonts/truetype/arphic/ukai.ttc: AR PL UKai TW:style=Book
/usr/share/fonts/truetype/arphic/ukai.ttc: AR PL UKai TW MBE:style=Book
/usr/share/fonts/truetype/arphic/uming.ttc: AR PL UMing TW:style=Light
/usr/share/fonts/truetype/arphic/uming.ttc: AR PL UMing CN:style=Light
/usr/share/fonts/truetype/arphic/uming.ttc: AR PL UMing HK:style=Light
```

安裝匯出pdf所需要的套件

```
sudo apt-get install texlive-xetex texlive-latex-recommended texlive-latex-extra texlive-fonts-recommended
```

設定conf.py ：爲了要在pdf中顯示中文，在檔案中找到latex_elements，加入下面的文字，設定中文字型。

```
'preamble': r'''
\usepackage{fontspec}     % 引入 fontsepc，這樣才可以用下面的設定字型的指令
\setmainfont{AR PL UMing TW}  % 預設字型
\setsansfont{AR PL UMing TW}  % sans family 字型
\setromanfont{AR PL UMing TW} % roman 字型
\setmonofont{AR PL UMing TW}  % 等寬字型

\XeTeXlinebreaklocale "zh"          % 設定斷行演算法爲中文
\XeTeXlinebreakskip = 0pt plus 1pt  % 設定中文字距與英文字距
''',
```

如果是在pytoh2的版本，在conf.py檔案最前面加入下面的文字以正常顯示中文

```
import sys, os
# Avoid unicode problem for python2
if sys.version_info.major == 2:
    reload(sys)
    sys.setdefaultencoding('utf8')
```

如果沒有設定，會出現下面的錯誤訊息

```
Encoding error:
'ascii' codec can't decode byte 0xe5 in position 385: ordinal not in range(128)
The full traceback has been saved in /tmp/sphinx-err-eDbiuU.log, if you want to report the issue to the developers.
```

1.6 匯出ePub

在Ubuntu下，不論是Python2或是Python3都是用「 make epub 」指令建立ePub格式的電子書

匯出HTML指令

```
make epub
```

執行畫面

```
allen@uServer:~/git/allen$ make epub
sphinx-build -b epub -d _build/doctrees   . _build/epub
Making output directory...
Running Sphinx v1.2.2
loading pickled environment... done
building [epub]: targets for 6 source files that are out of date
updating environment: 0 added, 0 changed, 0 removed
looking for now-outdated files... none found
preparing documents... done
writing output... [100%] index
writing additional files... genindex search
copying static files... done
copying extra files... done
writing mimetype file...
writing META-INF/container.xml file...
writing content.opf file...
WARNING: unknown mimetype for _static/test, ignoring
writing toc.ncx file...
writing sphinx.epub file...
build succeeded, 1 warning.

Build finished. The epub file is in _build/epub.
```

1.7 參考網站連結

- http://docutils.sourceforge.net/rst.html
- http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html

CHAPTER

TWO

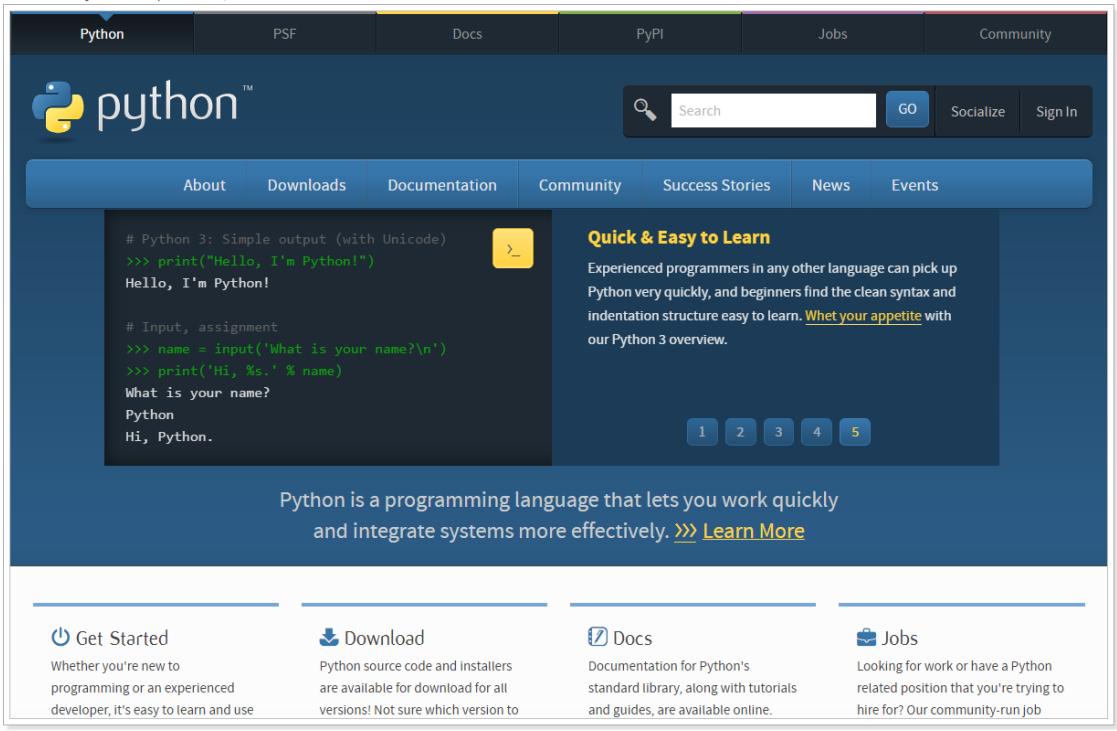
SPHINX-DOC 安裝（WINDOWS）

2.1 測試環境

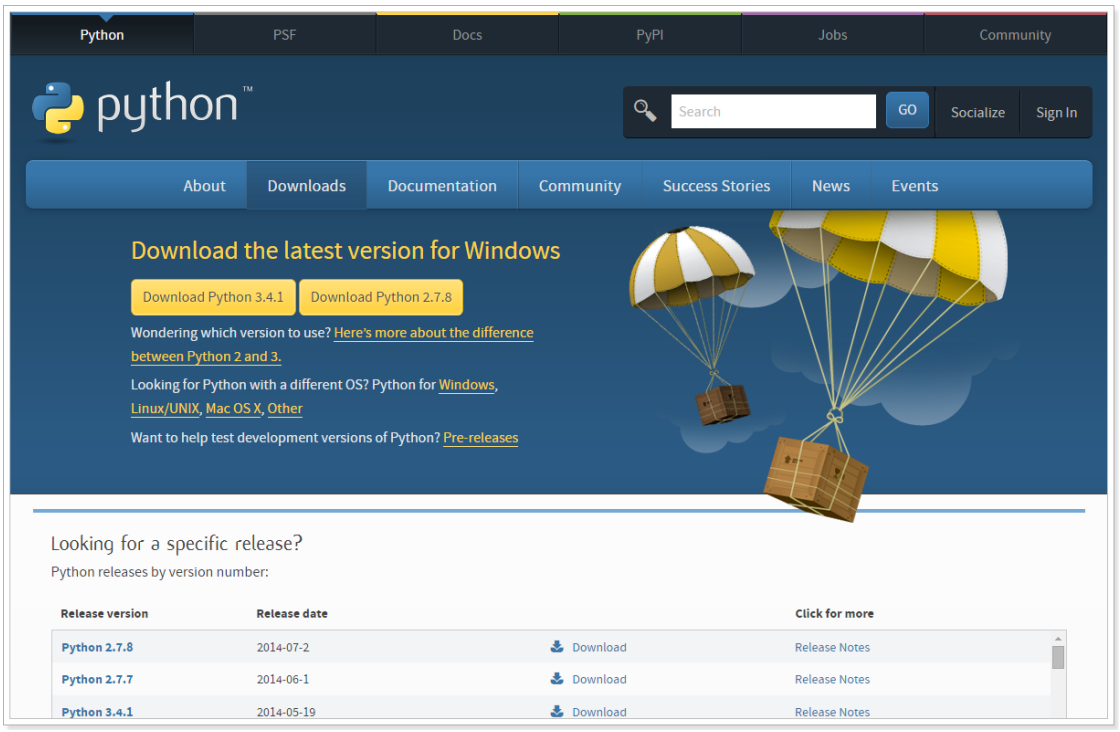
- 作業系統：Windows 7(64位元)
- Python：3.4.1
- SPHINX-DOC：1.2.3

2.2 下載 Python

- Python官方網站

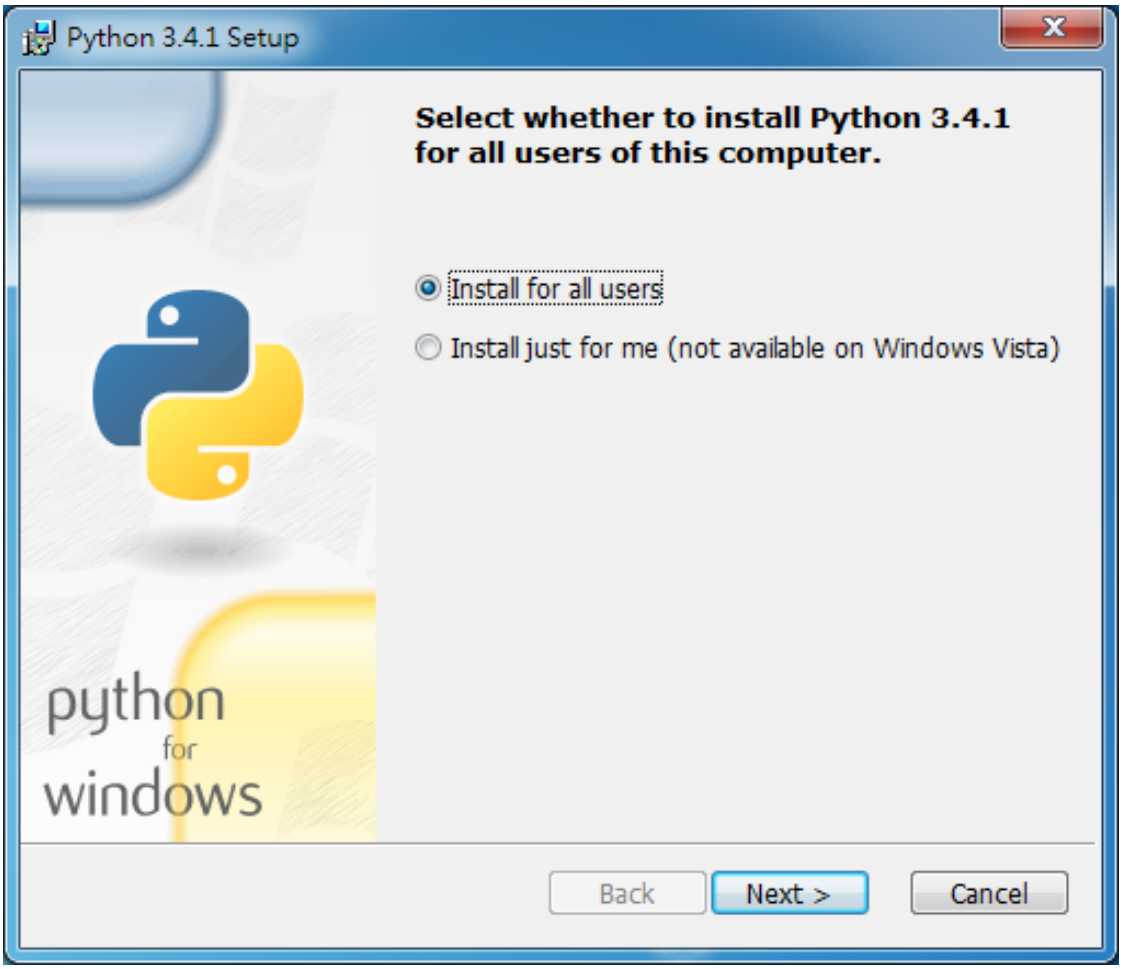


- Python Download ：進入下載頁面後，選擇『Download Python 3.x.x』下載Python3的版本

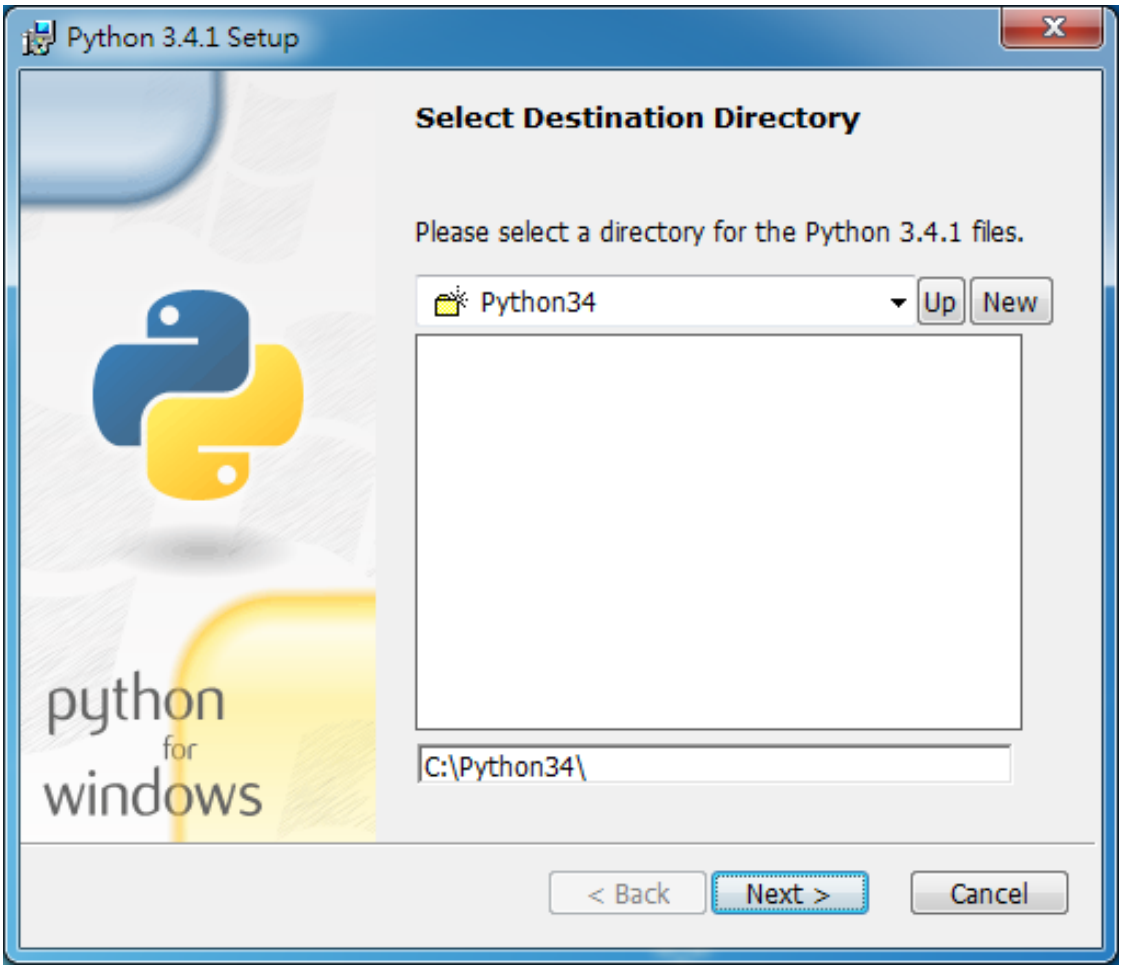


2.3 安裝 Python

- 執行Python安裝程式『python-3.4.1.msi』，使用預設值『Install for all users』



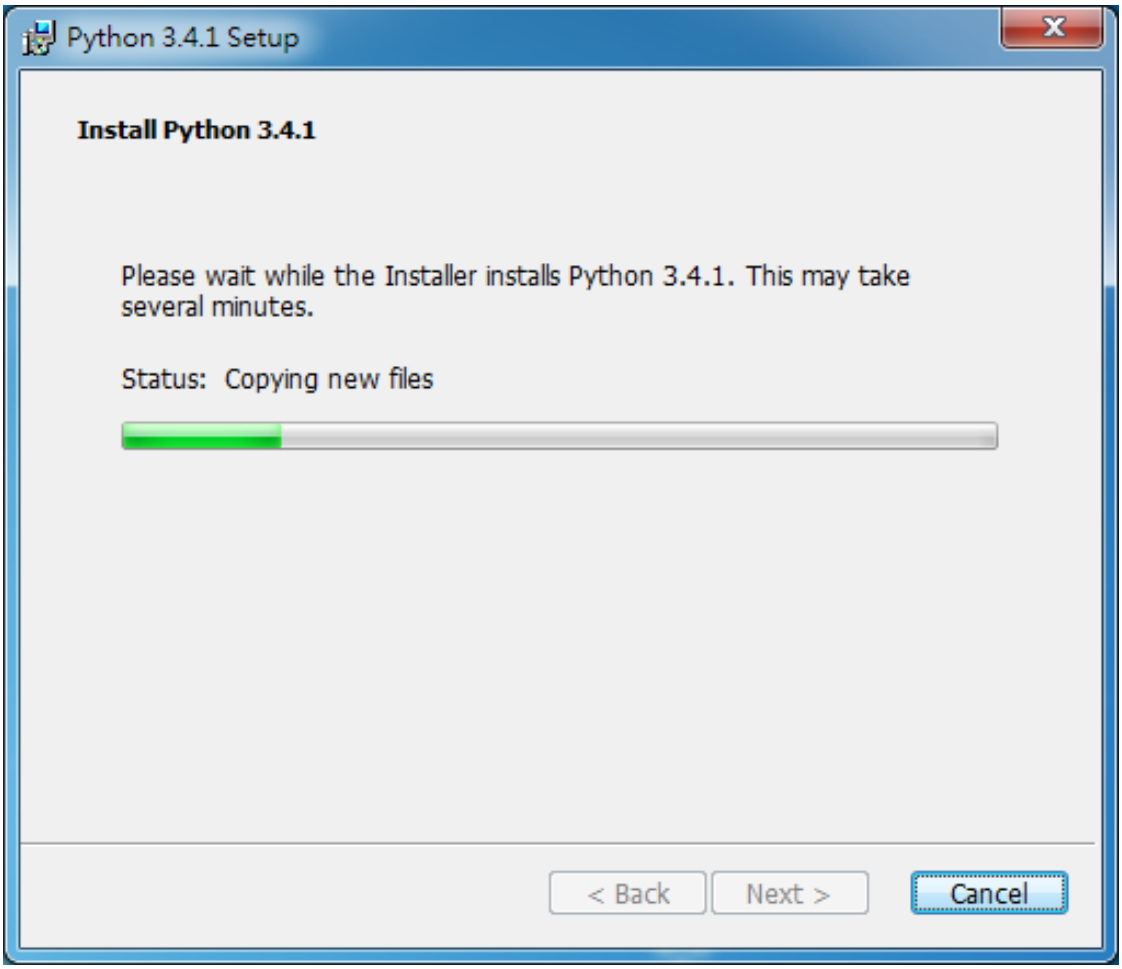
- 選擇安裝目錄，使用預設值『C:\Python34\』



- 客製化安裝，這地方移到最後一個項目『Add python.exe to Path』，這預設是沒有選的，因爲是第一次安裝，所以就把這個選項一起安裝，主要是將「C:\Python34\;C:\Python34\Scripts;」這一串文字自動加入到環境變數的Path中，如果沒有選取安裝，需自行手動加入。

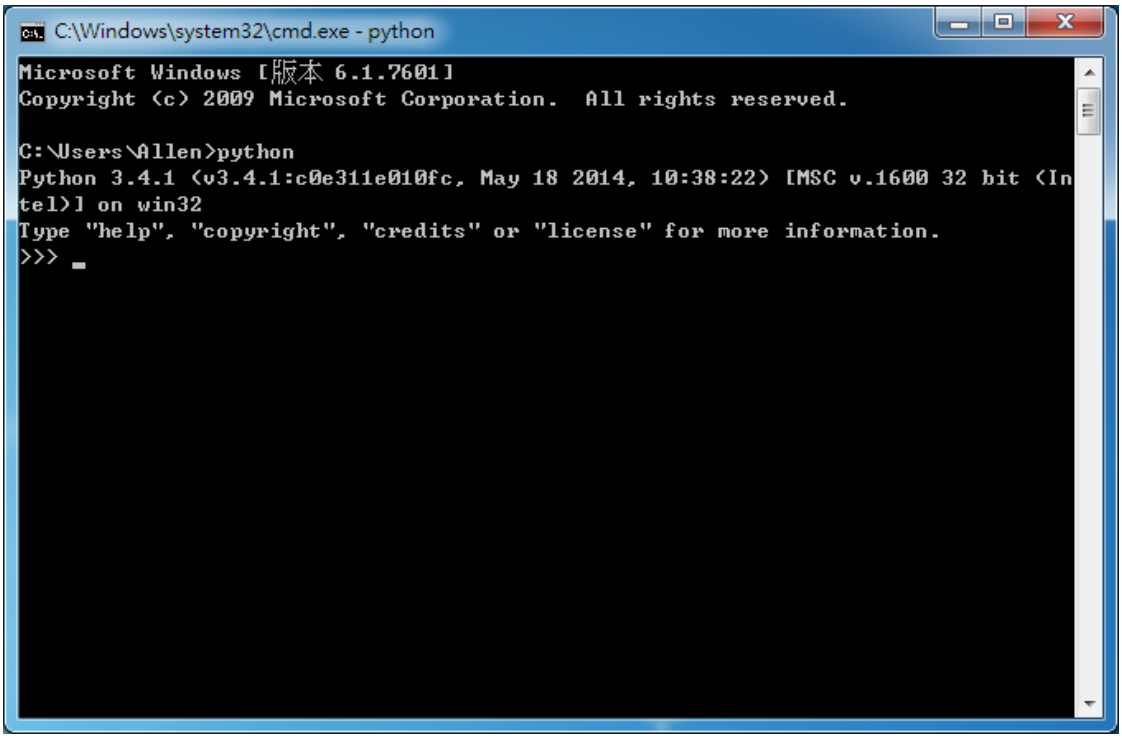

Python 3.4.1 Setup
Customize Python 3.4.1
Select the way you want features to be installed.
Click on the icons in the tree below to change the
way features will be installed.
Register Extensions
Tcl/Tk
Documentation
Utility Scripts
pip
Test suite
Add python.exe to Path
Prepend C:\Python34\ to the system Path
variable. This allows you to type 'python' into a
command prompt without needing the full path.
This feature requires 0KB on your hard drive.
python
for
windows
Disk Usage
Advanced
< Back
Next >
Cancel

- 軟體安裝進度


Python 3.4.1 Setup
Install Python 3.4.1
Please wait while the Installer installs Python 3.4.1. This may take several minutes.
Status: Copying new files
< Back
Next >
Cancel

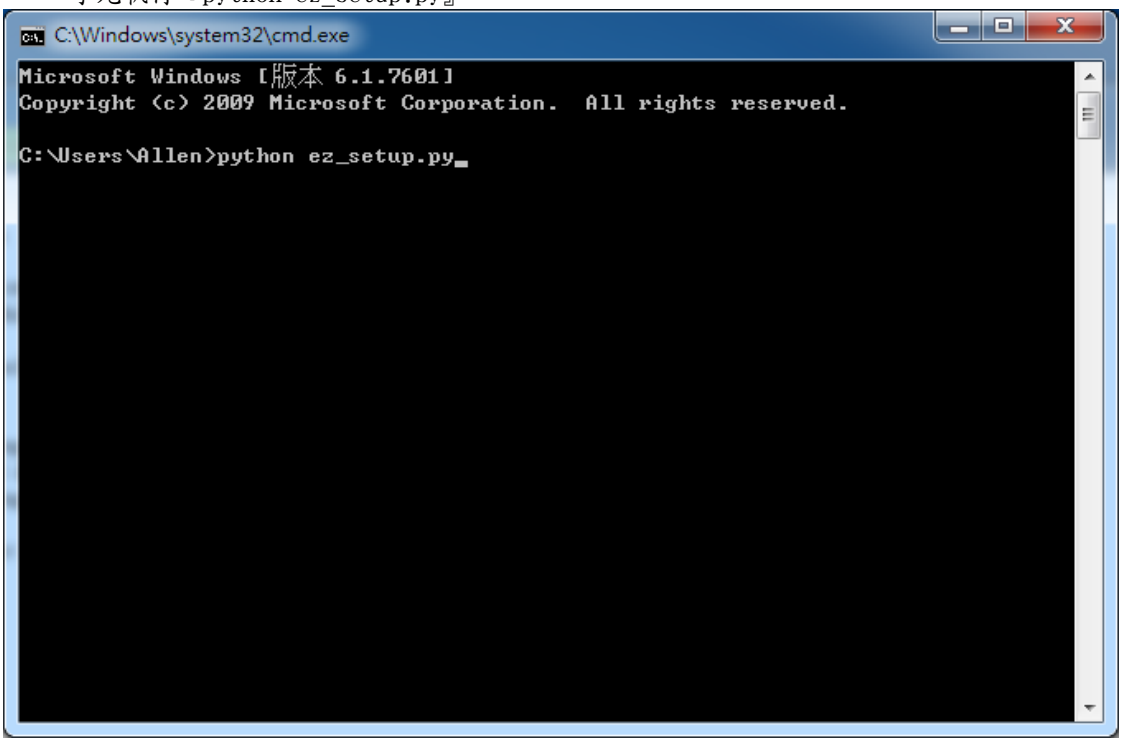
- 安裝完成

- 執行命令提示字元「開始->所有程式->附屬應用程式->命令提示字元」，然後直接輸入『python』，就會看到如下的畫面，如果沒有請先確認系統環境變數的path是否有設定python的資料夾路徑「C:\Python34\;C:\Python34\Scripts;」

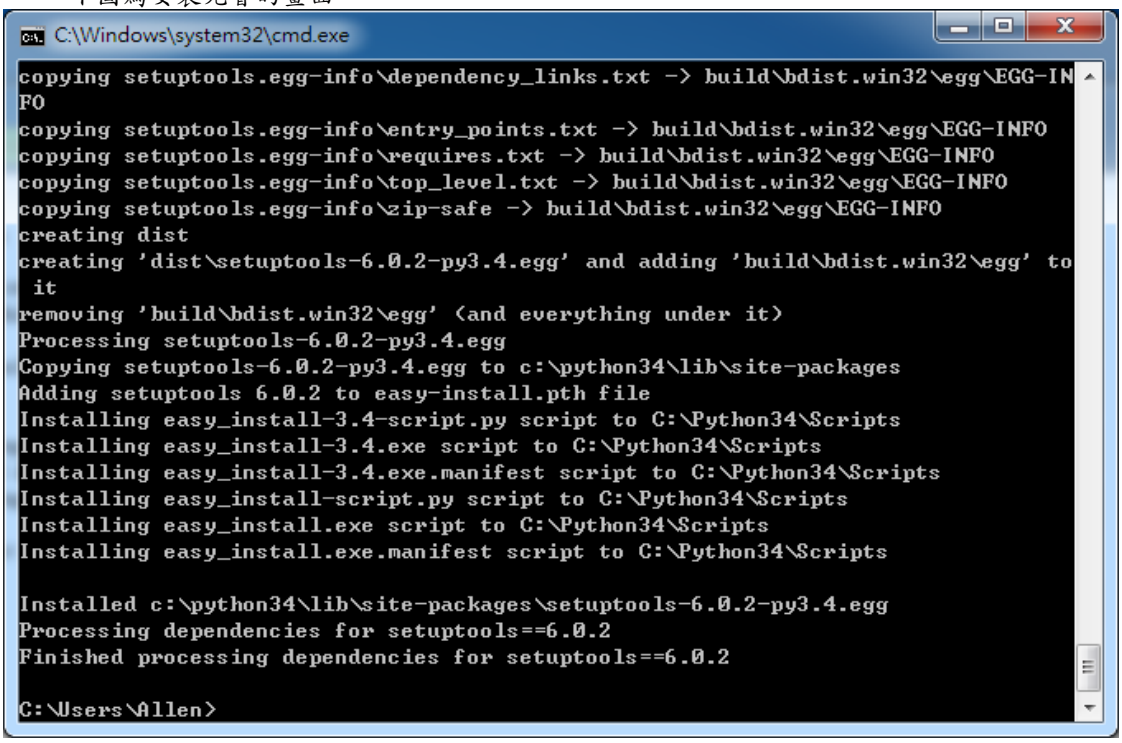


2.4 安裝easy_install

- 下載 ez_setup.py ，如下圖所下載的「ez_setup.py」存放在「C:\Users\Allen\」下並使用命令提示字元執行『python ez_setup.py』

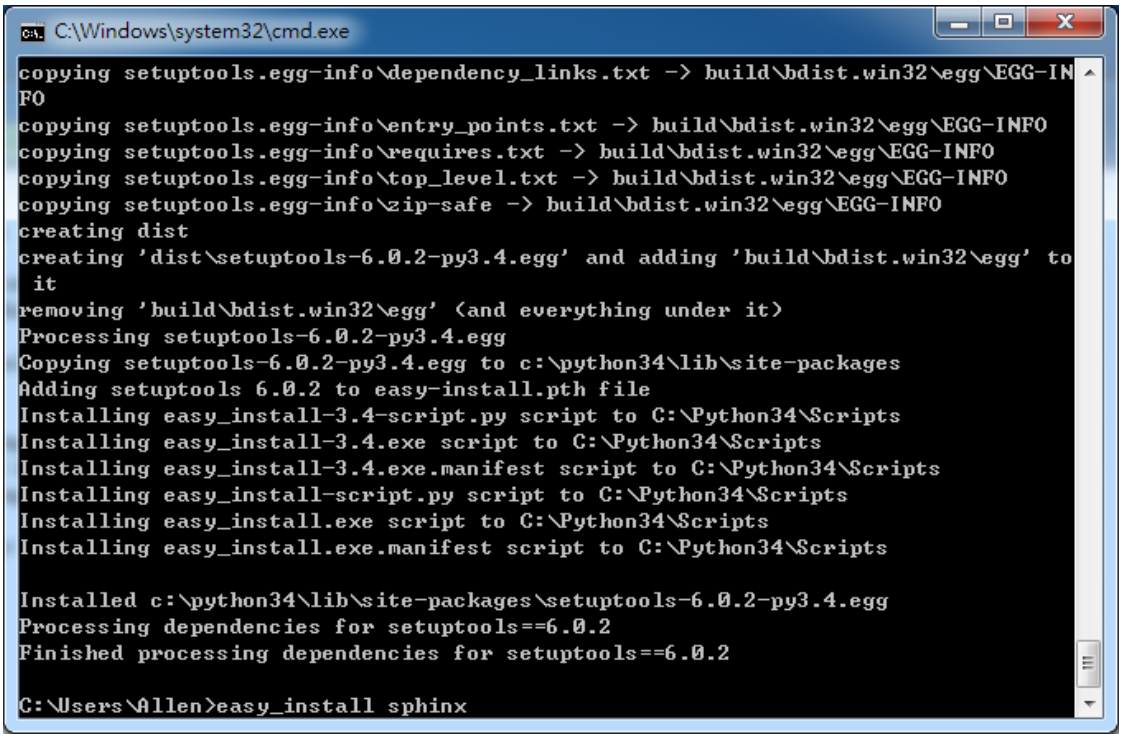


- 下圖爲安裝完會的畫面

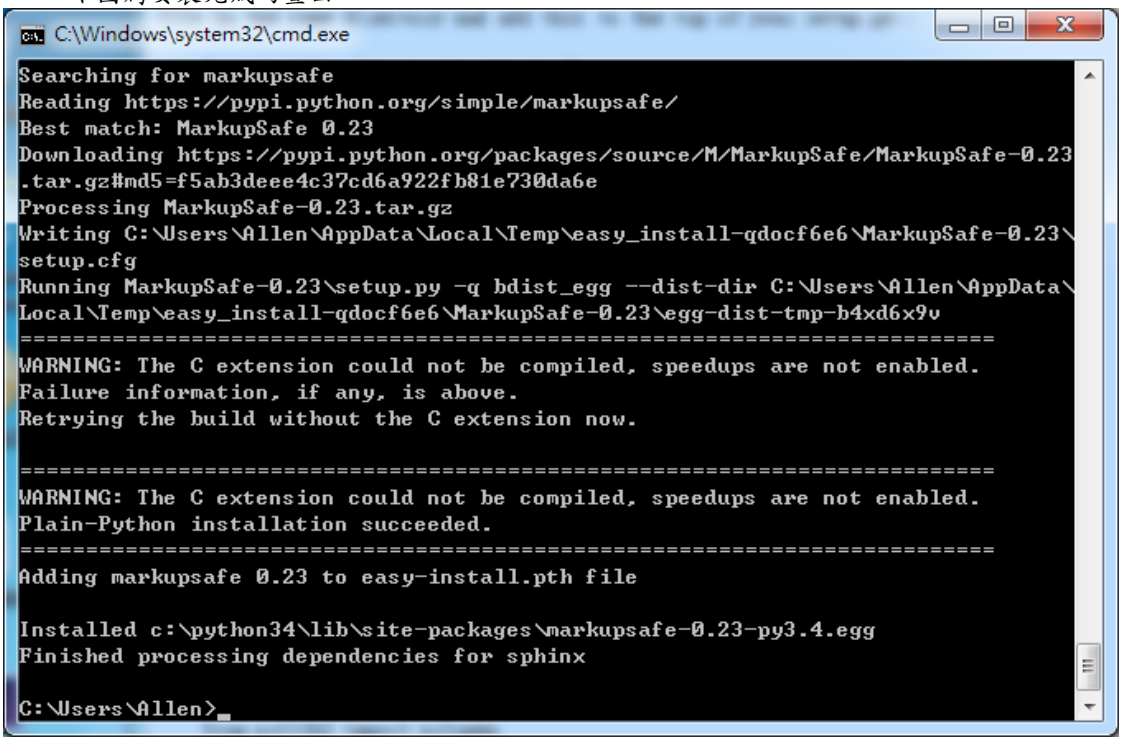


2.5 安裝Sphinx

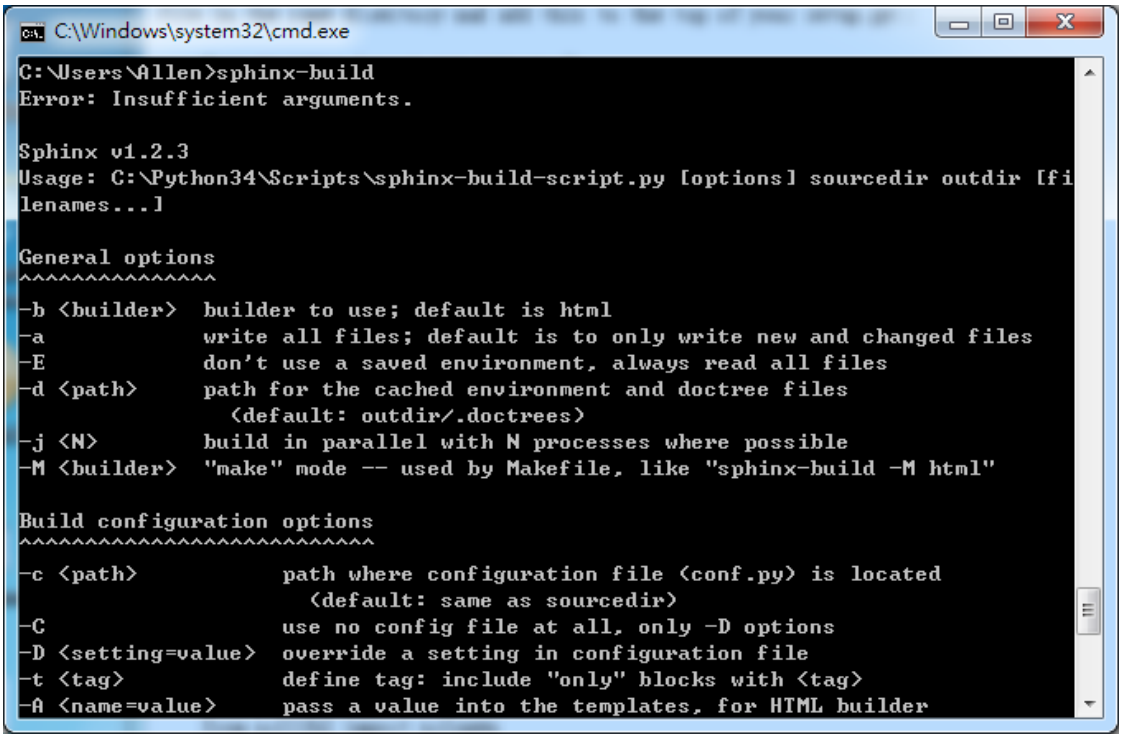
- 安裝好『easy_install』，接著輸入『easy_install sphinx』，安裝的時後需要一點時間



- 下圖爲安裝完成的畫面



- 安裝完成後，請輸入『sphinx-build』測試是否可以正常執行，如下圖



2.6 參考網站連結

- Sphinx-Doc Install

CHAPTER

THREE

GIT、GITOLITE3、GITWEB安裝

在多人開發的過程中，爲了便於整合不同使用者所建立的內容，在版本管控的多種系統中，如： CVS 、 Subversion 、 Git 幾個主要的系統中選用Git當作開發的版本管控系統，主要爲了之後可能會轉移到github上，再加上git的一些特性，如可快速的建立分支，不需中央伺服器即可在本地的檔案庫中做管理等…。

另外搭配 Gitolite 來管理使用者及檔案庫(Repository)，讓管理者便於設定檔案庫的存取權限，Gitolite綁定一個系統帳號，而其他的使用者都是透過ssh key建立，只要把key的名稱設定權限後，即可設定key及檔案庫的讀寫、執行或只能讀取等，並透過gitweb讓使用者用瀏覽器就可以觀看檔案庫的內容、記錄、版本差異、下載等功能。

3.1 GIT安裝

安裝GIT

```
sudo apt-get install git-core
```

新增使用者git

```
sudo useradd -m git
sudo passwd git
```

設置使用者的git全域參數

```
sudo su -l git
git config --global user.name "username"
git config --global user.email "email"
```

3.2 Gitolite3安裝

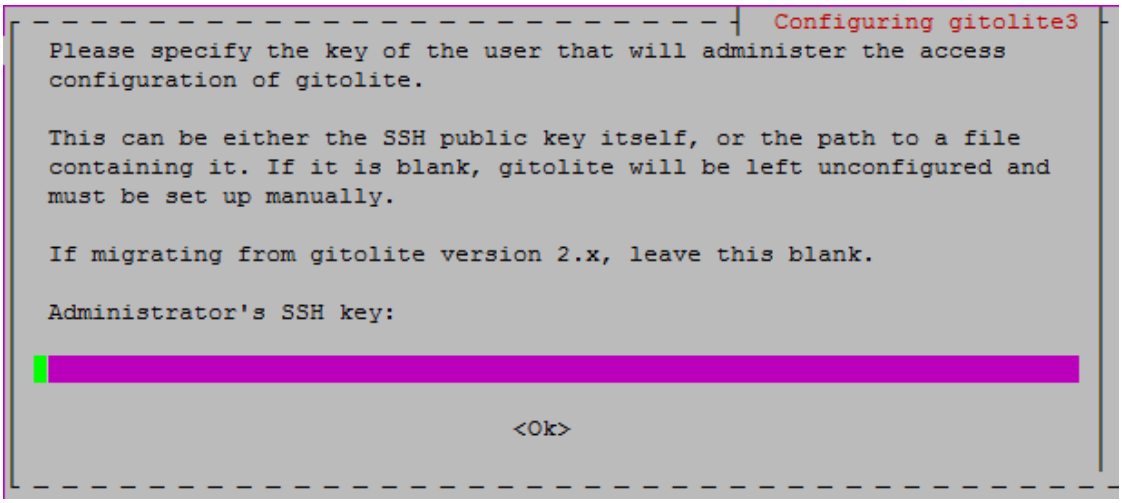
> 在安裝過程式會出現Configuring gitolite3的畫面，要輸入Administrator's SSH key，如果尚未產生好ssh key的話可以跳過，之後再設定使用sudo dpkg-reconfigure gitolite3就可以重新設定

安裝gitolite3

```
sudo apt-get install gitolite3
```

執行畫面

```
allen@uServer:/home/git$ sudo apt-get install gitolite3
[sudo] password for allen:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  git-daemon-run gitweb
The following NEW packages will be installed:
  gitolite3
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 85.3 kB of archives.
After this operation, 352 kB of additional disk space will be used.
Get:1 http://opensource.nchc.org.tw/ubuntu/ trusty/universe gitolite3 all 3.5.3.1-2 [85.3 kB]
Fetched 85.3 kB in 5s (16.6 kB/s)
Preconfiguring packages ...
Selecting previously unselected package gitolite3.
(Reading database ... 121302 files and directories currently installed.)
Preparing to unpack .../gitolite3_3.5.3.1-2_all.deb ...
Unpacking gitolite3 (3.5.3.1-2) ...
Processing triggers for man-db (2.6.7.1-1) ...
Setting up gitolite3 (3.5.3.1-2) ...
No adminkey given - not setting up gitolite. Do a dpkg-reconfigure to setup.
allen@uServer:/home/git$
```



產生git ssh key

用ssh-keygen指令產生，一般用 ssh-keygen -t rsa 即可產生公鑰、私鑰，預設會建立在家目前中的「.ssh」資料夾中，在建立的過程式可以自行修改公、私鑰的位置及檔名，公鑰預設檔案名稱爲「~/.ssh/id_rsa」，私鑰預設檔案名爲「~/.ssh/id_rsa.pub」，在產生公、私鑰的過程會要輸入passphrase，預設值是空的，如果要使用密碼請在這裡輸入後再按Enter，這樣之後連線時就會要輸入密碼後才可以登入。如下圖..

```
allen@ubuntu:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/allen/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/allen/.ssh/id_rsa.
Your public key has been saved in /home/allen/.ssh/id_rsa.pub.
The key fingerprint is:
1d:1a:20:cf:03:65:2a:a5:b2:dc:4f:bf:15:59:22:b6 allen@ubuntu
The key's randomart image is:
+--[ RSA 2048]----+
|    +.+          |
|   o B .         |
|. o . +o....     |
|.o..  ..o++.     |
|.. . . ESo.      |
|    o .   .      |
|     . . .       |
|        o        |
|       .         |
+-----------------+
```

以上說的內容可以直接用指令執行後即可產生公、私鑰，「-P」加入密碼(passphrase)，「-f」檔案產生的位置與名稱，如下

ssh-keygen指令

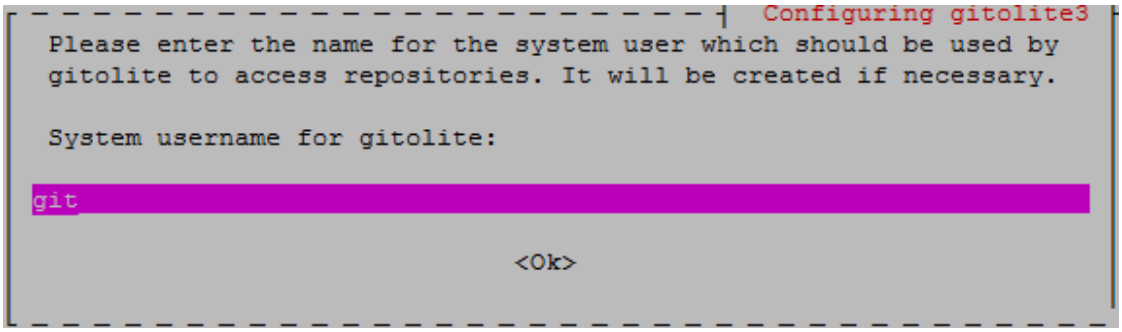
```
ssh-keygen -t rsa -P 'admin@gitolite' -f ~/.ssh/admin
```

```
git@uServer:~$ ssh-keygen -t rsa -P 'admin@gitolite' -f ~/.ssh/admin
Generating public/private rsa key pair.
Your identification has been saved in /home/git/.ssh/admin.
Your public key has been saved in /home/git/.ssh/admin.pub.
The key fingerprint is:
8c:e6:d3:79:29:38:76:a0:b3:96:03:6a:c7:6a:85:dc git@uServer
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|                 |
|       o         |
|. o   + S        |
| o.E + + . .     |
| .o.o.* = o      |
|.o o++ + o       |
|o.o...           |
+-----------------+
git@uServer:~$ ls ~/.ssh/
admin  admin.pub
git@uServer:~$
```

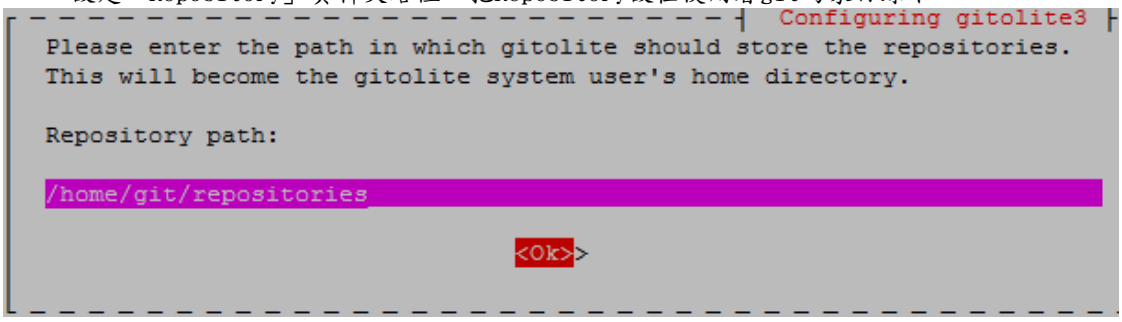
設定配置gitolite3

執行下方的設定配置指令後會出現設定畫面，首先出現的是設定gitolite3的系統管理者，在這邊我們預設定的系統管理者爲「git」

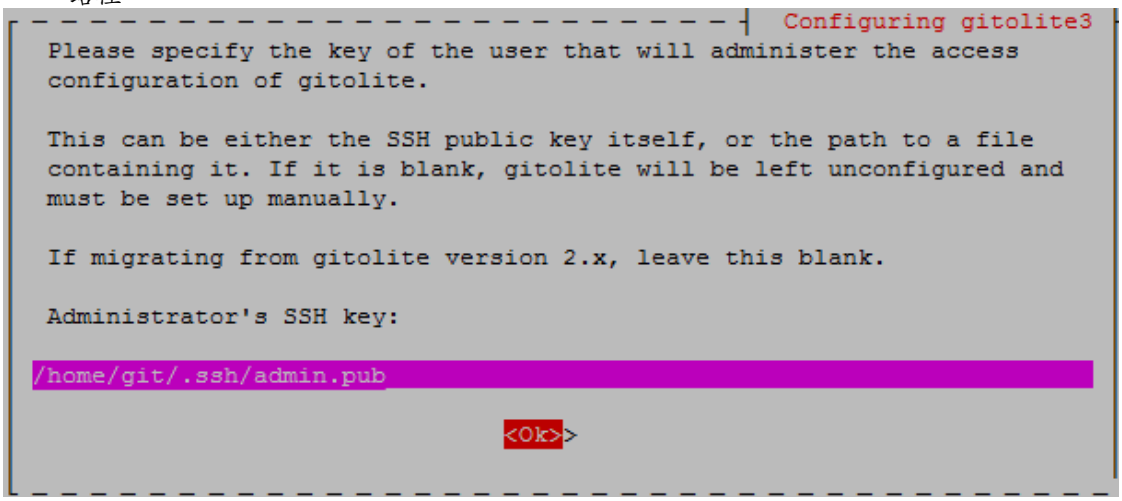
```
sudo dpkg-reconfigure gitolite3
```



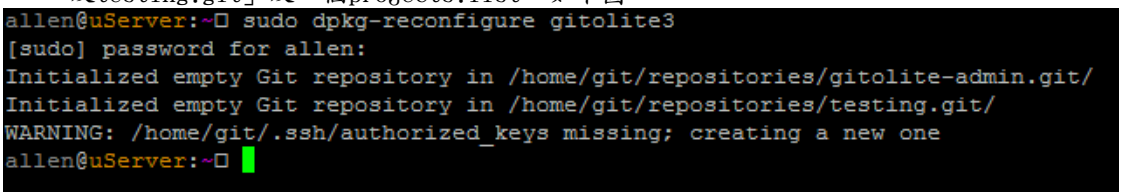
設定「Repository」資料夾路徑，把Repository設在使用者git的家目錄下



接著又回到一開始安裝時要設定的管理者ssh key，這裡用上面所設定admin.pub，要輸入完整的路徑



確認後就會看到下面的畫面，在repositories中將會自動產生兩個repo「gitolite-admin.git及testing.git」及一個projects.list，如下圖



```
git@uServer:~$ ls
projects.list  repositories
git@uServer:~$ ls -al
total 52
drwxr-xr-x 5 git  git  4096 Jun 17 17:49 .
drwxr-xr-x 5 root root 4096 Jun 17 15:26 ..
-rw------- 1 git  git    95 Jun 17 15:32 .bash_history
-rw-r--r-- 1 git  git   220 Mar 30  2013 .bash_logout
-rw-r--r-- 1 git  git  3637 Mar 30  2013 .bashrc
-rw-rw-r-- 1 git  git    46 Jun 17 15:29 .gitconfig
drwx------ 6 git  git  4096 Jun 17 17:49 .gitolite
-rw------- 1 git  git  5826 Jun 17 17:49 .gitolite.rc
-rw-r--r-- 1 git  git   675 Mar 30  2013 .profile
-rw------- 1 git  git    12 Jun 17 17:49 projects.list
drwx------ 4 git  git  4096 Jun 17 17:49 repositories
drwx------ 2 git  git  4096 Jun 17 17:49 .ssh
```

設定gitolite-admin

gitolite-admin主要是用於設定及管理repo與使用者，此外可以設定每個repo的存取權限，gitolite-admin預設只供管理者存取，所以要用設定gitolit3時所設定的administrator's ssh key來存取設定。

目前使用git使用者，也在此使用者建立ssh key，當做gitolite3的管理者，所以先登入git，並在家目錄中clone gitolite-admin做設定

```
git clone git@localhost:gitolite-admin.git
```

在git的家目錄把gitolite-admin取出後，進到gitolite-admin資料夾，將會有兩個檔案，分別爲「conf/gitolite.conf」與「keydir/admin.pub」

```
git@uServer:~/gitolite-admin$ tree
.
├── conf
│   └── gitolite.conf
└── keydir
    └── admin.pub

2 directories, 2 files
```

「conf/gitolite.conf」預設的內容如下，在gitolite-admin目前設定只供admin做讀寫及更新，其他使用者無法

```
repo gitolite-admin
    RW+     =   admin

repo testing
    RW+     =   @all
~
```

「keydir/admin.pub」keydir資料夾是放使用者ssh公鑰的地方，此資料夾中的admin.pub就是在設定gitolite時所輸入的administrator's ssh key。

Linux 新增使用者到gitolite3

先在自己家目錄中執行ssh-keygen -t rsa 產生公、私鑰，預設會存在~/.ssh/下，將公鑰id_rsa.pub更換名稱，例如目前帳號使用allen，則把id_rsa.pub改成allen.pub，並且把allen.pub放到gitolite-admin/keydir資料夾下，使用者的公鑰部分可以放到「/tmp」下，這樣切換到時用者時就不需再另外更改檔案擁有者及權限。

```
allen@uServer:~□ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/allen/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/allen/.ssh/id_rsa.
Your public key has been saved in /home/allen/.ssh/id_rsa.pub.
The key fingerprint is:
ec:c3:86:fe:d4:af:c2:9a:90:9a:0b:64:cf:ea:26:97 allen@uServer
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|                 |
|       .         |
| o      S        |
|o o   . + .      |
|.   +o ..* .     |
|.oEo o +o. .     |
|+++.  +o.....    |
+-----------------+
allen@uServer:~□ ls ~/.ssh/
id_rsa  id_rsa.pub
allen@uServer:~□ cp ~/.ssh/id_rsa.pub ~/.ssh/allen.pub
allen@uServer:~□ ls ~/.ssh/
allen.pub  id_rsa  id_rsa.pub
```

在使用者allen下就先cp一份公鑰到/tmp下，

```
git@uServer:~/gitolite-admin/keydir$ ls
admin.pub
git@uServer:~/gitolite-admin/keydir$ cp /tmp/allen.pub .
git@uServer:~/gitolite-admin/keydir$ ls
admin.pub  allen.pub
git@uServer:~/gitolite-admin/keydir$
```

接下來就執行git指令，把使用者的公鑰放到gitolite-admin repo中，在過程中有出現「Enter passphrase for key...」這部分因爲在建立admin公、私鑰時有輸入密碼，所以在這個時後就需要輸入之前的密碼以確定執行git push上傳更新。

```
git@uServer:~/gitolite-admin/keydir$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

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

        allen.pub

nothing added to commit but untracked files present (use "git add" to track)
git@uServer:~/gitolite-admin/keydir$ git add .
git@uServer:~/gitolite-admin/keydir$ git commit -m 'add user:allen'
[master 8eb038e] add user:allen
 1 file changed, 1 insertion(+)
 create mode 100644 keydir/allen.pub
git@uServer:~/gitolite-admin/keydir$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

In Git 2.0, Git will default to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Enter passphrase for key '/home/git/.ssh/id_rsa':
Enter passphrase for key '/home/git/.ssh/id_rsa':
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 667 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@localhost:gitolite-admin.git
   a16b8c2..8eb038e  master -> master
git@uServer:~/gitolite-admin/keydir$
```

新增GIT Repo

使用者新增好了之後，我將建立一個給allen專用的repo，加入了

```
repo allen
        RW+ = allen
```

這樣設定在gitweb預設不會將repo列出，如果要供大家讀取，則再加入一行即可

```
R     =  @all
```

```
repo gitolite-admin
    RW+     =   admin

repo testing
    RW+     =   @all

repo allen
    RW+     =   allen
```

接著做相同的動作，如下

```
git add .
git commit -m 'add repo:allen'
git push
```

```
git@uServer:~/gitolite-admin/conf$ vim gitolite.conf                    [10/83]
git@uServer:~/gitolite-admin/conf$ git status
On branch master
Your branch is up-to-date with 'origin/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:   gitolite.conf

no changes added to commit (use "git add" and/or "git commit -a")
git@uServer:~/gitolite-admin/conf$ git add .
git@uServer:~/gitolite-admin/conf$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   gitolite.conf

git@uServer:~/gitolite-admin/conf$ git commit -m 'add repo:allen'
```

當執行git指令，把設定檔更新後，回到「~/repositories」中，將會看到自己建立了一個repo「allen.git」

```
git@uServer:~/repositories$ ls -al
total 20
drwx------ 5 git git 4096 Jun 20 11:37 .
drwxr-xr-x 7 git git 4096 Jun 20 11:37 ..
drwx------ 7 git git 4096 Jun 20 11:37 allen.git
drwx------ 8 git git 4096 Jun 20 11:37 gitolite-admin.git
drwx------ 7 git git 4096 Jun 17 17:49 testing.git
git@uServer:~/repositories$
```

用使用者allen 去clone allen.git，執行完後會如下圖，將會在執行的目錄中增加一個allen的git repo，預設裡面是空的，只有一個.git的設定檔資料夾

```
allen@uServer:~/git$ git clone git@allen.go38.net:allen.git
Cloning into 'allen'...
The authenticity of host 'allen.go38.net (120.106.134.5)' can't be established.
ECDSA key fingerprint is a9:c8:28:c9:32:ba:a6:32:75:7d:6c:1a:64:29:c3:d2.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'allen.go38.net,120.106.134.5' (ECDSA) to the list of
 known hosts.
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
allen@uServer:~/git$ ls
allen
```

repo初始內容

```
allen@uServer:~/git/allen$ tree -a
.
└── .git
    ├── branches
    ├── config
    ├── description
    ├── HEAD
    ├── hooks
    │   ├── applypatch-msg.sample
    │   ├── commit-msg.sample
    │   ├── post-update.sample
    │   ├── pre-applypatch.sample
    │   ├── pre-commit.sample
    │   ├── prepare-commit-msg.sample
    │   ├── pre-push.sample
    │   ├── pre-rebase.sample
    │   └── update.sample
    ├── info
    │   └── exclude
    ├── objects
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags

10 directories, 13 files
allen@uServer:~/git/allen$
```

3.3 安裝GitWeb

安裝apache2

```
sudo apt-get install apache2
```

設定啓用cgi

```
sudo vim /etc/apache2/sites-enabled/000-default.conf
把這行前方的#拿掉 「Include conf-available/serve-cgi-bin.conf」
sudo a2enmod cgi
sudo apache2 restart
```

```
allen@uServer:/var/www/html$ sudo vim /etc/apache2/sites-enabled/000-default.con
f
allen@uServer:/var/www/html$ sudo a2enmod cgi
Your MPM seems to be threaded. Selecting cgid instead of cgi.
Enabling module cgid.
To activate the new configuration, you need to run:
  service apache2 restart
allen@uServer:/var/www/html$ sudo service apache2 restart
 * Restarting web server apache2
   ...done.
allen@uServer:/var/www/html$
```

安裝gitweb

```
sudo apt-get install gitweb
```

設定gitweb

安裝完後資料放在預設目錄「/usr/share/gitweb/」下，而設定檔放在「/etc/gitweb.conf」，apache2的預設目錄「/var/www/html」，將gitweb會讀取到的相關css及圖檔資料夾static做連結到網站的根目錄

```
sudo ln -s /usr/share/gitweb/static /var/www/html
```

編輯gitweb.conf設定參數如下圖

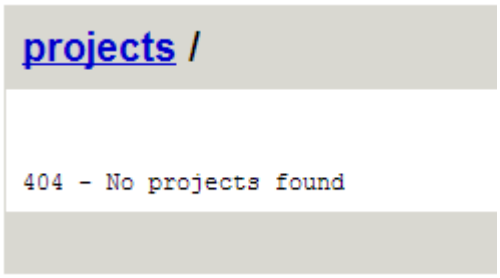
```
sudo vim /etc/gitweb.conf
```

注意$projectroot，$projects_list要改成自己所設定的路徑

```
# path to git projects (<project>.git)
#$projectroot = "/var/lib/git";
$projectroot = "/home/git/repositories";
# directory to use for temp files
$git_temp = "/tmp";

# target of the home link on top of all pages
$home_link = $my_uri || "/";

# html text to include at home page
$home_text = "indextext.html";

# file with project list; by default, simply scan the projectroot dir.
#$projects_list = $projectroot;
$projects_list = "/home/git/projects.list";

# stylesheet to use
@stylesheets = ("/static/gitweb.css");

# javascript code for gitweb
$javascript = "/static/gitweb.js";

# logo to use
$logo = "/static/git-logo.png";

# the 'favicon'
$favicon = "/static/git-favicon.png";

# git-diff-tree(1) options to use for generated patches
#@diff_opts = ("-M");
@diff_opts = ();
```

gitweb權限設定 當設定好後啓動apache2，用瀏覽器進入 http://localhost/cgi-bin/gitweb.cgi 會發現

出現404，並沒有repo專案出現，在這裡需要做算讀取權限的設定



將git加入群組www-data

```
sudo usermod -a -G git www-data
```

資料夾repositories設定爲755 檔案projects.list設定爲644 資料夾repositories中的repo，原本資料夾權限預設爲700，群組的部分不能讀取，如果要開放的話，請將該xxx.git資料夾設定爲755，如下方指令要開放testing.git，

```
chmod 755 -R testing.git
```

當設定以上的權限後重新啓動apache2即可看到所開放的repo，但如果使用者有做git push後會發現，網頁上該被push過的repo又不見了，但再執行一次chmod 755 -R testing.git後網頁上又出現了，發現在push後有些資料夾的權限會變成預設的700，所以要再做下面的修改，將設定repo預設的建立權限爲750，這樣使用者在做push更新時，相關檔案建立時www-data群組就可以做存取的動作，修改如下

在使用者git家目錄下

```
vim ~/.gitolite.rc
```

將UMASK的預設值0077改成0027後，存儲並重新啓動apache2，再重新push一次後就會發相部分相關檔案在push後所建立的權限爲750，並且gitweb網頁也可以正常的觀看。

```
%RC = (

    # ------------------------------------------------------------------

    # default umask gives you perms of '0700'; see the rc file docs for
    # how/why you might change this
    #UMASK                           =>  0077,
    UMASK                           =>  0027,

    # look for "git-config" in the documentation
    GIT_CONFIG_KEYS                 =>  '',
```

3.4 參考網站連結

- Git 版本控制系統 http://ihower.tw/git/index.html
- 30天精通Git版本控管 http://ithelp.ithome.com.tw/ironman6/player/doggy/dev/1
- Git Community Book 中文版 http://gitbook.liuhui998.com/index.html
- Pro Git Book http://git-scm.com/book/zh-tw

CHAPTER

FOUR

GIT HOOKS

4.1 相關連結

Git 客製化 - Git Hooks GitHub-WebHook

CHAPTER

FIVE

共同寫作操作流程

5.1 相關軟體安裝

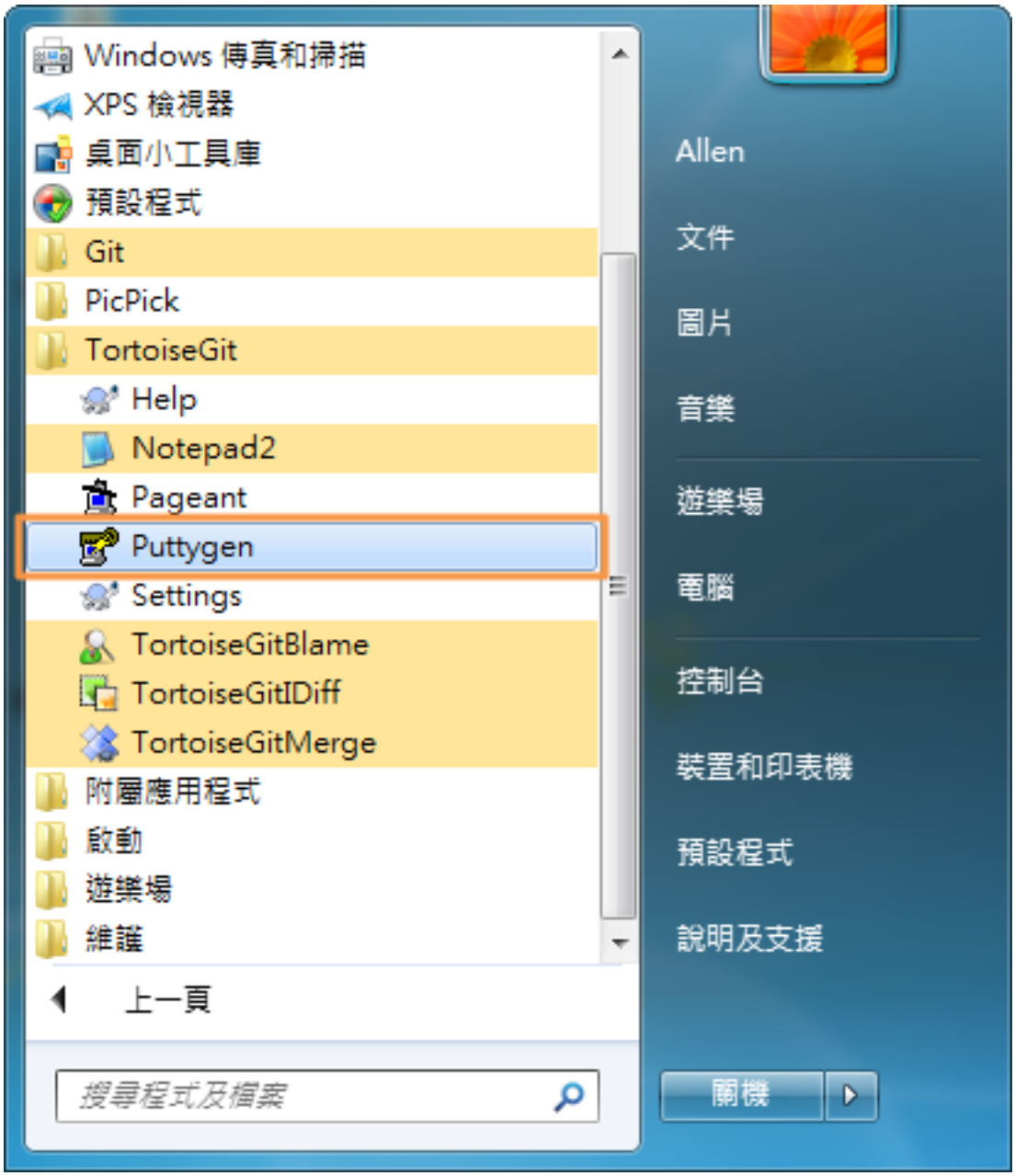
1. TortoiseGit (安裝教學)
2. Git (安裝教學)

5.2 建立使用者帳號(SSH-KEY)

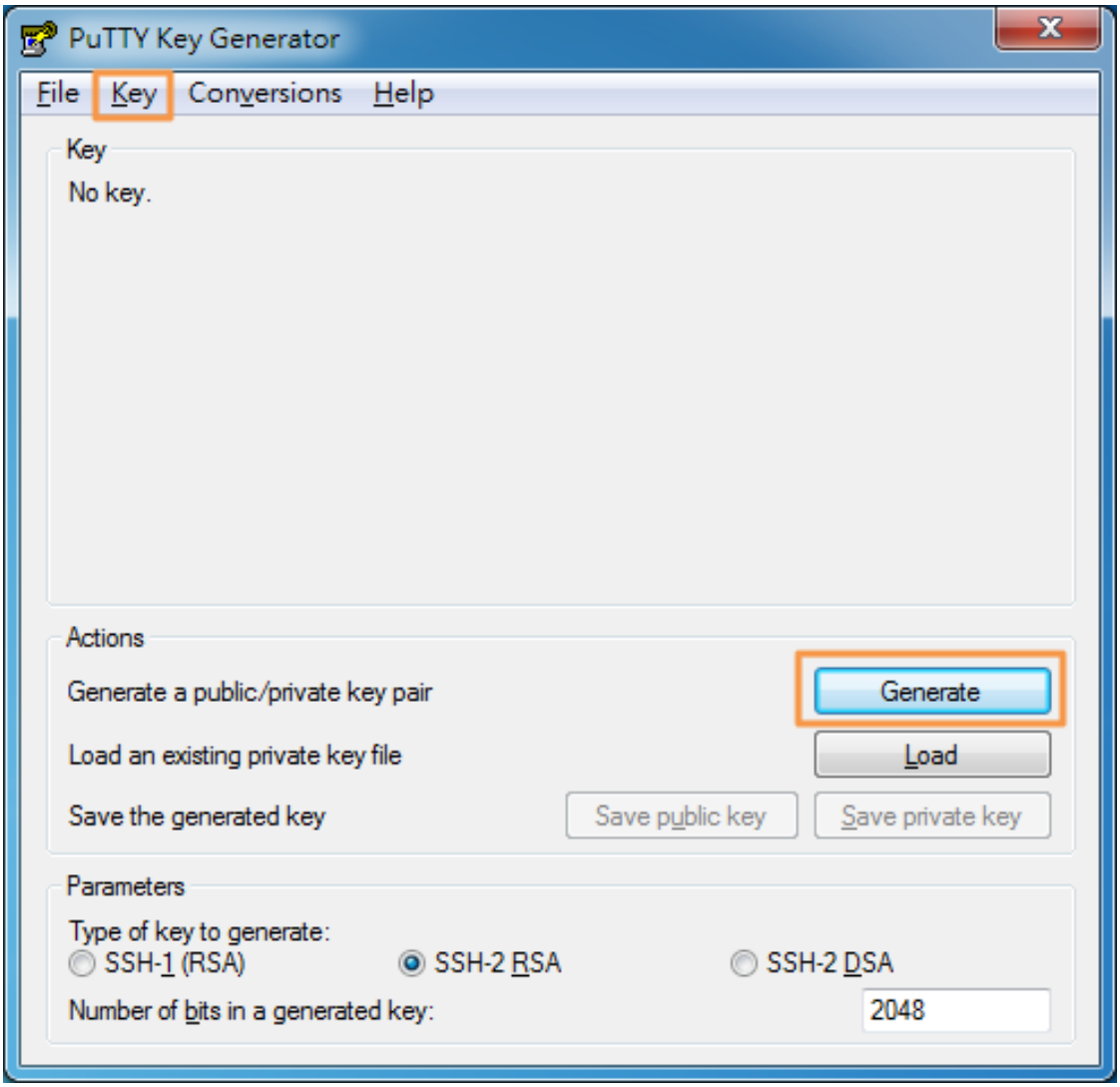
在安裝好TortoiseGit與Git後，使用此軟體建立ssh公鑰(Public Key)與私鑰(Private Key)，安裝的兩個軟體都可以建立ssh-key，使用安裝TortoiseGit附帶一起安裝的 Puttygen 建立，此方式是有圖形介面(GUI)，較容易看到所產生的內容，操作方式請參照 Puttygen建立ssh-key，如果要使用Git Bash建立ssh-key請參照 Git Bash建立ssh-key，此方式是屬於文字介面，操作方式如Windows的『命令題示字元模式』與Linux的『console模式』，但操作的指令比較像Linux的console模式。

5.2.1 Puttygen建立ssh-key

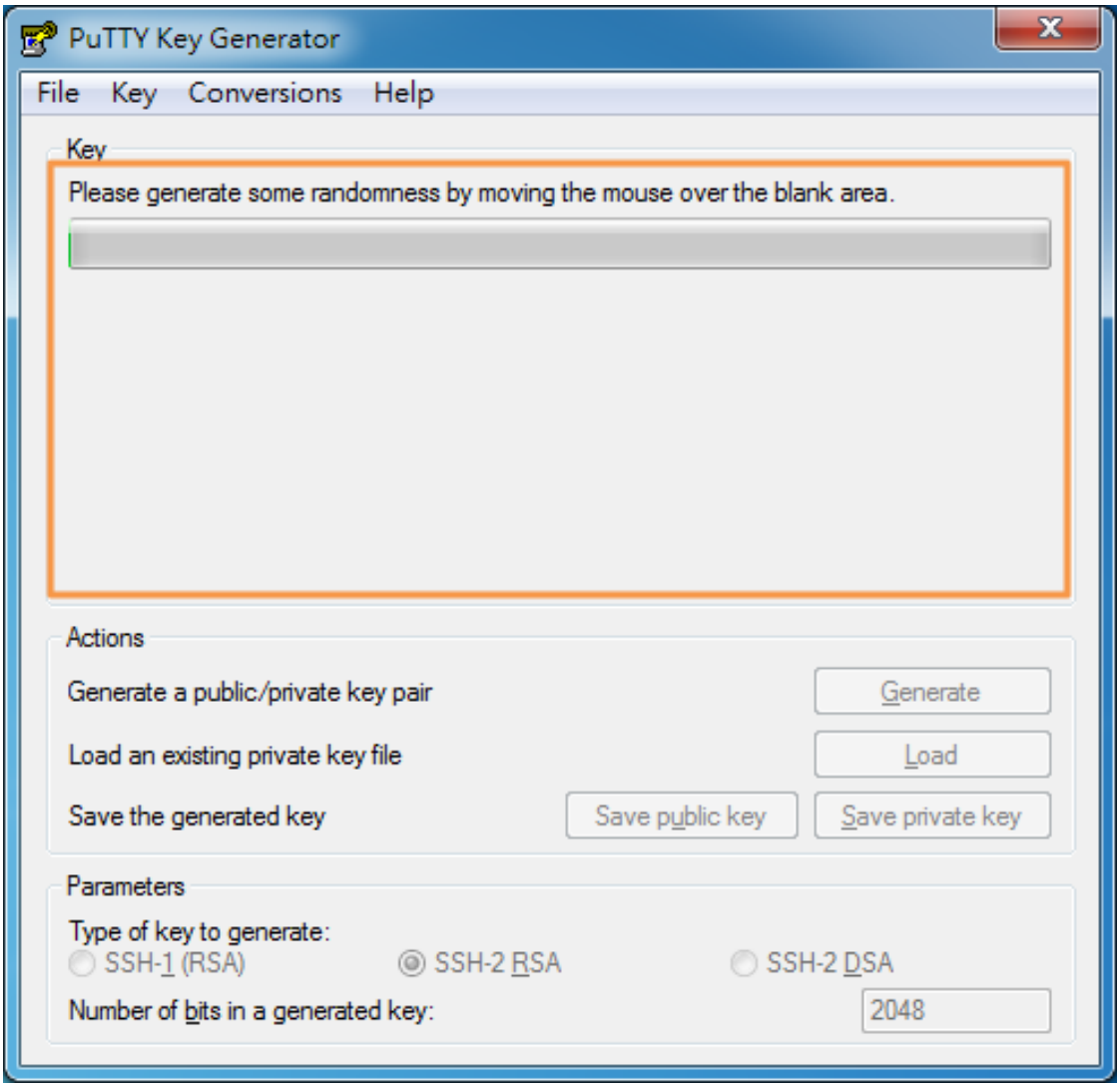
- 啓動Puttygen，打開『開始程式集』，找到 TortoiseGit 點選打開後就會看到如下圖的內容，點選執行『Puttygen』



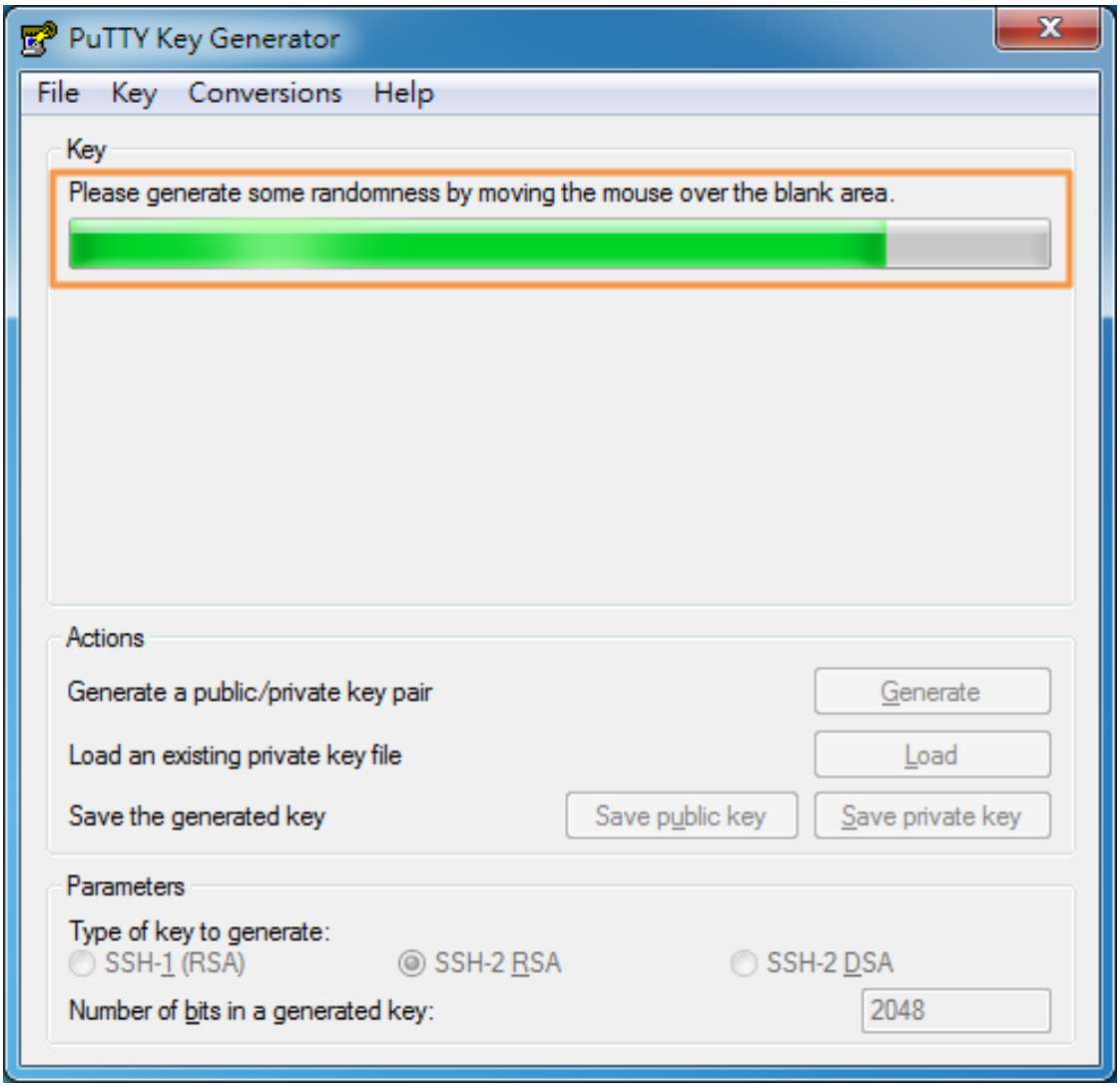
- 啓動Puttygen後的畫面如下



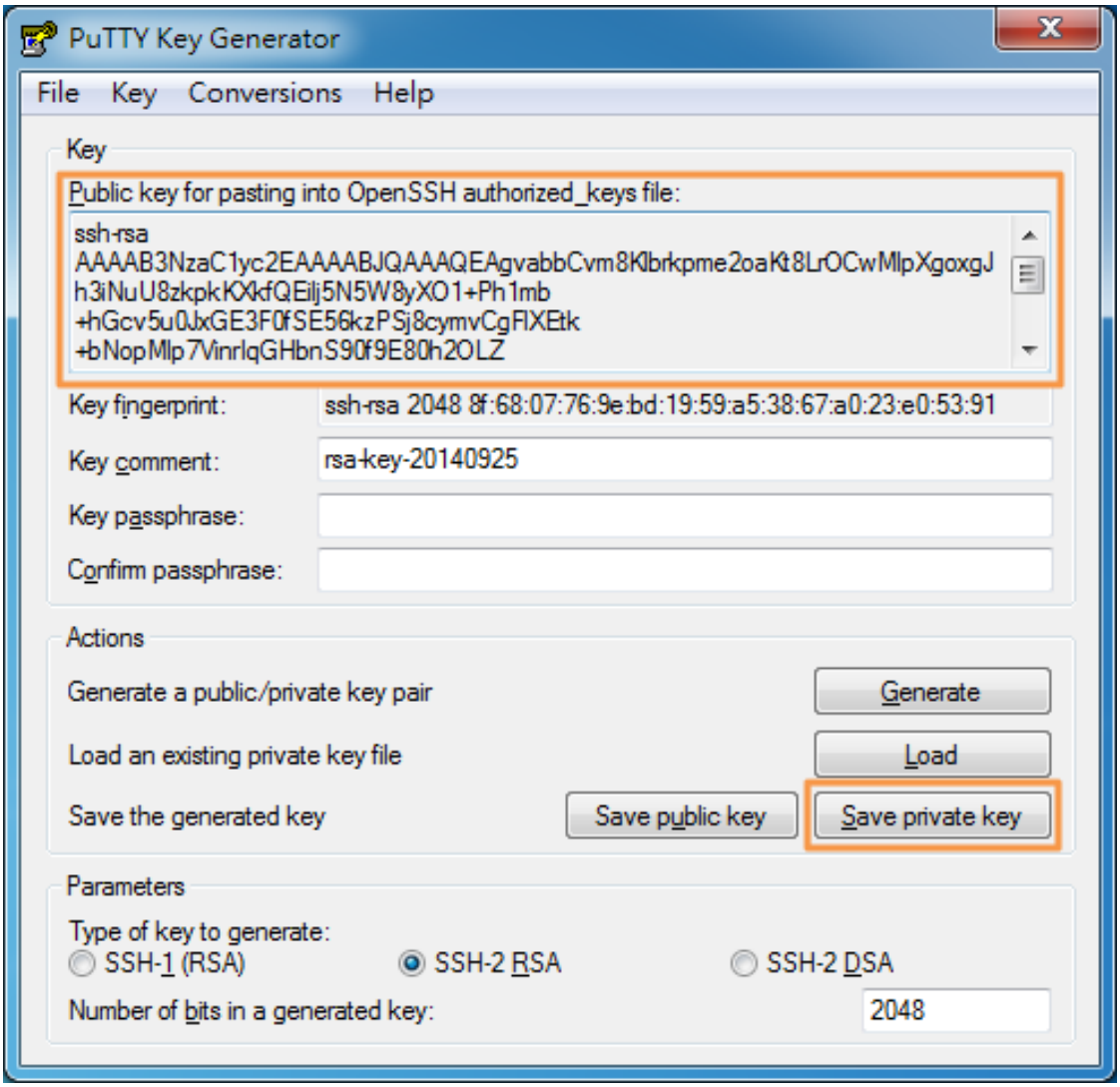
- 點選畫面中的『Generate』，或者是上方選單的『Key->Generate key pair』，點選後會看到畫面上多了一串字『Please generate some randmness by moving the mouse over the blank area.』，簡單來說就是『請在key的空白區塊移動滑鼠以產生ssh-key』及一個Process Bar以顯示ssh-key產生的進度。



- 不斷的在key的空白區塊移動滑鼠時，Process Bar的目前進度就會一直增加



- 當Process Bar跑完全都變綠色時，代表已完成產生ssh-key，這時key的空白區塊就會改變成顯示所產生的ssh-key



- 這時可用下方的『Save private key』按鈕來儲存私鑰，點選按鈕後會跳出存儲視窗，先選擇要存放的資料夾，接著在檔案名稱的地方輸入檔名，副檔名爲『ppk』，最後點選存檔即可。

  公鑰正常來說也是點選『Save public key』然後一樣的方式就可以完成，但發現直接存檔，把此公鑰放在伺服器上面時會無法正常使用，最後發現因爲在每64個字母後面就加一個空白，因爲此空白造成上傳到伺服器時無法正常的運作，只要用文字編輯器打開，將每64個字母後方的空白刪除後存檔即可，另外一個方式就是將『Public key for pasting into OpenSSH authorized_keys file:』下方區塊的內容全選複製，然後打開文字編輯器，貼上所選取的public key，再儲存檔案就可以了，公鑰的副檔名爲『pub』。

  建議：公鑰部分，直接複製貼上畫面中的Public key，避免在刪除空白時不小時刪錯，造成無法使用。

  點選『Save public key』所儲存的內容如下

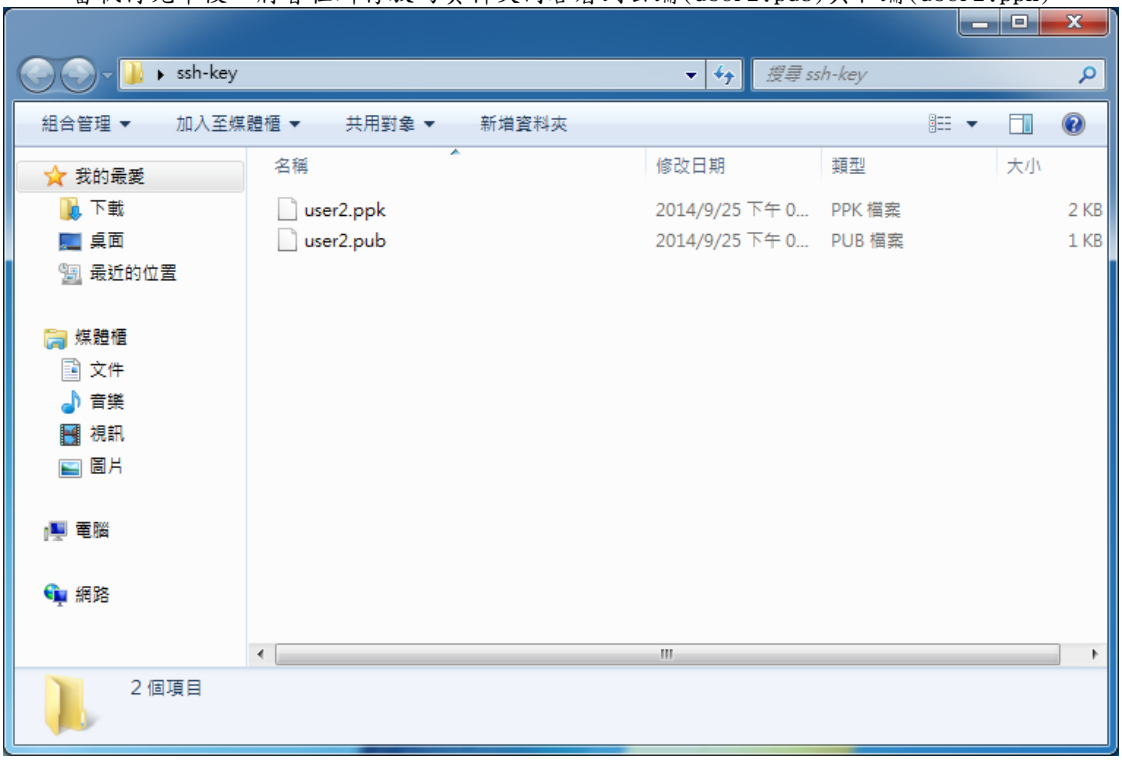
  ```
  ---- BEGIN SSH2 PUBLIC KEY ----
  Comment: "rsa-key-20140925"
  AAAAB3NzaC1yc2EAAAABJQAAAQEAieoX4nQSi9b0wm5phNZA7wimvtBLiK37IUcG
  YzGX0FukwmPDF6hZ1qIvxwPHn5Qv1uI4EIFdTH1ybWAh1K8awmQuoSKkaFwRxM8q
  x2DXyHdHGTwbFjNSnh/mRD9dNrR0kXfqqkb7SyT5wGfi5dDRpquo9y8Mv8kkXkUo
  ```

```
x0o1I7bT377FBT1Y19VAFIF9PpNeBYpwI+1KR5aB6q7ABx5ynWo6YFDVPN03czB1
1s1rSo7/Y3rlvB8r0zxSyTMj8OXo4ZirzefwOclge8fpfs0XiGgOT6FrTerXzxWE
zF7Sc115SdLsz122G4znCpf3AIQ795Zcwg5/Qpm9KFJ/6gU1/Q==
---- END SSH2 PUBLIC KEY ----
```

將儲存的內容刪除每64個字母後方的空白

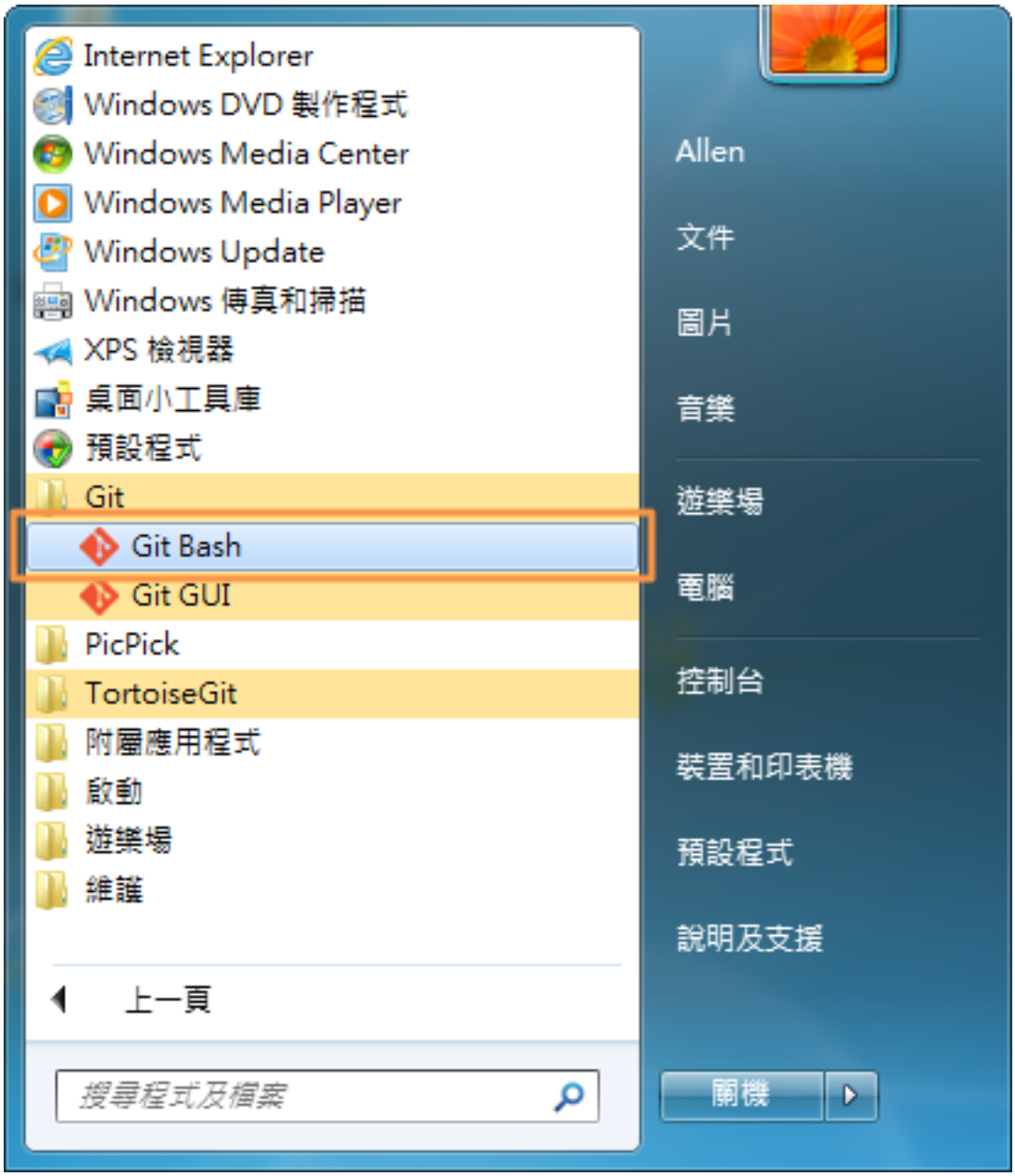
```
---- BEGIN SSH2 PUBLIC KEY ----
Comment: "rsa-key-20140925"
AAAAB3NzaClyc2EAAAABJQAAAQEAieoX4nQSi9b0wm5phNZA7wimvtBLiK37IUcGYzGX0FukwmPDF6hZ1qIvxwPHn5Qv1uI4EIFdTH1ybWAh1K8awmQ
---- END SSH2 PUBLIC KEY ----
```

- 當執行完畢後，將會在所存放的資料夾內容看到公鑰(user2.pub)與私鑰(user2.ppk)



5.2.2 Git Bash建立ssh-key

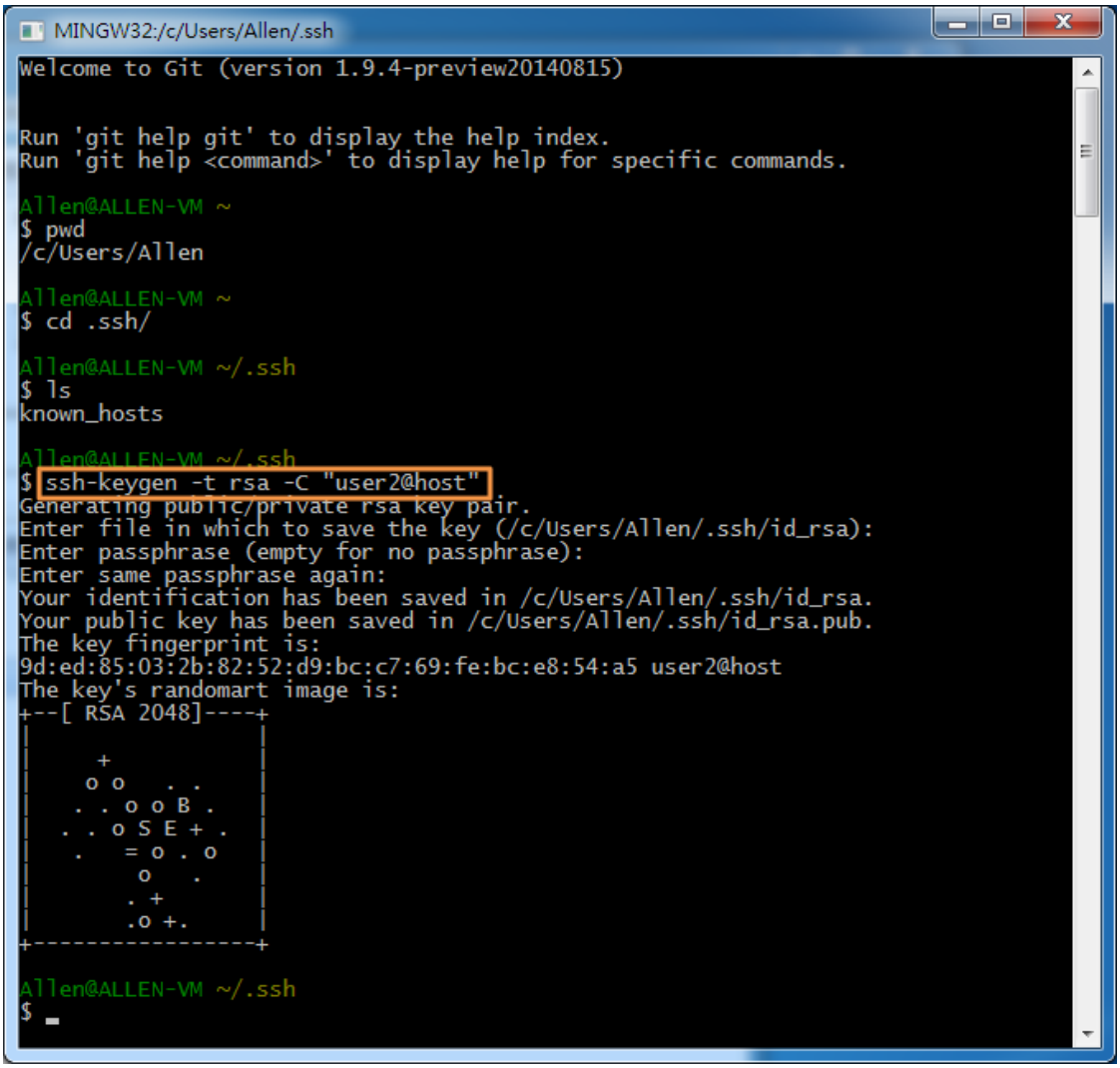
- 啓動Git Bash，打開『開始程式集』，找到 Git 點選打開後就會看到如下圖的內容，點選執行『Git Bash』



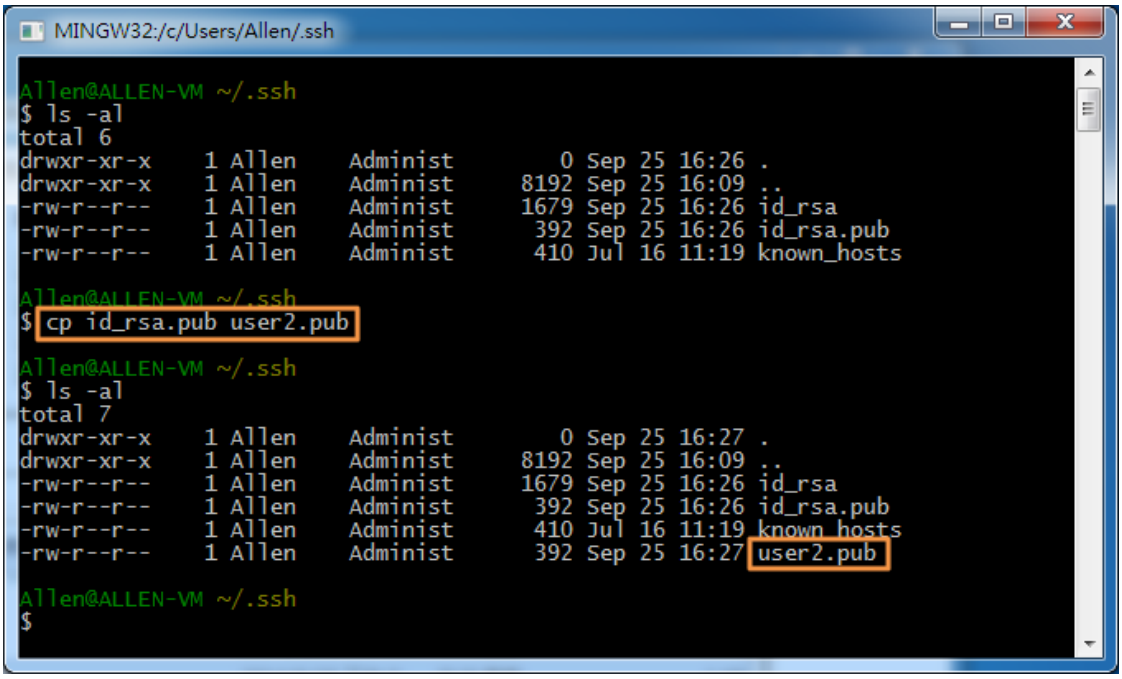
- 建立ssh-key的指令

```
ssh-keygen -t rsa -C "user2@host"
```

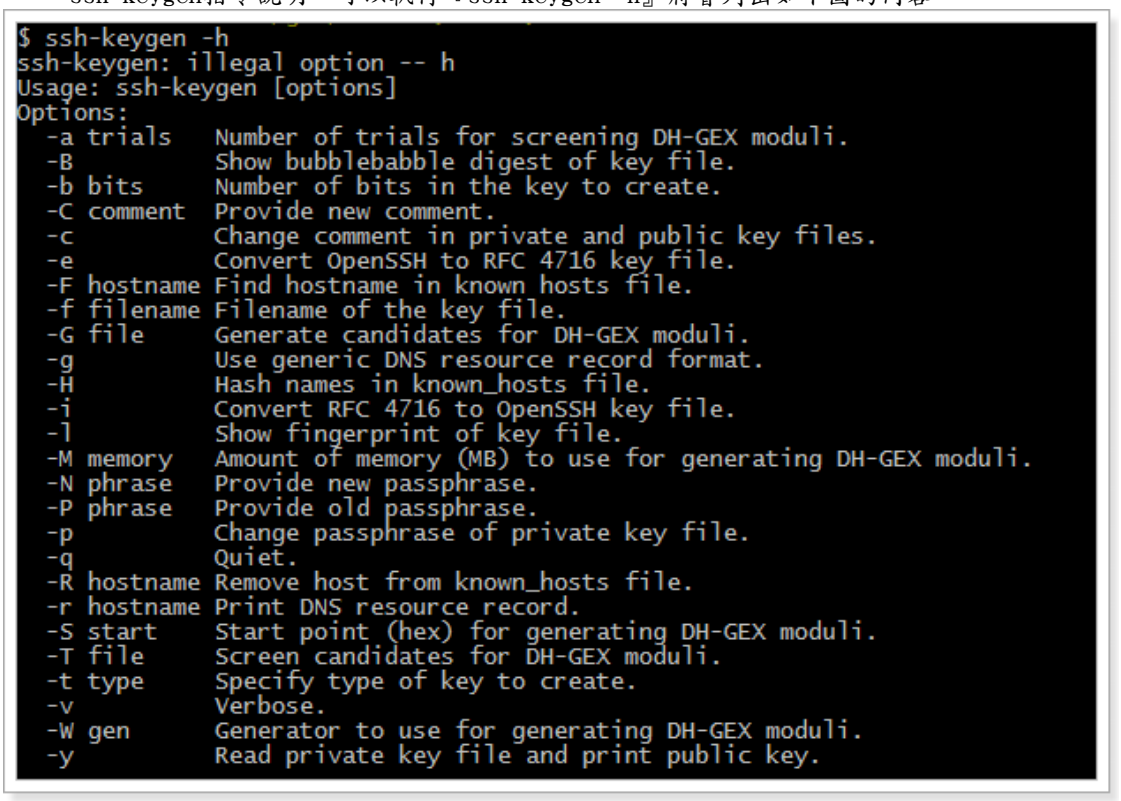
進入畫面後，可以先用pwd確認目前的資料夾路徑，然後進入到想要存放ssh公鑰(Public Key)與私鑰(Private Key)的資料夾，然後執行『ssh-keygen -t rsa』即可，『-C ``user2@host``』這是加入註解說明文字，有加這個參數，會將雙引號中的文字加在所產生ssh-key的最後方，在建立過程會出現『Enter passphrase』這部分如果是空白沒有輸入密碼，在使用git操作時將不會要求輸入密碼，反之，在這些輸入密碼後，在操作git也會出現要輸入密碼訊息，輸入正確的密碼後才會執行git指令。



- 執行完後，可以輸入『ls -al』列出目前資料夾產生了什麼檔案，基本上會出現公鑰(id_rsa.pub)與私鑰(id_rsa)這兩個檔案，接著將公鑰(id_rsa.pub)複製成另一個檔案，例如目前的使用者名稱爲user2，所以執行『cp id_rsa.pub user2.pub』，再次執行『ls -al』檢查資料夾中是否有新增一個『user2.pub』檔案。

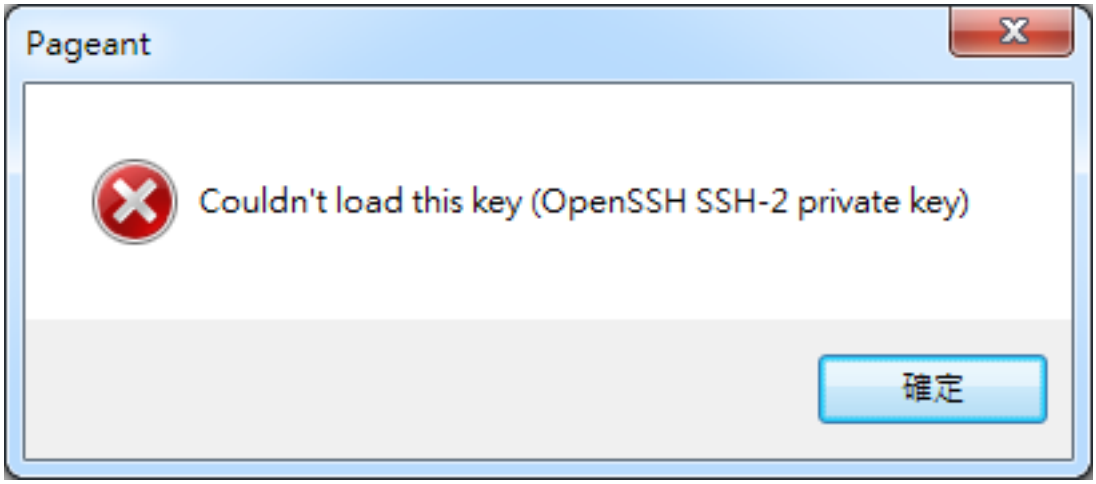


- ssh-keygen指令說明，可以執行『ssh-keygen -h』將會列出如下圖的內容

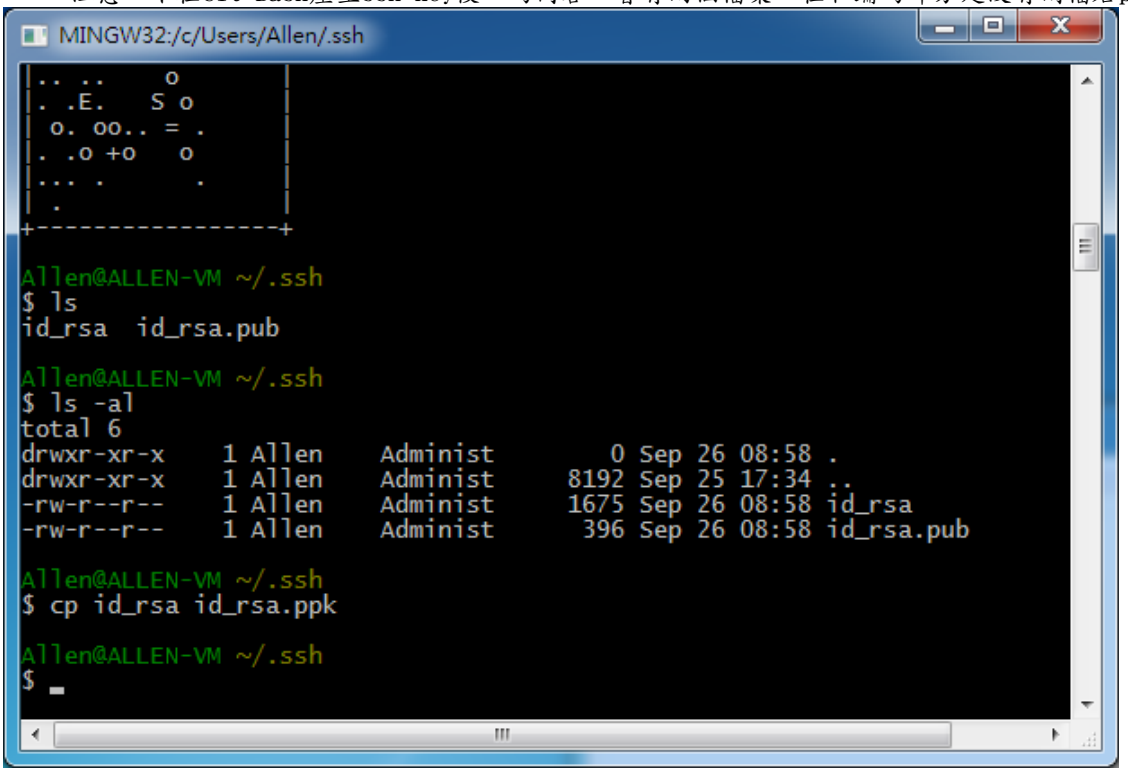


轉換Git Bash產生的私鑰格式

- 當使用 Git Bash建立ssh-key 的方式建立ssh公鑰(Public Key)與私鑰(Private Key)後，在之後使用Pgageant做連線時會出現下面的視窗，顯示所選擇的key不能載入，因爲格式爲OpenSSH而所使用的工具是Putty系列的工具，所以需要做一下式格的轉換



- 注意一下在Git Bash產生ssh-key後，的內容，會有兩個檔案，在私鑰的部分是沒有副檔名ppk的，



Git Bash產生的私鑰格式

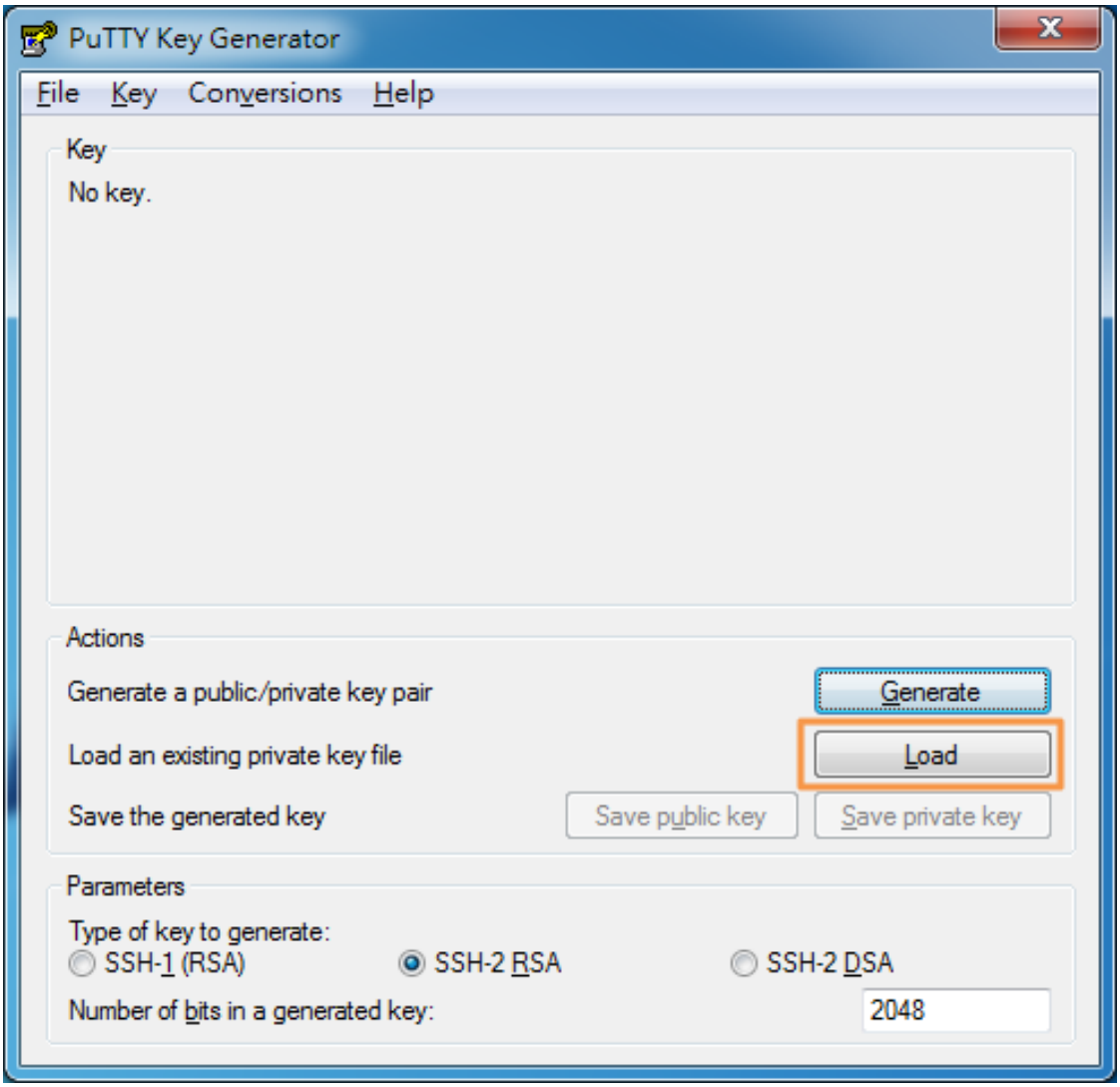
```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA21wVMmtJSprBBhBifFDyIvTMdFd3oGQw4XegQunAR0N/MdGd
aQSJdOJz1jkd81pOXnpkdBjfpnvxQO09Ad0QbpfLu1/hOb/74S8M1Ivc3tHPmefd
DDDAGokEd8vevJGyXceLv6yIoZxgNP9npDAE1k+DiHAVUtk3IBMxaP6bUy36RZbx
CakSemFqGtO2L51/9YI4j3LmGbbnHh8K4YLD9dHfbey7CN3KgvPKkptW/49LrbWZ
QhW16mXGRWfSIwWEML6B+EsUFqwhVte9TFTtxINJa3cvHwbxmsgmVP51MOMiWpUf
```

```
iK7LM4BdW7dpy5KW+FzGj5UwVxpoku2HRRA14wIDAQABAoIBAQCgW1dvjTh931V+
/11Qscfgv/36irp7u0J24IF6iPCz5+bC345n6BYoKScASW7X7SZfMK/goxJT3UVG
/3T7OMS7ZCJeLsAx/GPAgS18KiZjiRNvWi2grL6yZzmp7ZI9XXD711UU4KbdJbi6
5X65axUOKgBorC8aaoo+22kCbSXVDmrprYoJUOkAbc2Jpjp64j+/9Lz1WA6791/W
mtt9xd1DBYVMBbDIYhqpax7xQlrN1YgpdwLUI6kK1myAW2ZOUr0qT2a0rysYvKn0
e9d8qaGrtNqc19/Qb1BALg3PaME+oIPkOD1oqI86g/WtjQgjfmYvaS7SVnyj0p9j
EE2XiK4pAoGBAP/isRKIJzI0aUgImE5y0cALeJK5kvJuDiAx1kRVTou jXkZqx+t4
dvYQRrtaVuw1NF4EWKr6kMCqHXu/cSuPGFhoFMyyD25u jiPuU+bZLwF3hyUtXuU0
XDYHtuq19XW01RgII2txyPVXSdGuWDKkLqqaBwP+7ymo4mBpnJF0DHafAoGBANt1
NSK4qFI4PYCK jruyIpUucou4tGsQ1gckT j41pSNNNJdaWAYKUDtspPevZJGDdi92
BE1XhSrEq2gS5dpG/x6qYvHMnyz1hLyG8q1KHBV93pX0B3cgRZ1BkTK+Rn4RWuGY
eD134GFckp+GhWri8AvzCqgJ jFBAO0eopbWa3M49AoGAKkD+yHafMk7a58Az11Ca
ZLIAOQpXz5yIzJw jmikUDU7u1GXxmVqNG1 jZtTpXvC2U9i11ktZJb0cqczLLt0ur
031sYd jNX4 j2A8zpIeefMX jRY jw5vVOGHo3rkJp3a/Zuccd2QdnK02kskW1euw4N
frJskiZYy+JNb2pBJmupWzUCgYBWHAhMZm5NTJn9PODC jyP4qy/w2PmZ5PkpN jqp
MfY5Ass29BED1bo jhv633Q819YJDHyiRapDq9h5XiMq9ugvSiYKnY+W1K2T44G6I
tHa0 jEPxPdU6DeShPNPv0C/KuZ6SVFQgIIRfT4aMp6 jkBsWvup6czppKxthNNaRH
vMsU9QKBgHbJQJugAdxLqJh0udFQDo/5dZ1BQ1GL0PPcCVrWCU8D04dzPM89JguN
fmG0aqT4a4hPCc1f0rz71LKe6VkvVcqMYqE99PaSxshPptVq0Wr36m0eHaC5wogS
3U8V3u06mNwLPqIaRPnIQ/FAiRDHwheAkPx ji/K01DTWFxYioFo5
-----END RSA PRIVATE KEY-----
```

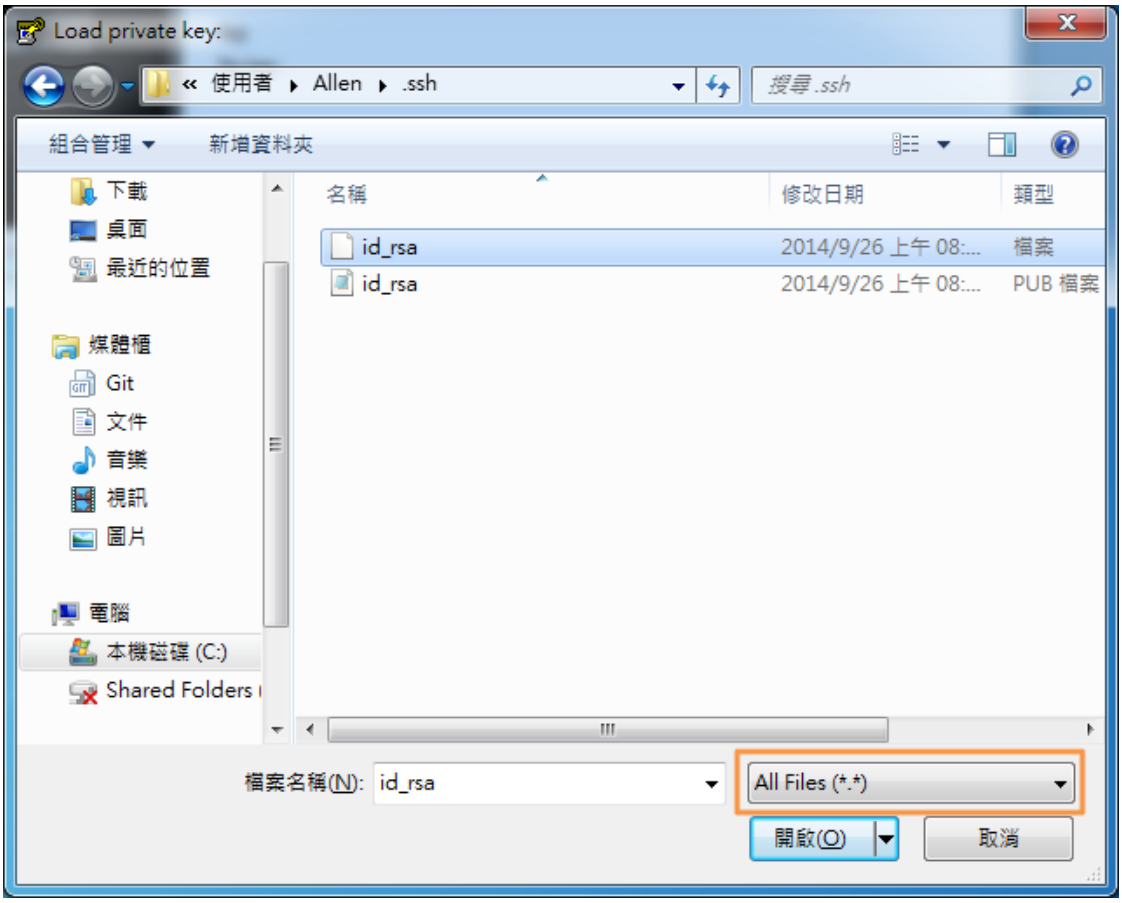
經Puttygen轉換後的私鑰內容

```
PuTTY-User-Key-File-2: ssh-rsa
Encryption: none
Comment: imported-openssh-key
Public-Lines: 6
AAAAB3NzaC1yc2EAAAADAQABAAABAQDbXBUya01KmsEGEGJ8UPIi9Mx0V3egZDDh
d6BC6cBHQ38x0Z1pBI104nOWOR3yWk5eemR0GN+me/FA470B3RBu18u7X+E5v/vh
LwyUi9ze0c+Z590MMMAaiQR3y968kbJdx4u/rIihnGA0/2ekMATWT40IcBVS2Tcg
EzFo/ptTLfpF1vEJqRJ6YWoa07YvnX/lg jiPcuYZtuceHwrhgsP10d9t7LsI3cqC
88qSm1b/j0uttZ1CFaXqZcZFZ9I jBYQwvoH4SxQWrCFW171MV03Eg01rdy8fBvGa
yCZU/nUzQyJa1R+IrsszgF1bt2nLkpb4XMaP1TBXGmiS7YdFEDX j
Private-Lines: 14
AAABAQCgW1dv jTh931V+/11Qscfgv/36irp7u0J24IF6iPCz5+bC345n6BYoKScA
SW7X7SZfMK/goxJT3UVG/3T7OMS7ZCJeLsAx/GPAgS18KiZ jiRNvWi2grL6yZzmp
7ZI9XXD711UU4KbdJbi65X65axUOKgBorC8aaoo+22kCbSXVDmrprYoJUOkAbc2J
pjP64 j+/9Lz1WA6791/Wmtt9xd1DBYVMBbDIYhqpax7xQlrN1YgpdwLUI6kK1myA
W2ZOUr0qT2a0rysYvKn0e9d8qaGrtNqc19/Qb1BALg3PaME+oIPkOD1oqI86g/Wt
jQgjfmYvaS7SVnyj0p9 jEE2XiK4pAAAAgQD/4rESiCcyDm1ICJh0c jnAC3iSuZLy
bg4gMZZEVU6Lo15GasfreHb2EEa7W1bsNTReBFiq+pDAqh17v3Er jxhYaBTMsg9u
bo4j71Pm2S8Bd4c1LV71NFw2B7bqtfV1tNUYCCNrcc j1V0nRr1gypC6qmgcD/u8p
q0JgaZyRdAx2nwAAAIEA23U1IrioUjg9gIq0u7Ii1S5yi7i0axCWByR0PiW1I000
11pYBgpQ02yk969kkYN2L3YESVeFKsSraBL12kb/Hqpi8cyfLPWEvIbyqUocFX3e
1c4HdyBFmUGRMr5GfhFa4Zh4OXfgYVySn4aFauLwC/MKqAmMUEA456i1tZrcz j0A
AACAds1Am6AB3EuomHS50VA0 j/11mUFCUYvQ89wJWtYJTwPTh3M8zz0mC41+YbRq
pPhriE8JyV86vPvUsp7pWS9VyoxioT309pLGyE+m1Wo5avfqY54doLnCiBLdTxXe
7TqY3As+ohpE+chD8UCJEMfCF4CQ/G0L8o6UNNYXFiKgW jk=
Private-MAC: 86abfdef11715e764427eff2a62554b9697682aa
```

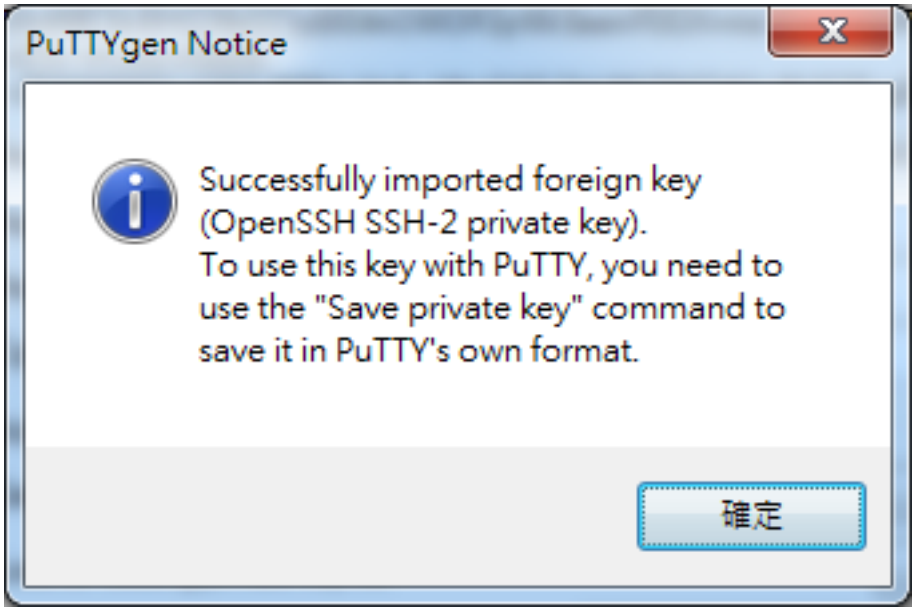
- 這時再次開啓Puttygen程式，直接點選『Load』按鈕，將會開起一個檔案讀取視窗



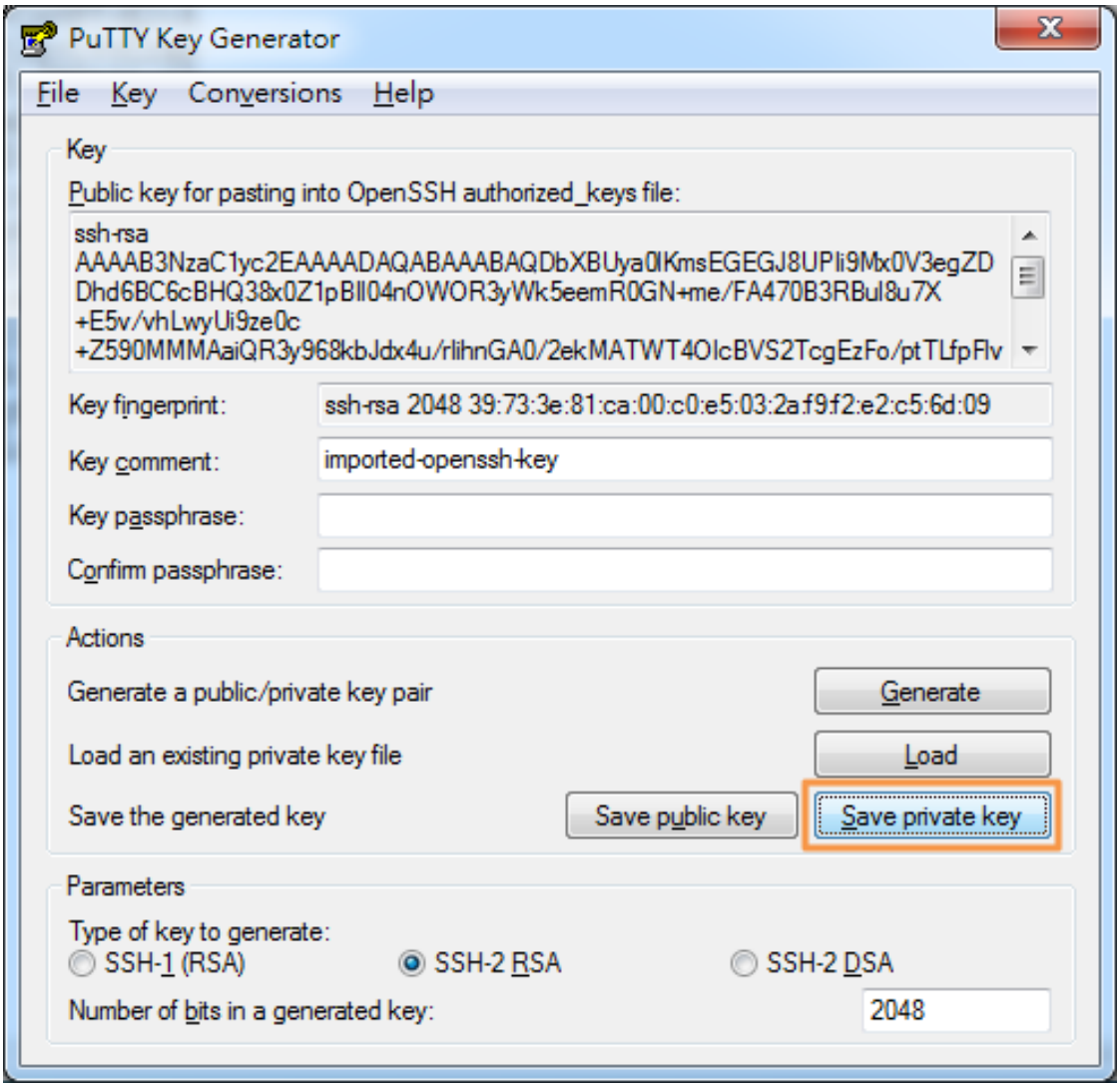
- 首先選取存放ssh-key的資料夾，此時可能會發現什麼檔案都沒有，接著變更下方的副檔名篩選欄，將預設的『PuTTY Private Key Files』變更設定爲『All Files』，就會出現檔案，選取要載入的私鑰後點選下方的『開啓』按鈕



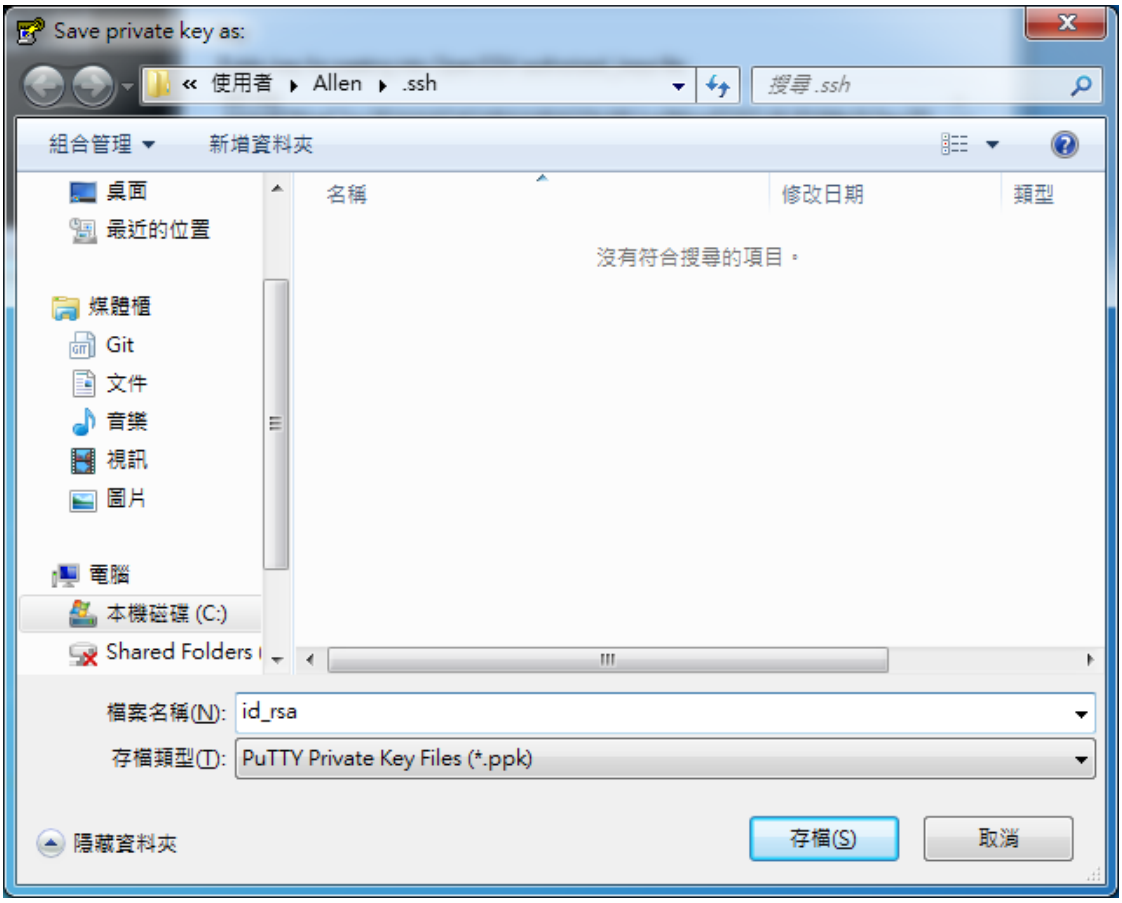
- 開啓私鑰檔案後Pageant將會自動判斷到此檔爲OpenSSH的格式，也有教學該如何轉換到Putty的格式



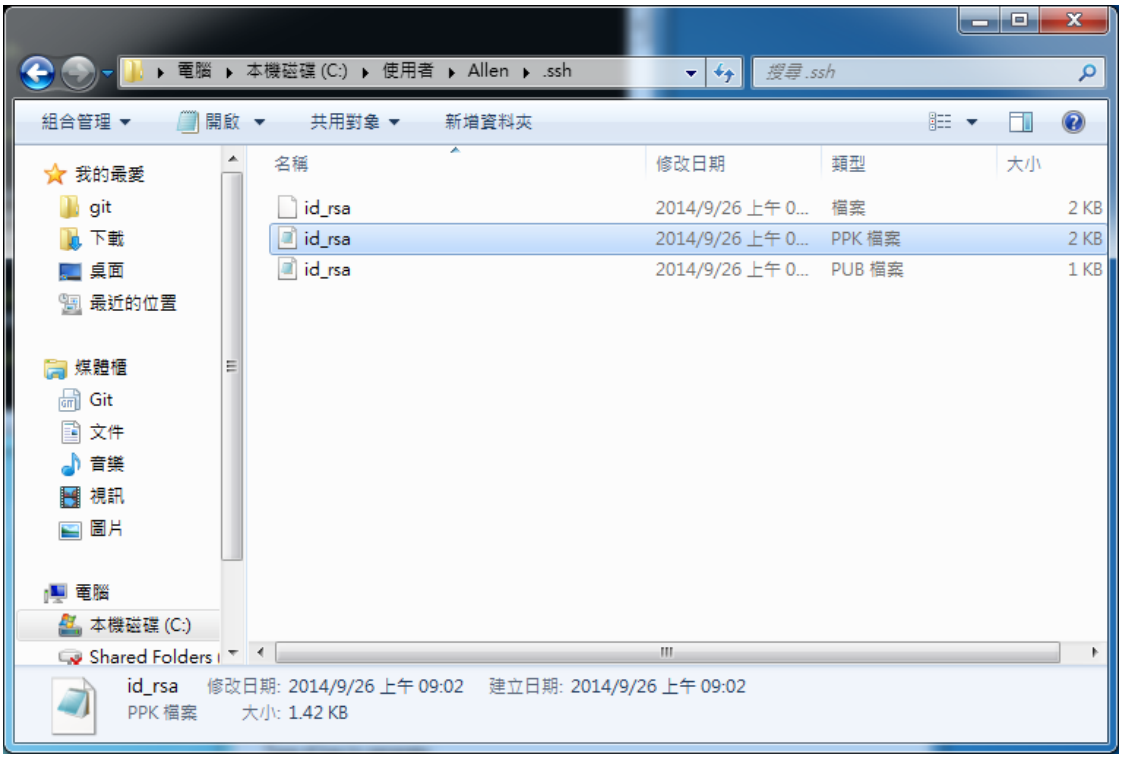
- 點選『確認』關閉提示視窗後將會出現載入私鑰後的內容，依提示視窗所給的訊息，執行『Save private key』按鈕來儲存私鑰



- 點選『Save private key』按鈕後會跳出存儲視窗，先選擇要存放的資料夾，接著在檔案名稱的地方輸入檔名，副檔名爲『ppk』，最後點選存檔即可



- 最後打開檔案管理員，找到存放ssh-key的資料夾，確認是否有完成存檔



5.3 上傳公鑰(Public Key)

不論是用 Puttygen建立ssh-key. 或是 Git Bash建立ssh-key. 將私鑰留在要操作的電腦上，只要將公鑰，副檔名爲pub的檔案寄給管理人員，請管理人員將公鑰放到伺服器中設定後供使用者連線使用，檔案名稱格式參照 公鑰檔案名稱格式.

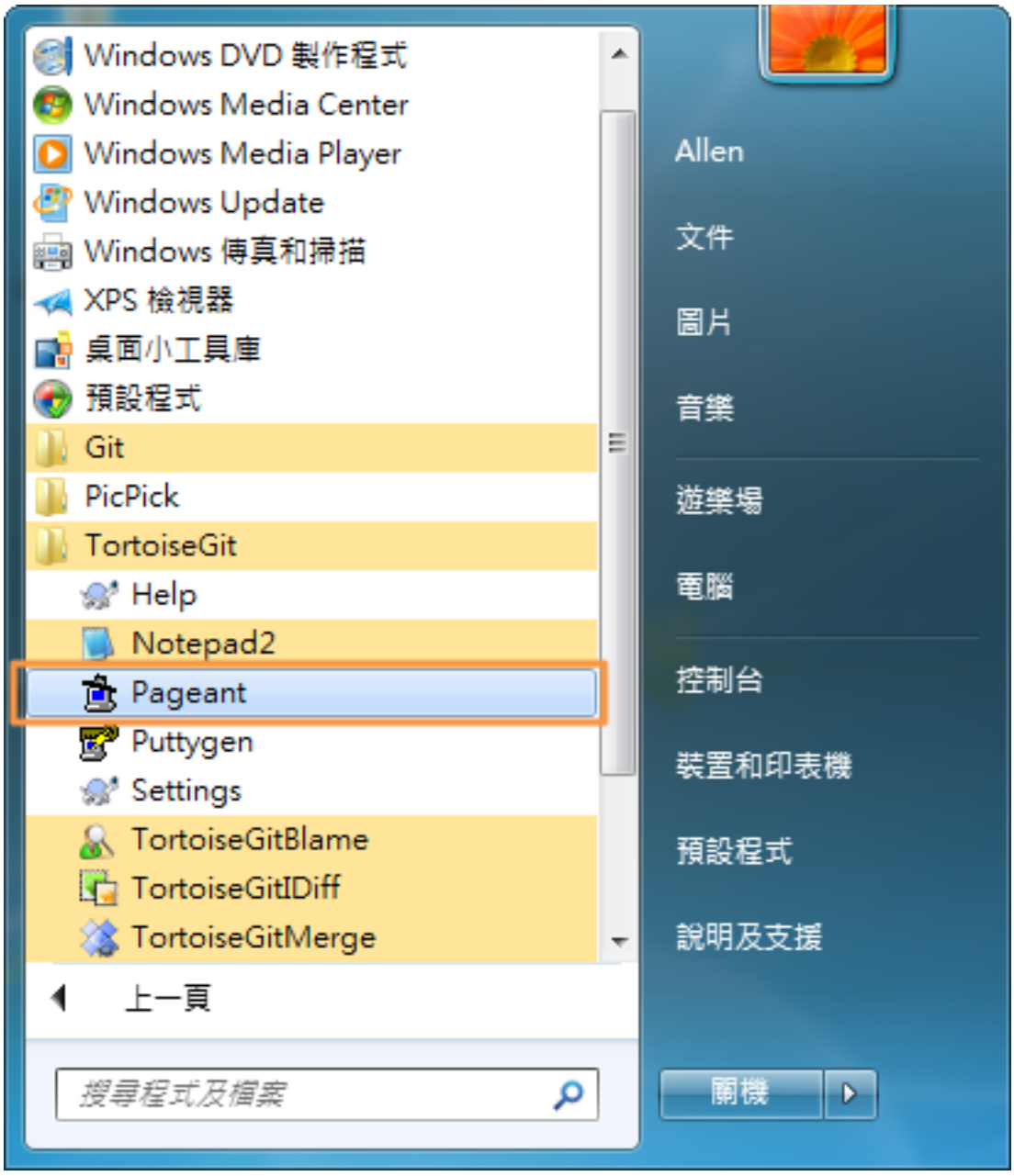
5.3.1 公鑰檔案名稱格式

使用者英文名字@單位名稱.pub

5.4 存取檔案庫(Repo)

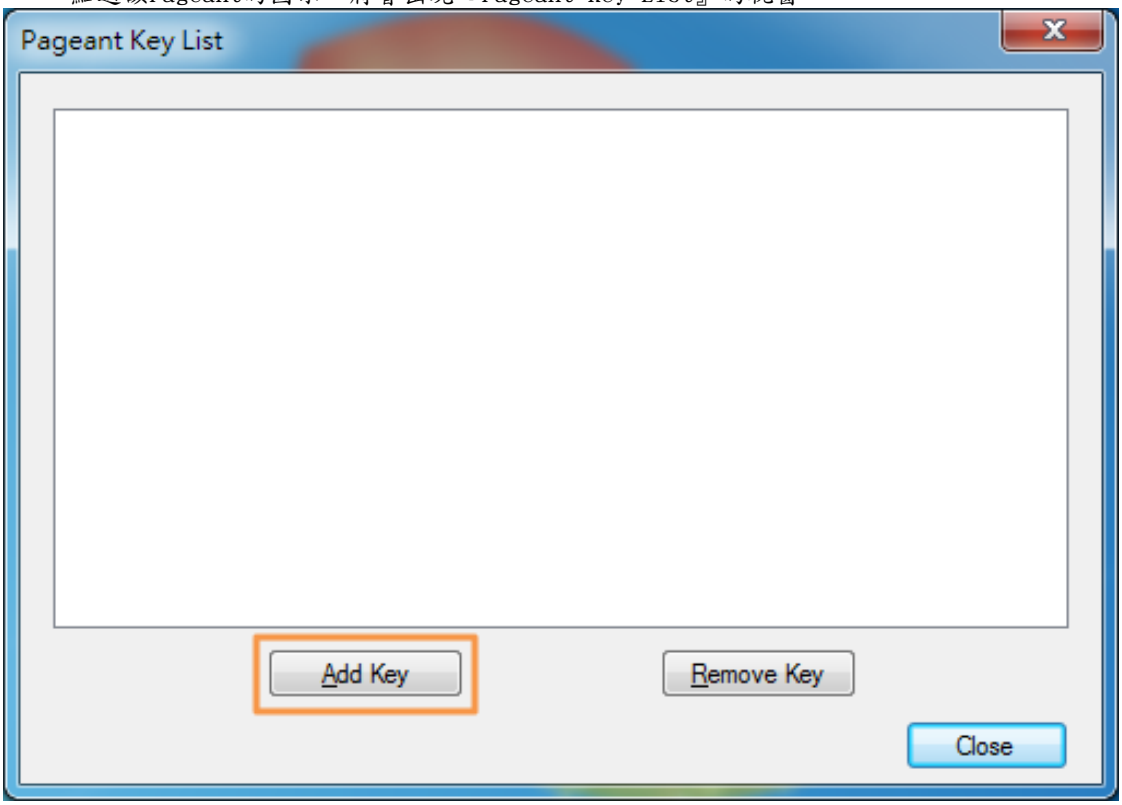
5.4.1 Pageant自動對應連線

- 啓動Pageant，打開『開始程式集』，找到 TortoiseGit 點選打開後就會看到如下圖的內容，點選執行『Pageant』

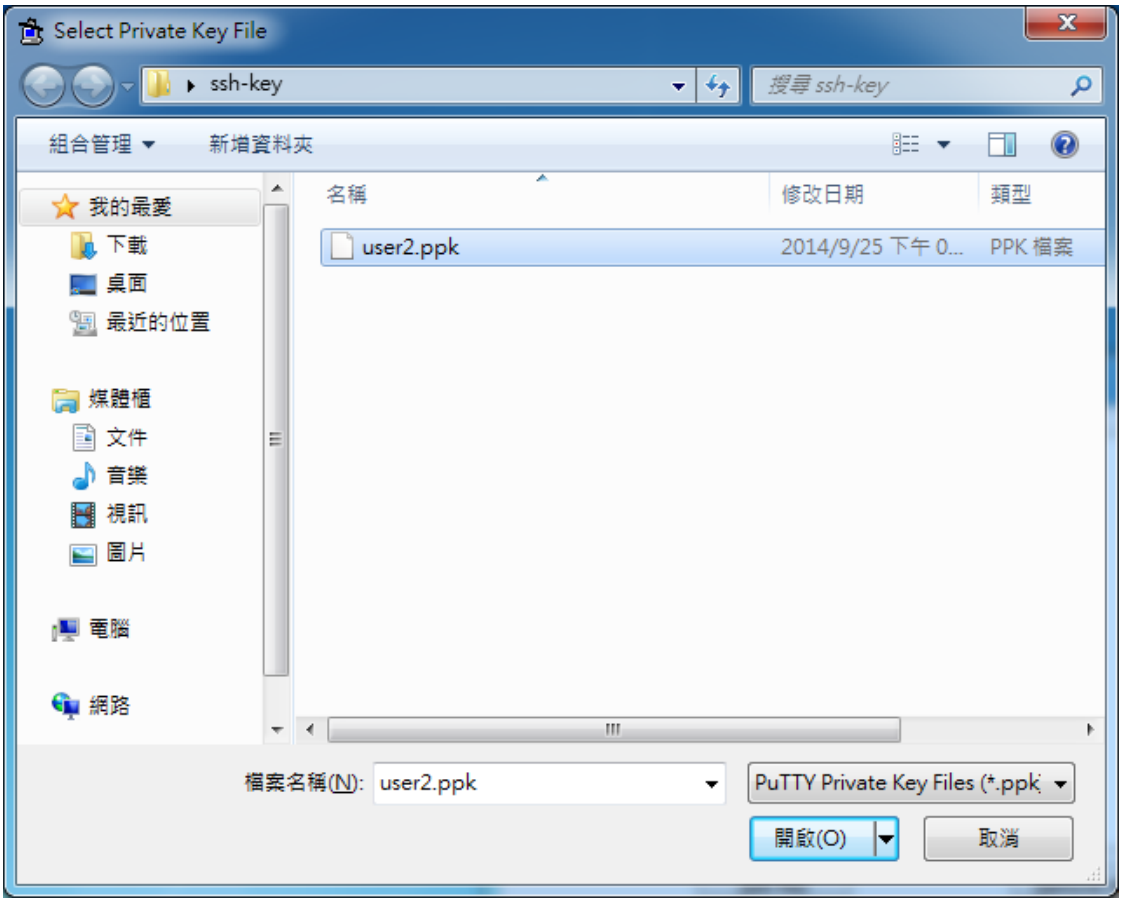


- 執行後可以在工作列右下角的找到如下圖的通知區域中是否有Pageant在運作中

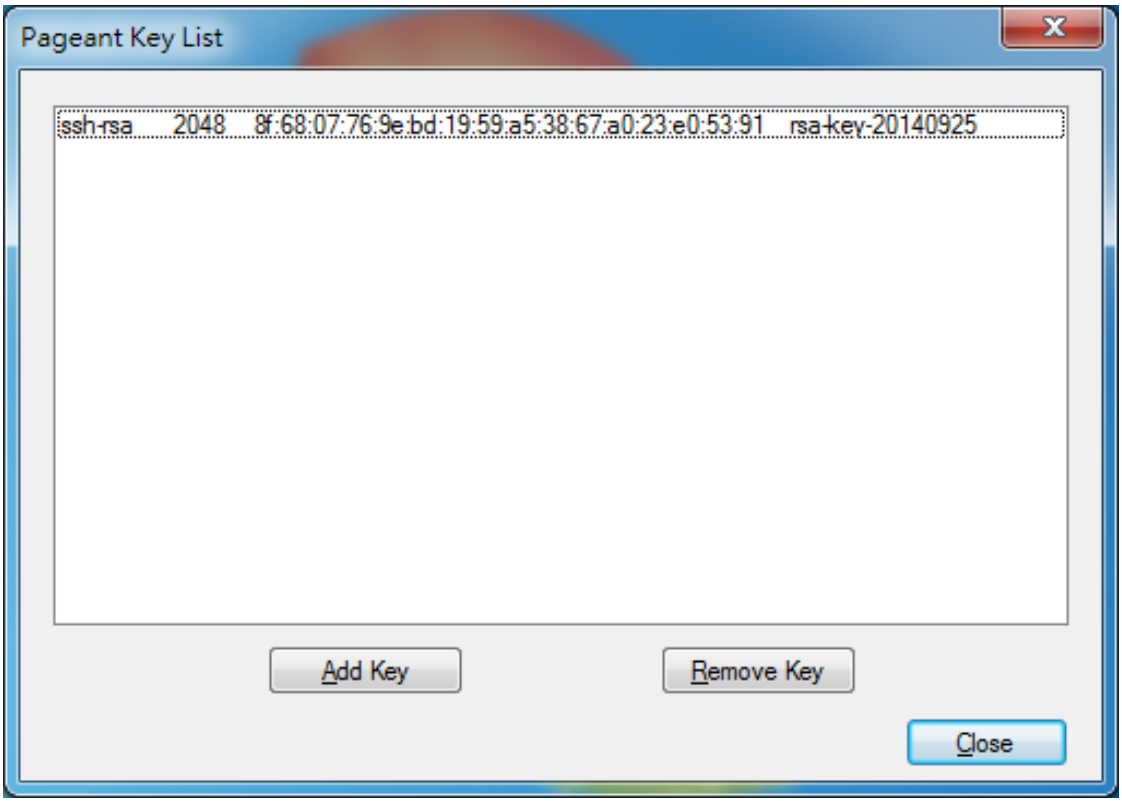
- 點選該Pageant的圖示，將會出現『Pageant Key List』的視窗



- 點選『Add Key』後將會出現『Select Private Key File』的視窗，請找到存放私鑰的資料夾，選取副檔名爲ppk的私鑰檔案，點選開啓

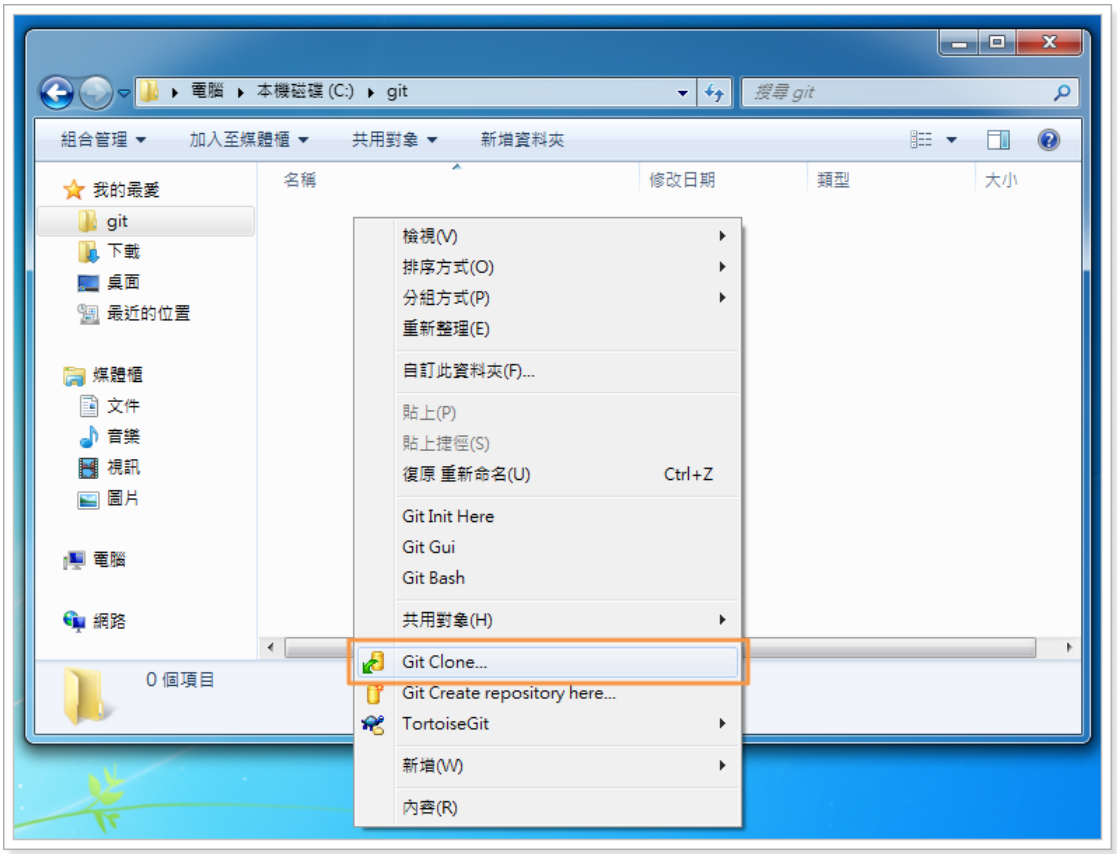


- 這樣就完成載入，如果電腦有需要連接不同的伺服器，可以重複上面的動作，加入N個私鑰，Pageant將會自動對應



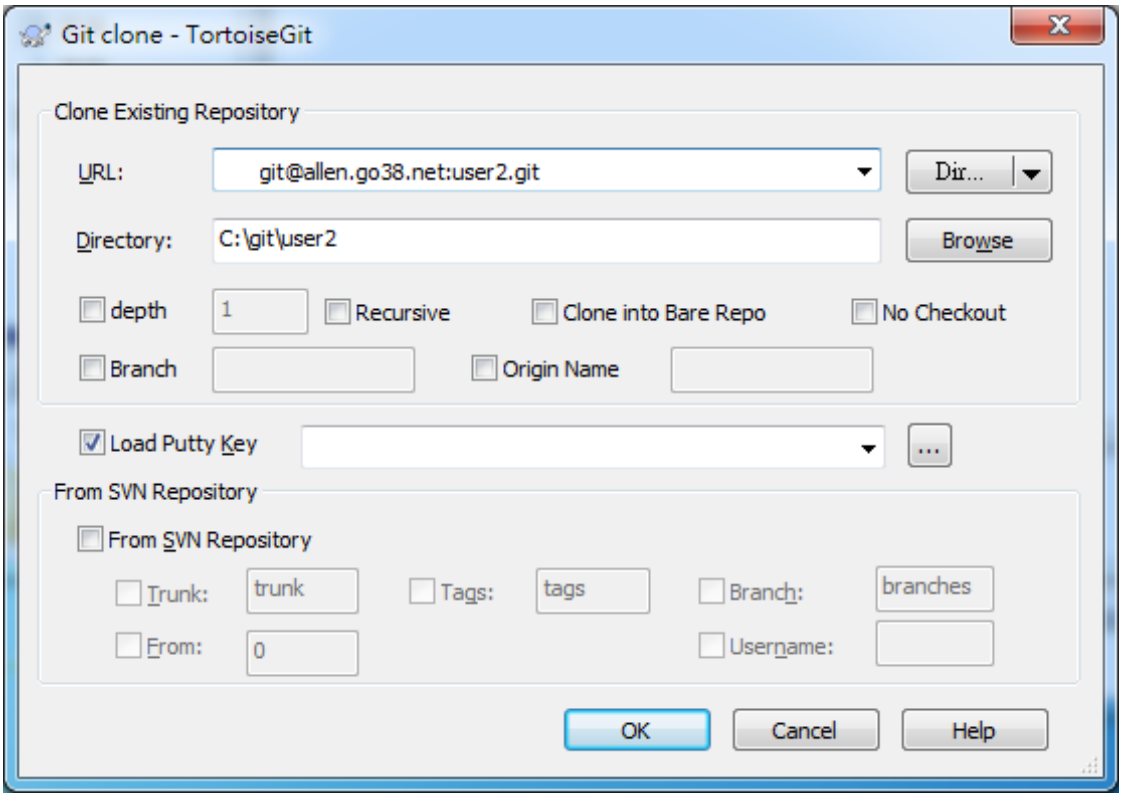
5.4.2 使用TortoiseGit Clone Repo

- 當使用者的ssh-key建立好，伺服器上的設定也完成後，接下來就是將檔案庫(Repo)從伺服器上複製(clone)到操作電腦上，使用TortoiseGit軟體的圖形介面(GUI)去操作控制檔案庫(Repo)，道先用開起檔案管理員，找到要建立檔案庫(Repo)的資料夾，然後在資料夾空白區塊點選滑鼠右鍵，就會有TortoiseGit的功能列選項可以使用，因爲第一次存取檔案庫(Repo)，所以要使用clone(中文版爲：克隆)複製到本機操作電腦中。

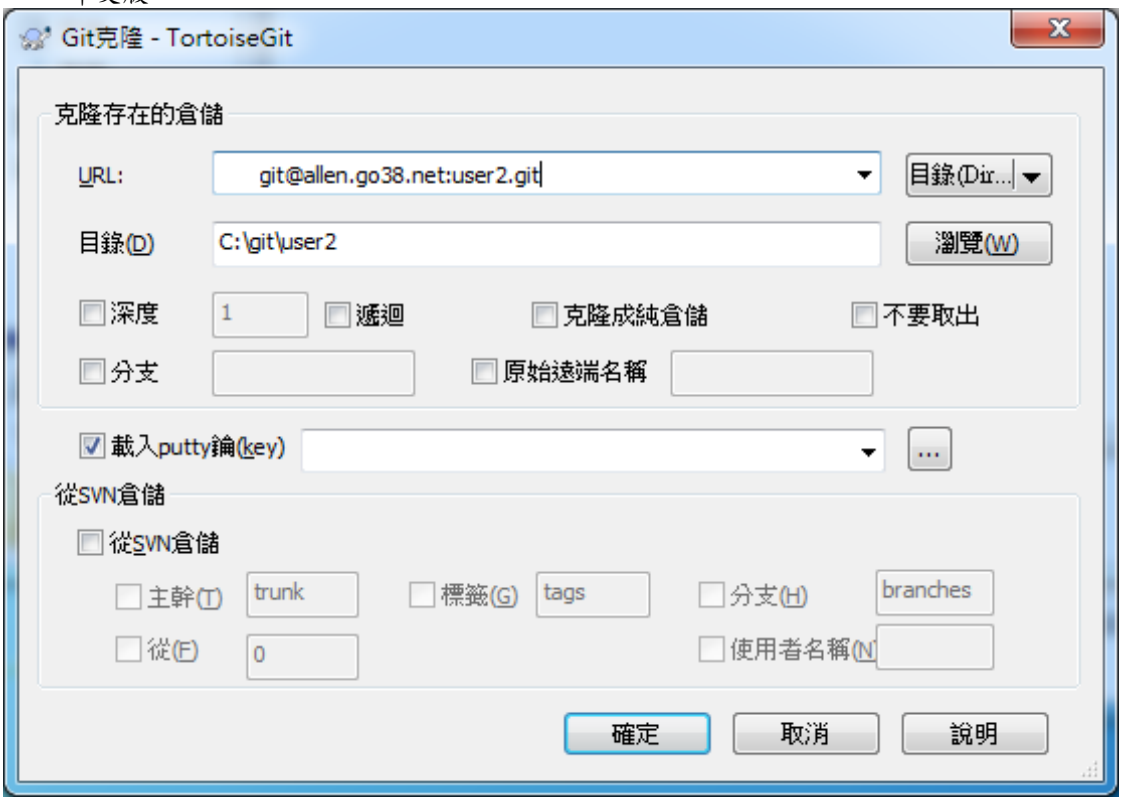


- 點選右鍵功能列的『Git Clone...』或『Git 克隆...』後或出現設定視窗，在這邊最主要就是『URL』，目前的URL格式爲『ssh://user@server/project.git』或『user@server:project.git』，依URL格式轉換到目前伺服器的URL爲『ssh:// git@allen.go38.net/ user2.git』或『git@allen.go38.net:user2.git』，所以在視窗中的URL輸入正確的內容後，下方的目錄欄位會自動帶入project的名稱，此範例爲user2，最後點選『確定』就開始執行複製(clone)

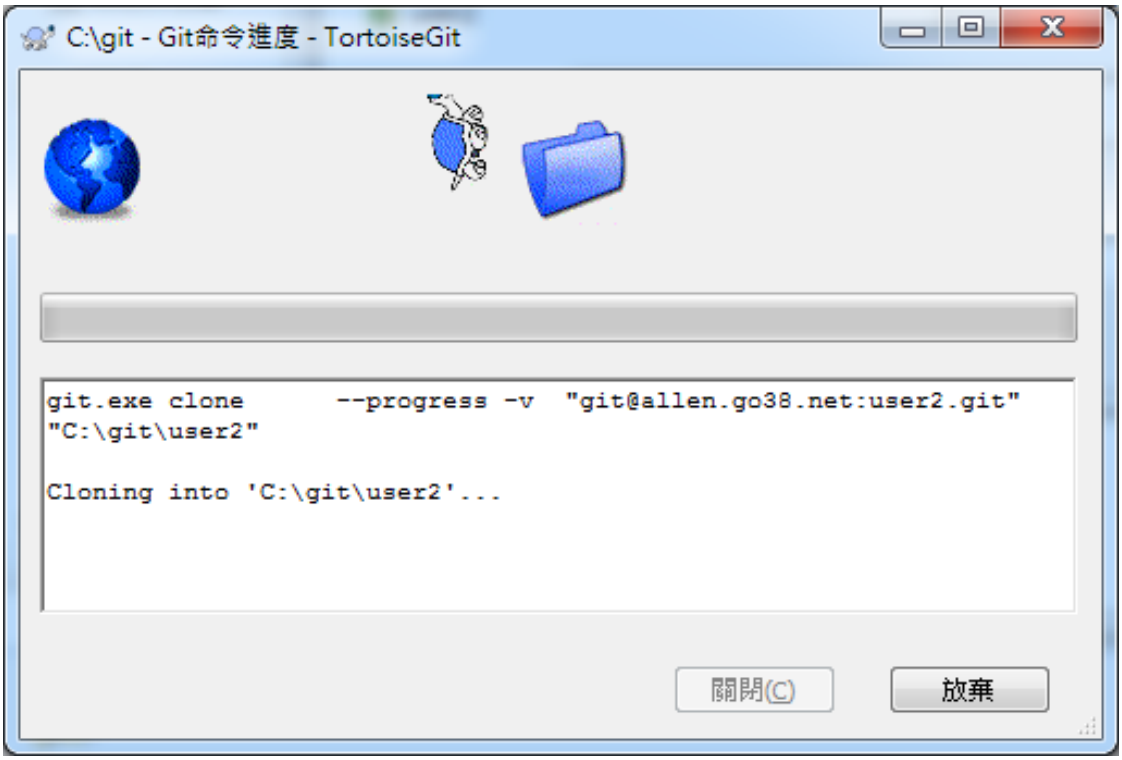
  英文版



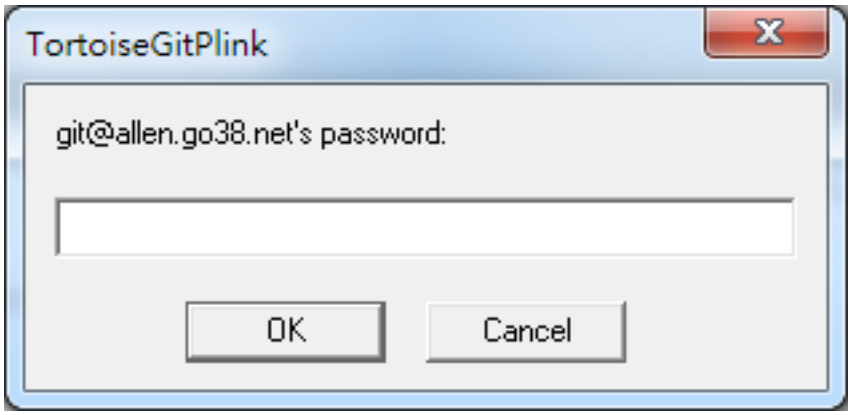
中文版



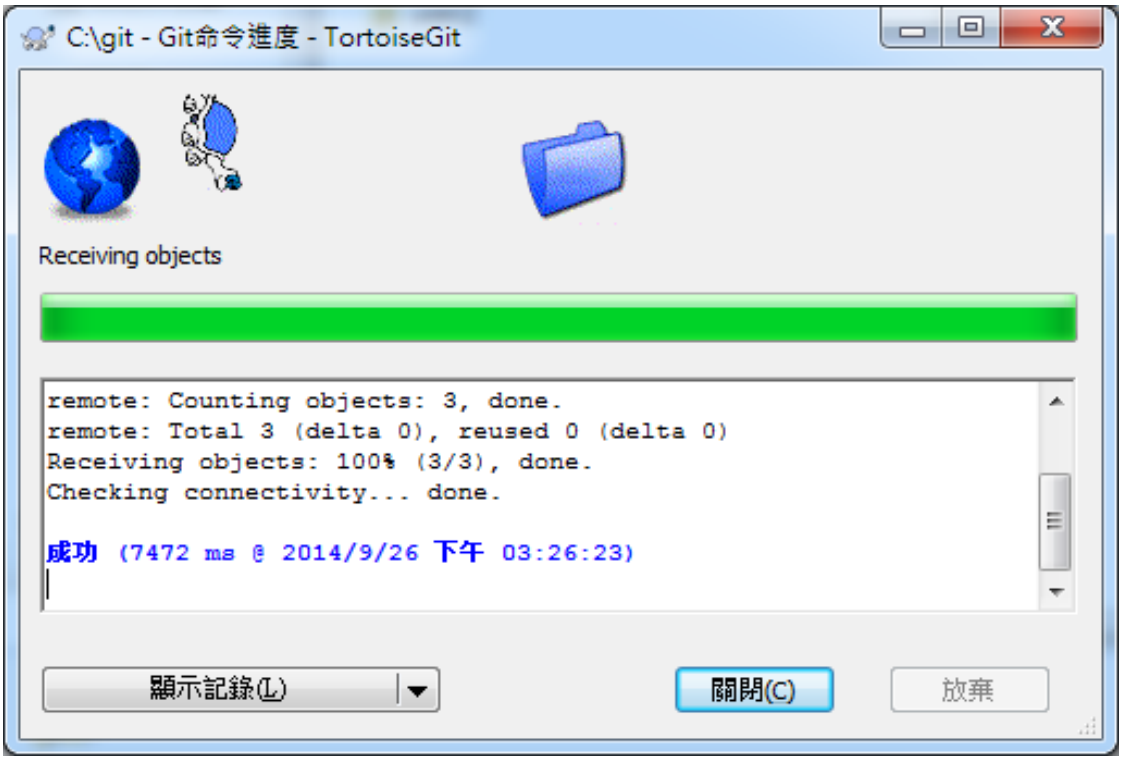
- 下圖爲開始執行Git指令進度的視窗



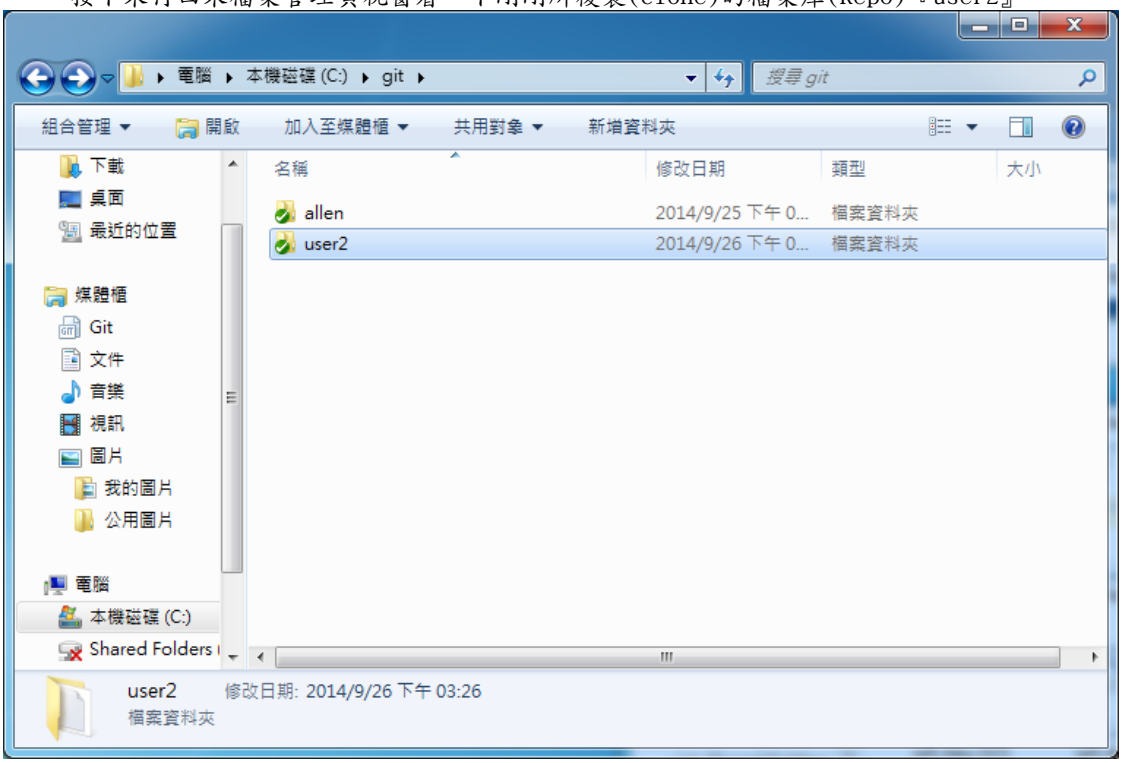
- 在執行Git指令進度視窗的過程中，如果出現下圖的話有可能是Pageant沒有啓動，如果有啓動則要檢查一下私鑰是否有加入



- 下圖爲Git指令執行成功的畫面

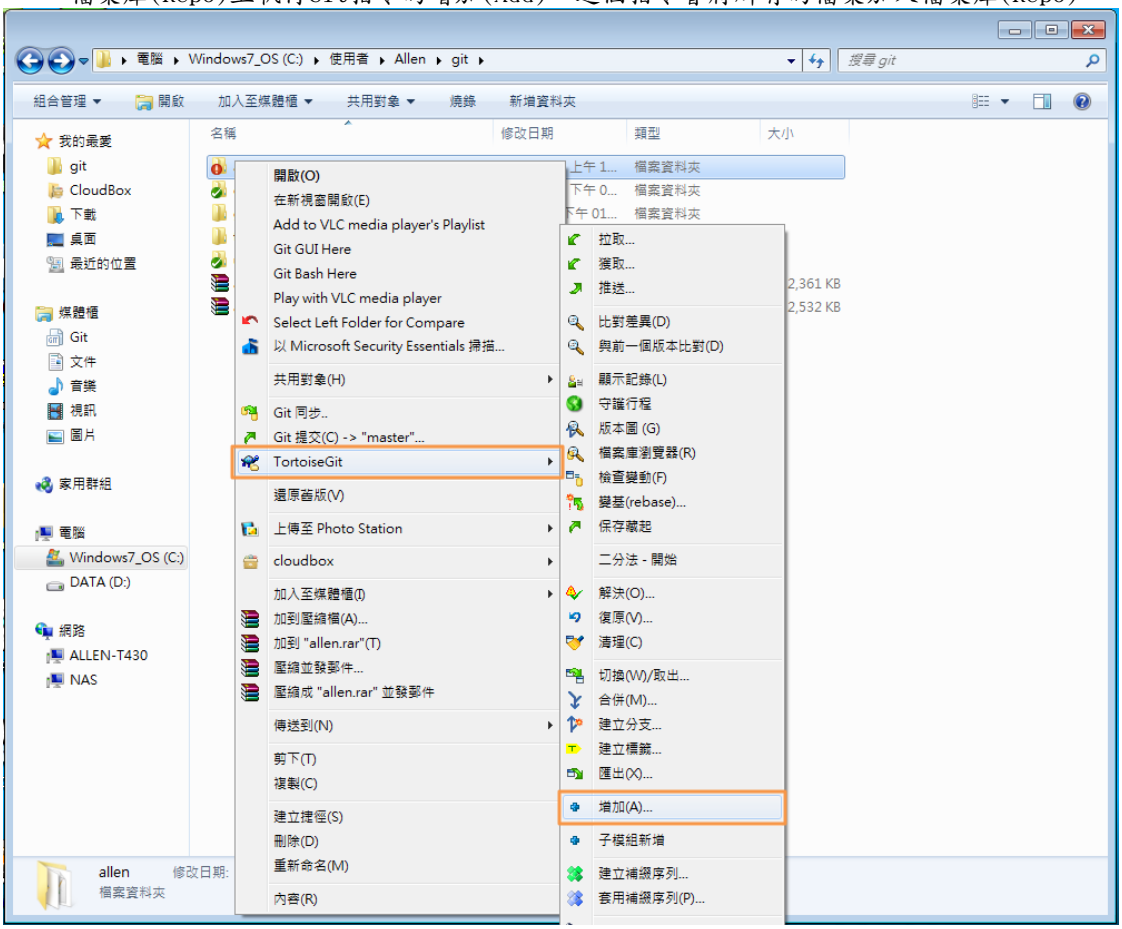


- 接下來再回來檔案管理員視窗看一下剛剛所複製(clone)的檔案庫(Repo)『user2』

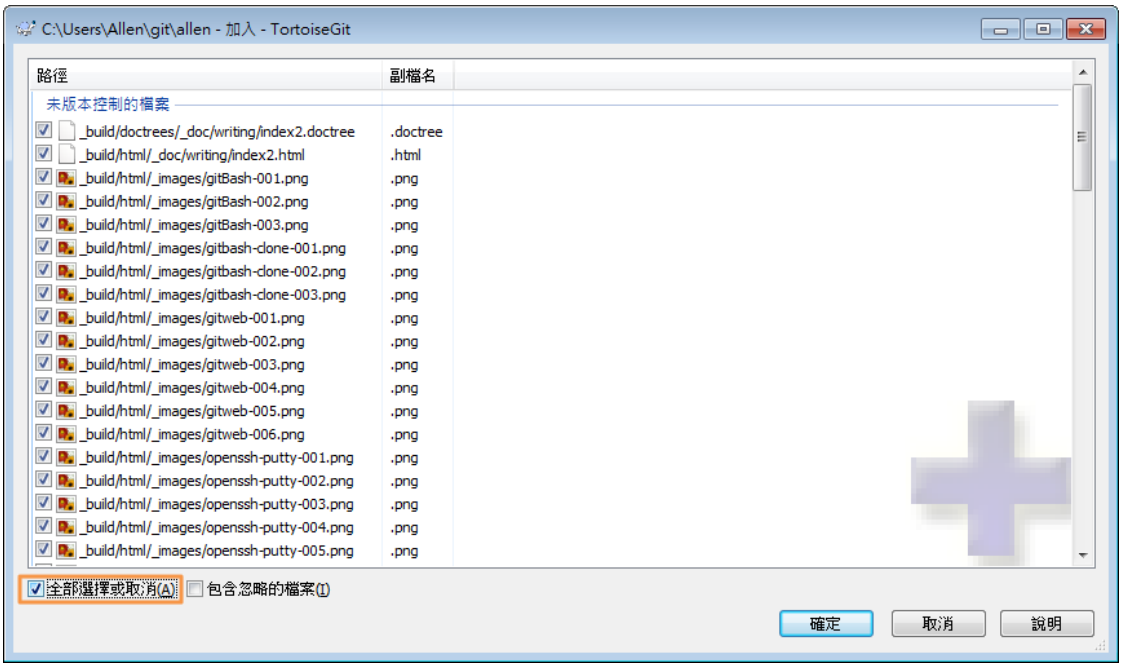


5.4.3 使用TortoiseGit Push Repo

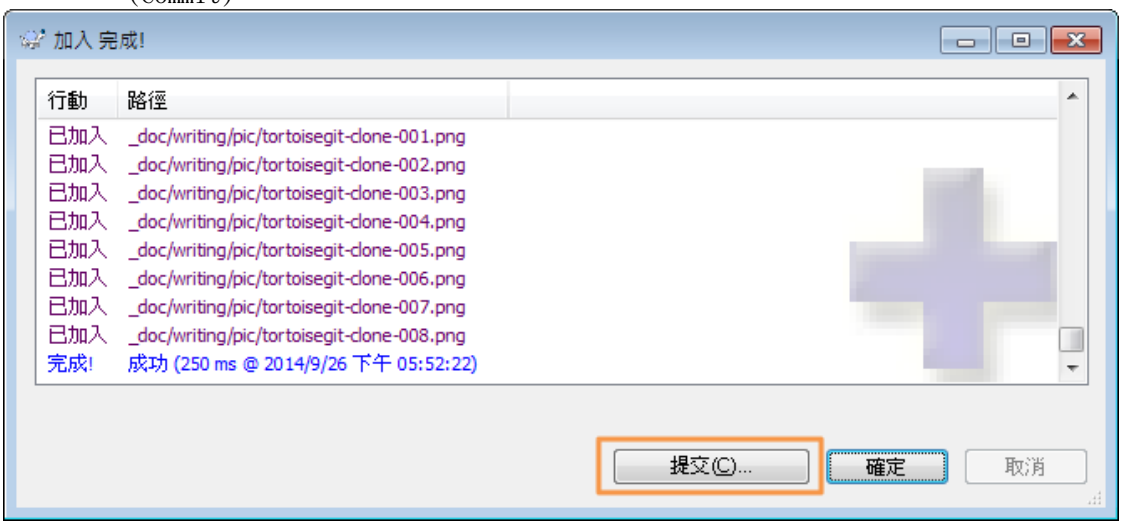
- 當文章內容修改完畢，要將檔案上傳推送(Push)到伺服器的檔案庫(Repo)，只要在本機的檔案庫(Repo)資料夾中點選滑鼠右鍵的功能列中操作即可，在這個地方要注意一下，如果是對檔案庫(Repo)的資料夾上點選滑鼠右鍵操作Git指令這種方式是對整個檔案庫(Repo)操作，如果是在檔案庫(Repo)裡面的資料夾或檔案上點選滑鼠右鍵操作Git指令，這樣只會對該資料夾或檔案做處理，下圖就是在檔案庫(Repo)上執行Git指令的增加(Add)，這個指令會將所有的檔案加入檔案庫(Repo)



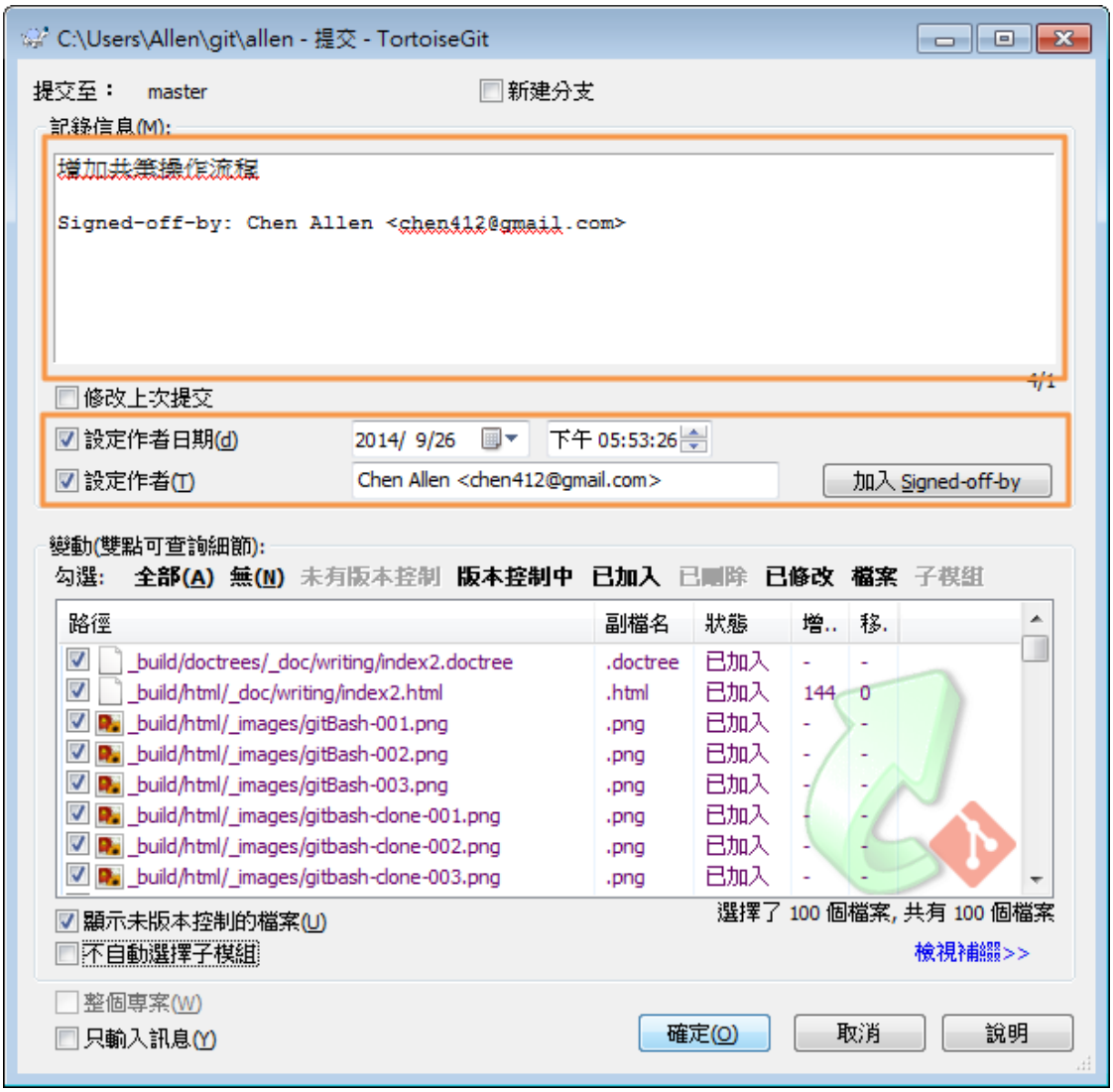
- 當點選增加(Add)後會出現一個視窗，該視窗內所列出的檔案就是與本機檔案庫(Repo)最後一個版本比對後尚未加入的檔案清單，在這邊可以選擇檔案要不要加入到檔案庫(Repo)中，可以自行勾選，如果檔案有很多想要全選或全部取消勾選，可在下面『全部選擇或取消』，最後點選『確定』



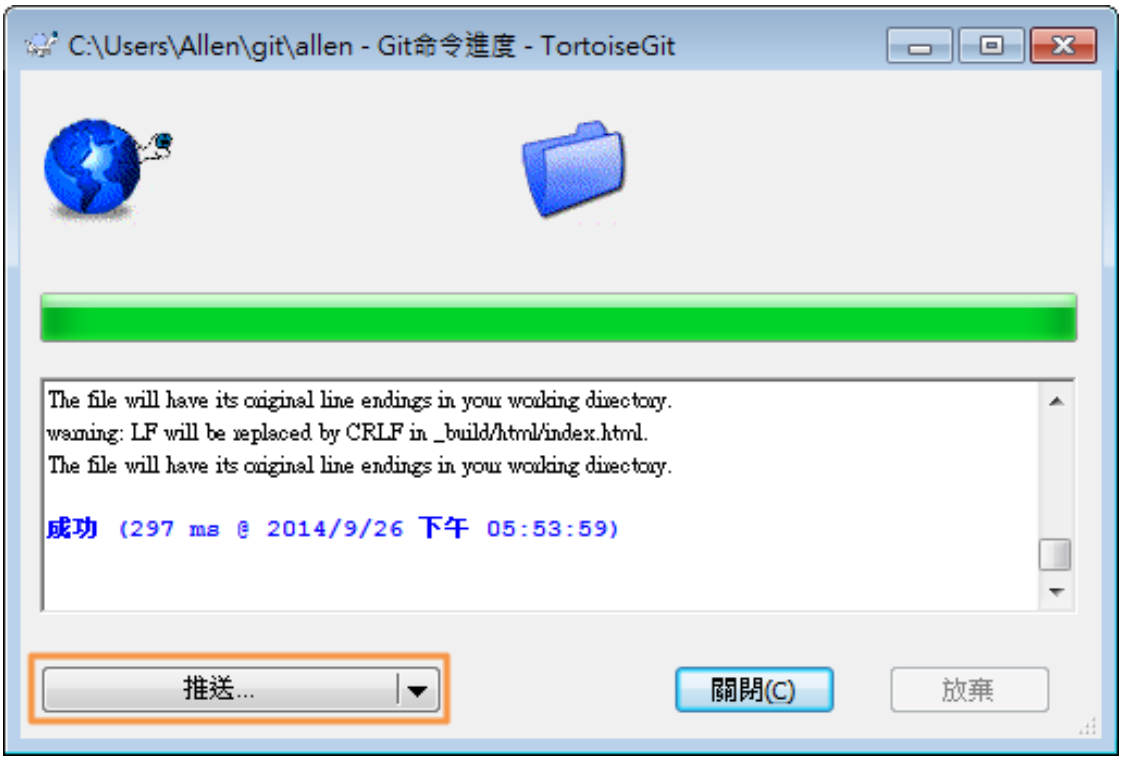
- 點選確定後，就會把所勾選的檔案加入到檔案庫(Repo)中，如果正常的狀況下，會出現所有的檔案都『已加入』，並且在最下面出現『完成』，這個時後並沒有將檔案上傳更新到伺服器，這些操作都還是在本機的檔案庫(Repo)中操作，接著點選如下圖中的提交(Commit)



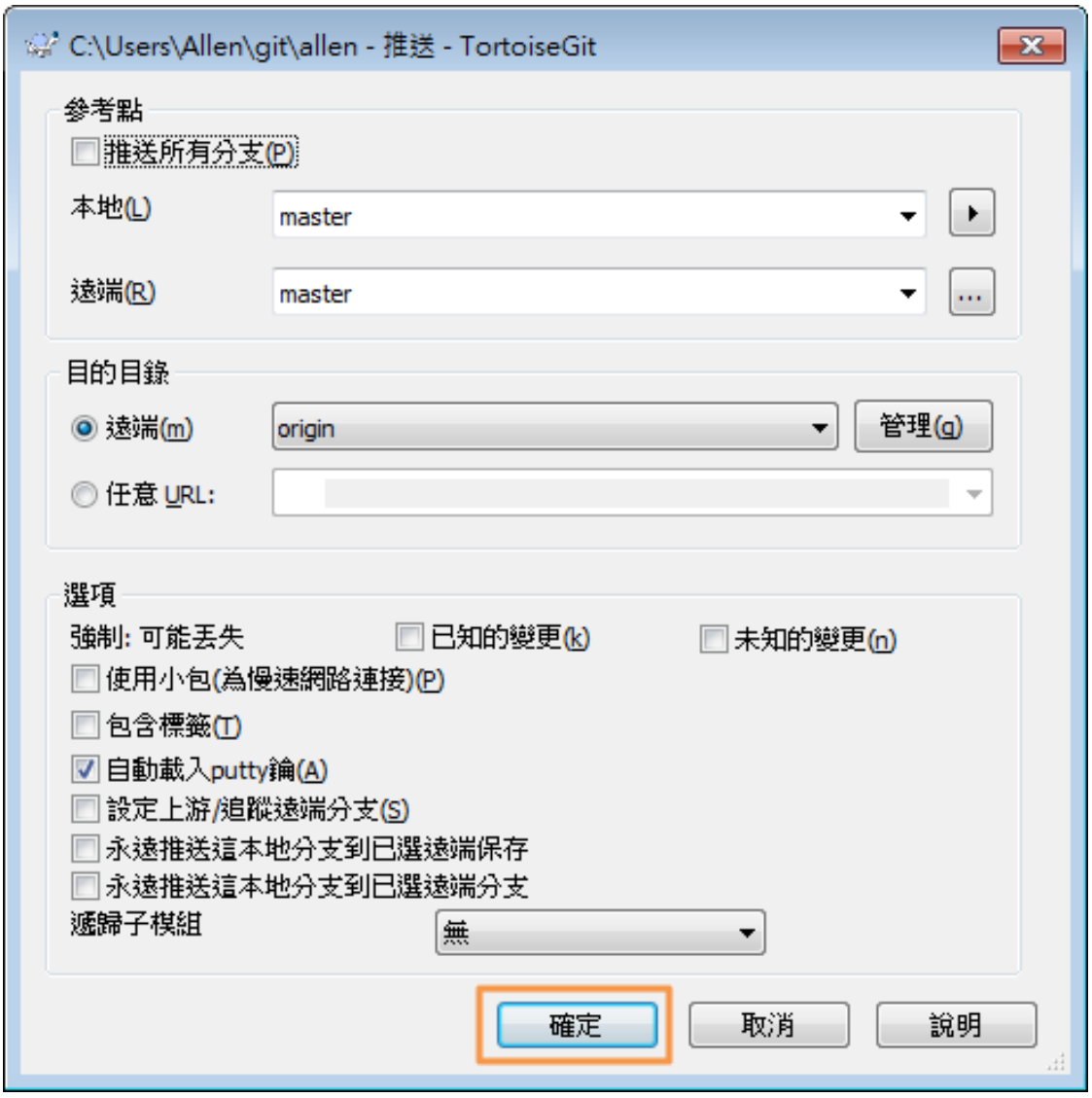
- 接著就會看到提交(Commit)的視窗，在視窗最上方的『記錄信息』填寫這次提交的簡略內容，方便之後尋找並回復，在下方可以勾選設定作者日期、設定作者，點選『加入Signed-off-by』會自動把設定作者的內容自動填到『記錄信息』的最下方，在『變動』可以看到這次提交(Commit)的檔案異動狀況，最後點選『確定』後就會把這次提交(Commit)的內容加入到檔案庫(Repo)



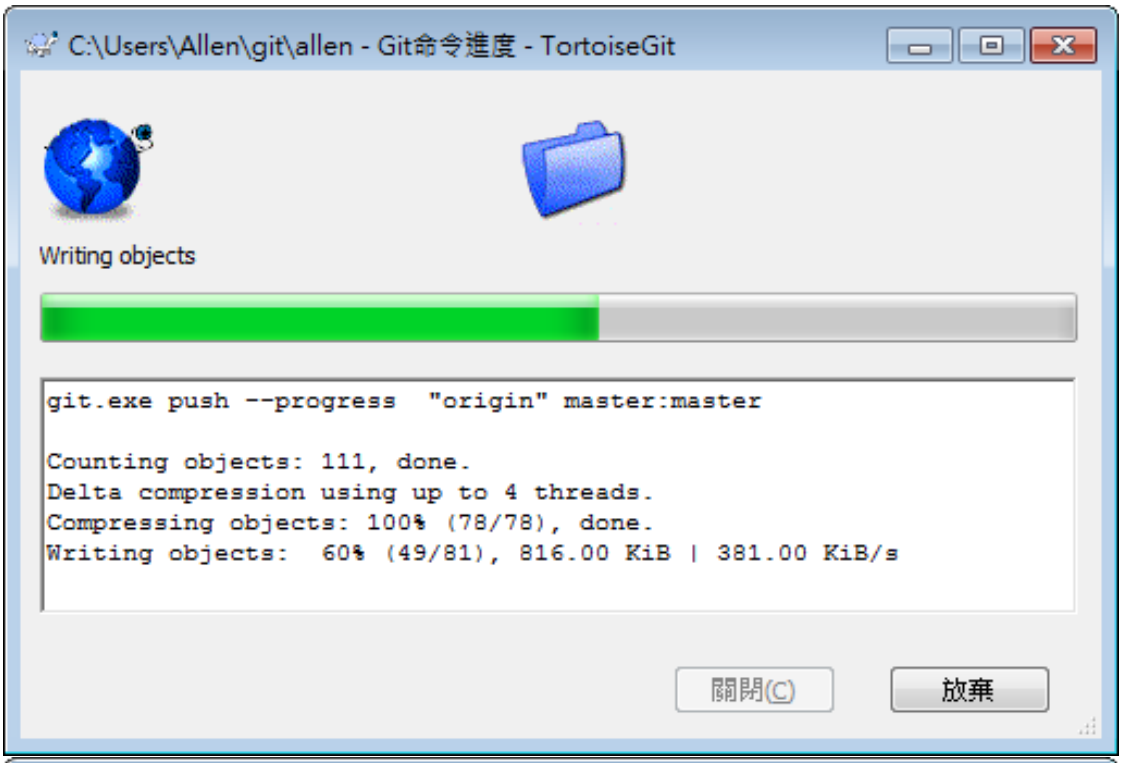
- 點選『確定』後執行Git指令，這時就會出現『Git命令進度』的視窗，當看到『完成』後，本機的檔案庫(Repo)就把這次提交(Commit)的內容增加變更成最新的版本內容，但這時所有的操作還是都在本機的檔案庫(Repo)中操作，尚未上傳到遠端的伺服器中，如果要上傳推送(Push)，請點選畫面中的『推送』

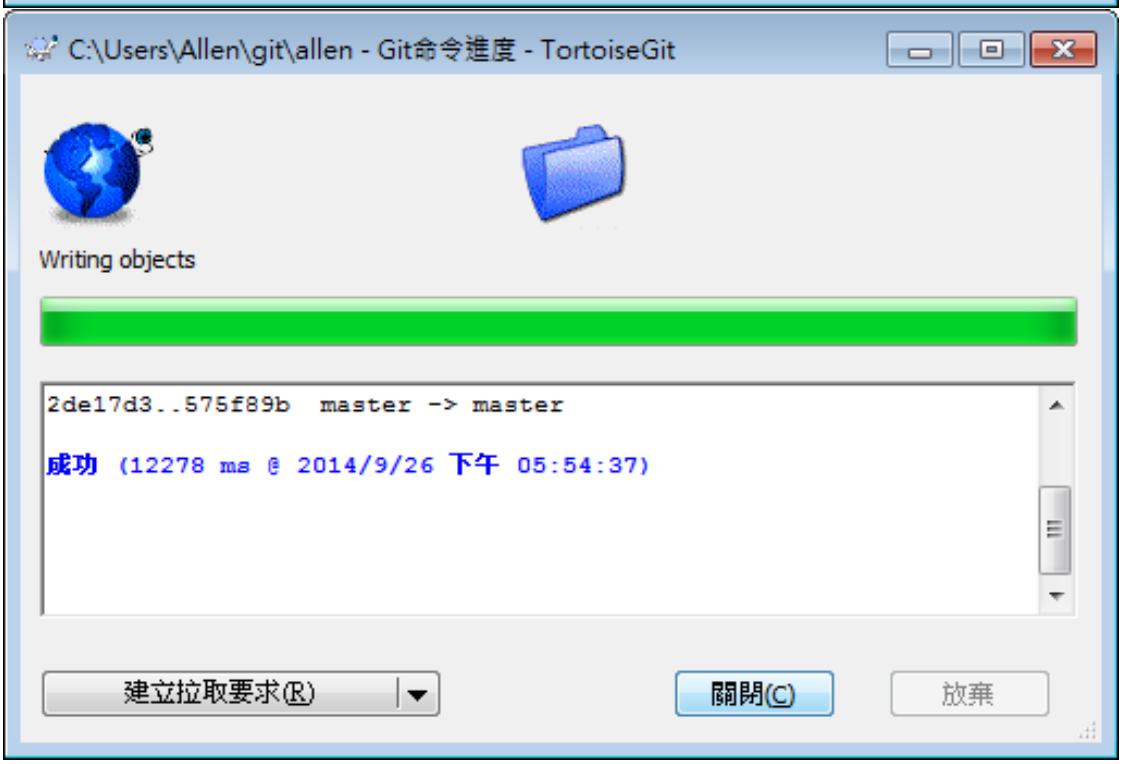


- 點選『推送』後接著出現『推送』的設定視窗，在這邊先使用預設值，點選『確定』，開始將本機的檔案庫(Repo)上傳到遠端伺服器



- 接著又會出現上傳推送(Push)指令進度視窗，此時就會開始上傳到遠端伺服器，最後會出現『完成』，這時就完成整個更新的流程

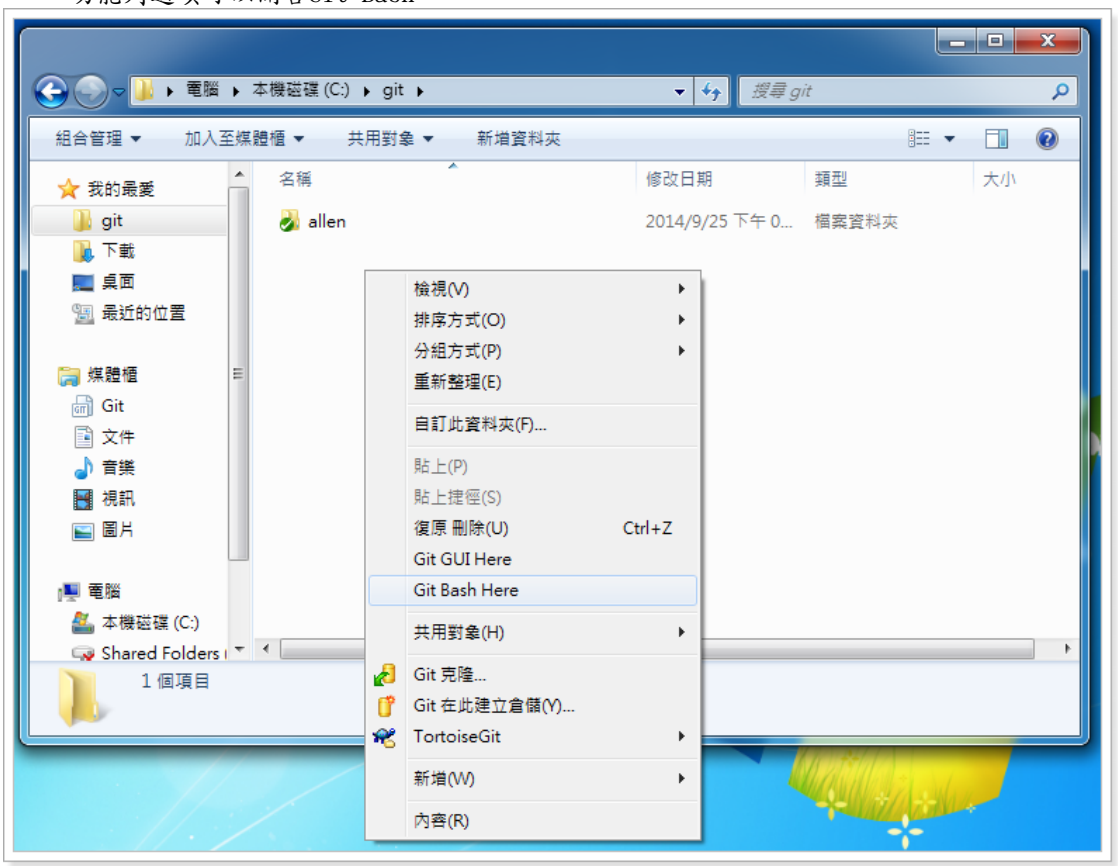




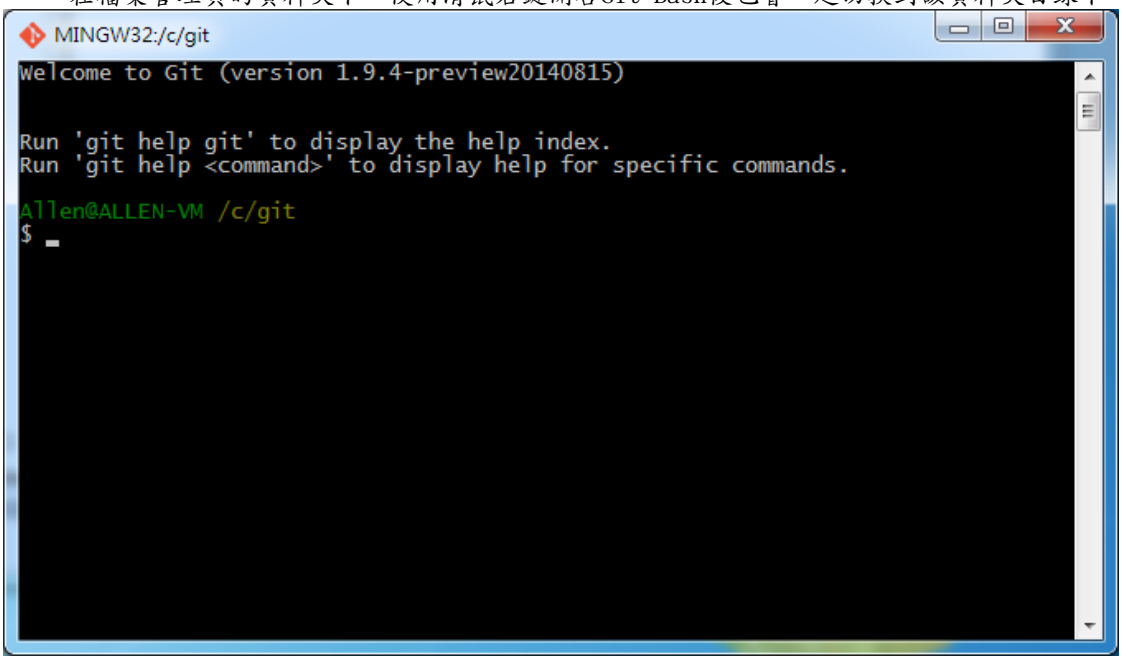
- 這時可以到 gitweb 上看到該檔案庫(Repo)的更新內容

5.4.4 使用Git Bash Clone Repo

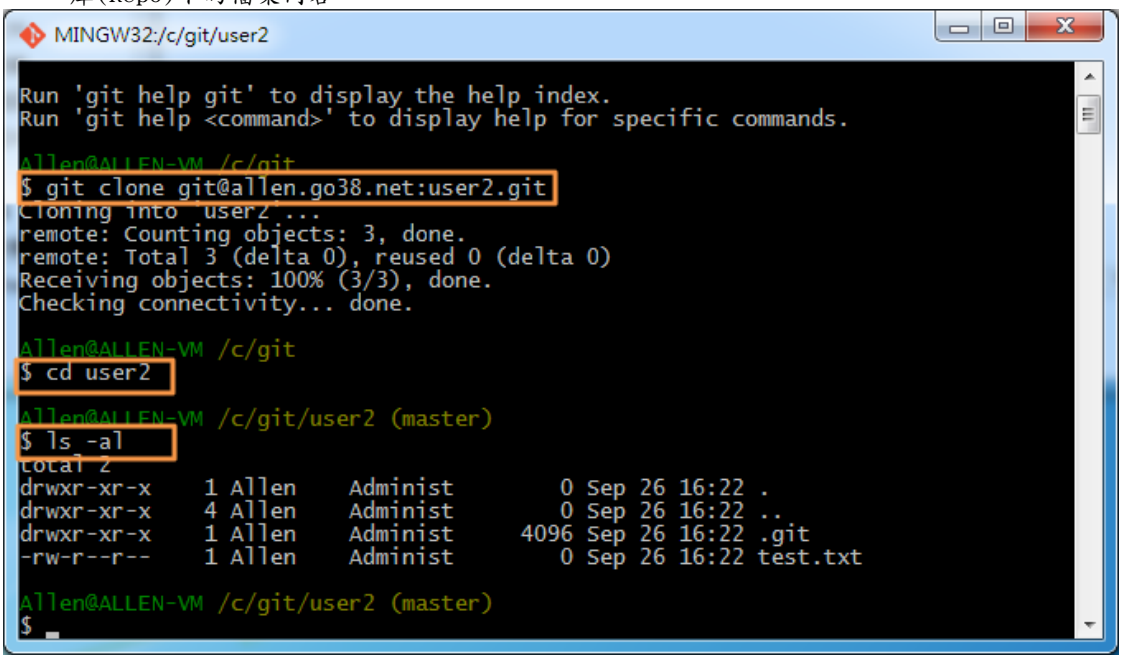
- 當使用者的ssh-key建立好，伺服器上的設定也完成後，接下來就是將檔案庫(Repo)從伺服器上複製(clone)到操作電腦上，使用Git 軟體的文字介面(console)去操作控制檔案庫(Repo)，道先用開起檔案管理員，找到要建立檔案庫(Repo)的資料夾，然後在資料夾空白區塊點選滑鼠右鍵，就會有Git的功能列選項可以開啓Git Bash



- 在檔案管理員的資料夾下，使用滑鼠右鍵開啓Git Bash後也會一起切換到該資料夾目錄下



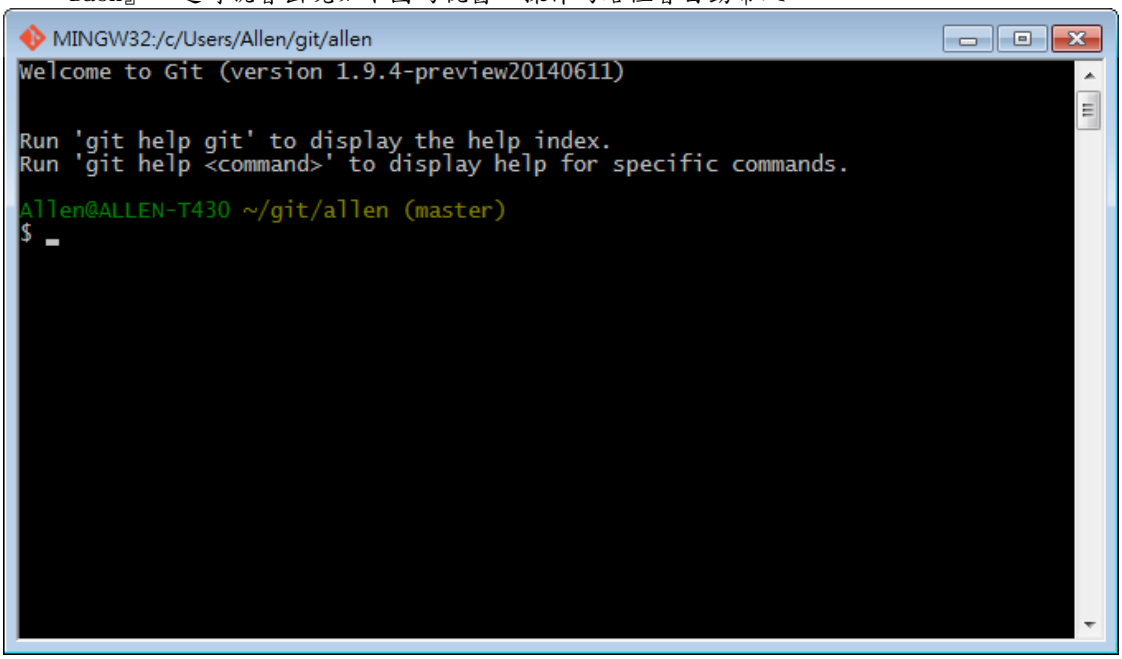
- 接著就輸入git clone指令將要存取的檔案庫(Repo)複製下來，依URL的格式輸入『git clone ssh://git@allen.go38.net/user2.git』或『git clone git@allen.go38.net:user2.git』，正常無誤的話就會看到類似下圖的內容，接著執行『cd user2』切換到檔案庫user2，再使用『ls -al』列出該檔案庫(Repo)下的檔案內容



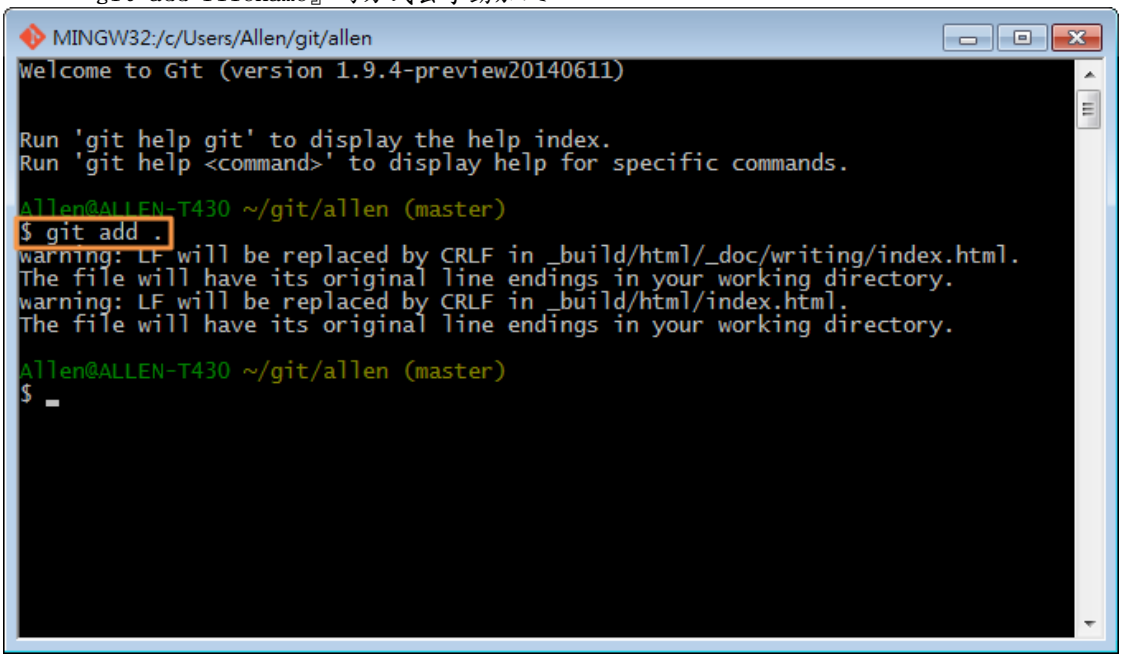
使用Git Bash Push Repo

- 在圖形介面的操作方式有介紹過如何使用TortoiseGit將寫好的文件上傳推送(Push)到伺服器上，現

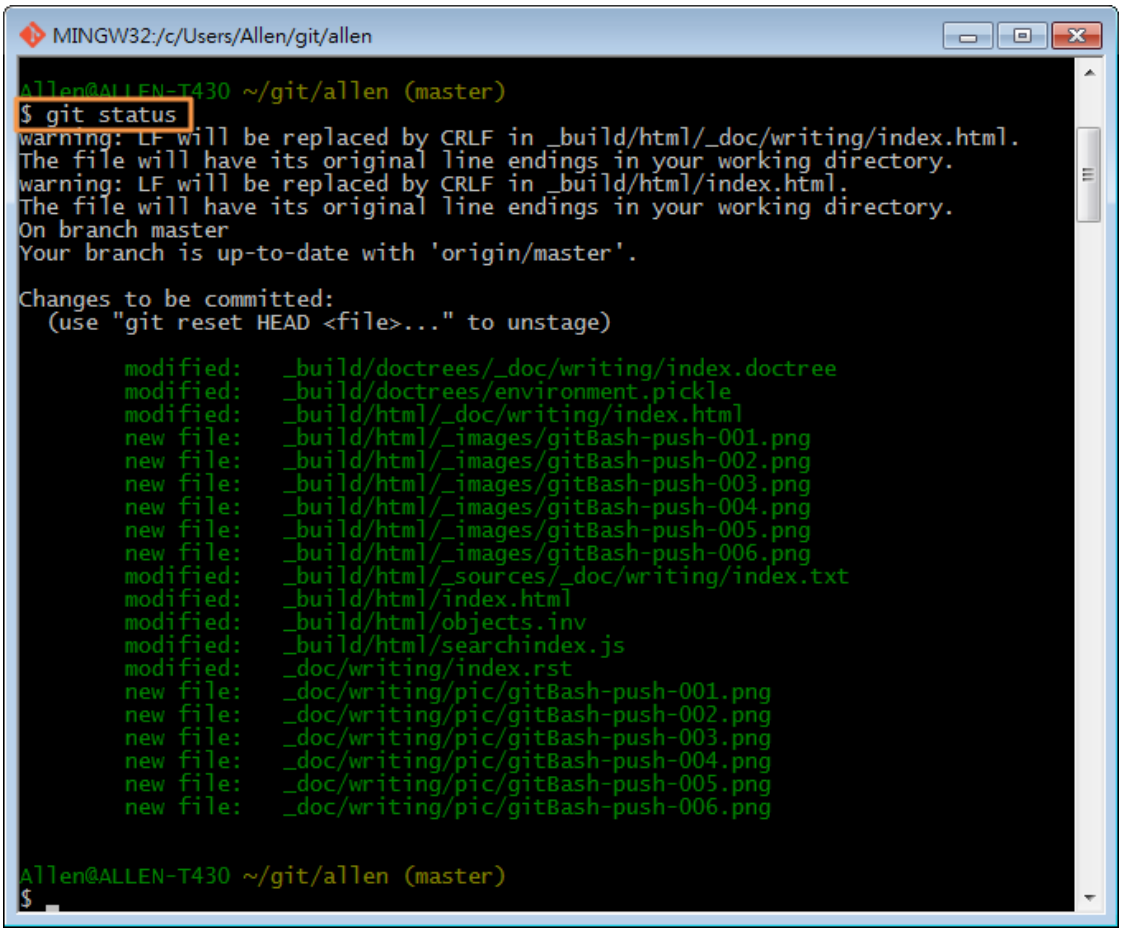
在使用 Git Bash文字指令的方式將寫好的文件上傳推送(Push)到伺服器上，操作的指令主要有三個，和TortoiseGit的操作是相同的；首先打開要處理的檔案庫(Repo)，然後點選滑鼠右鍵，點選『Git Bash』，這時就會出現如下圖的視窗，操作的路徑會自動帶入



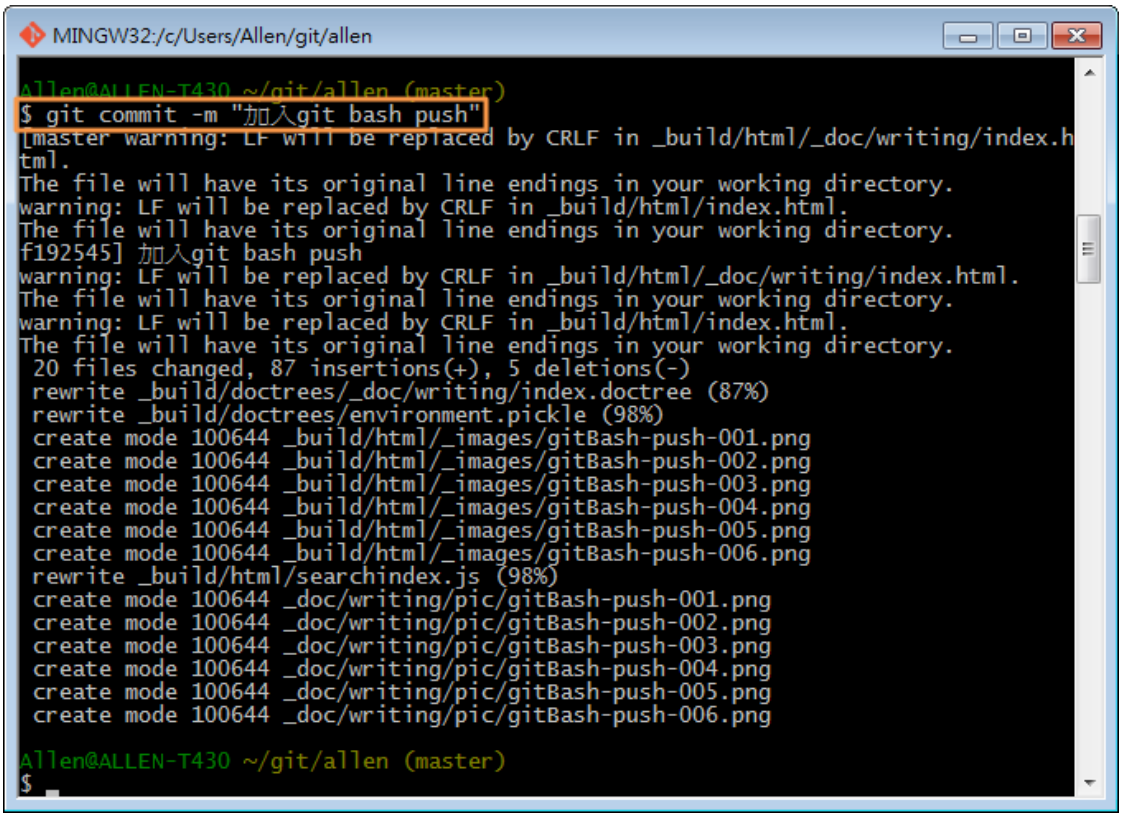
- 操作如同TortoiseGit的『增加』，在Git下就是『git add .』後面的『.』就是全部都加入的意思，在在TortoiseGit會有好看的圖形介面可以使用勾選的方式選擇要不要加入，如果在Git Bash下只能用『git add filename』的方式去手動加入



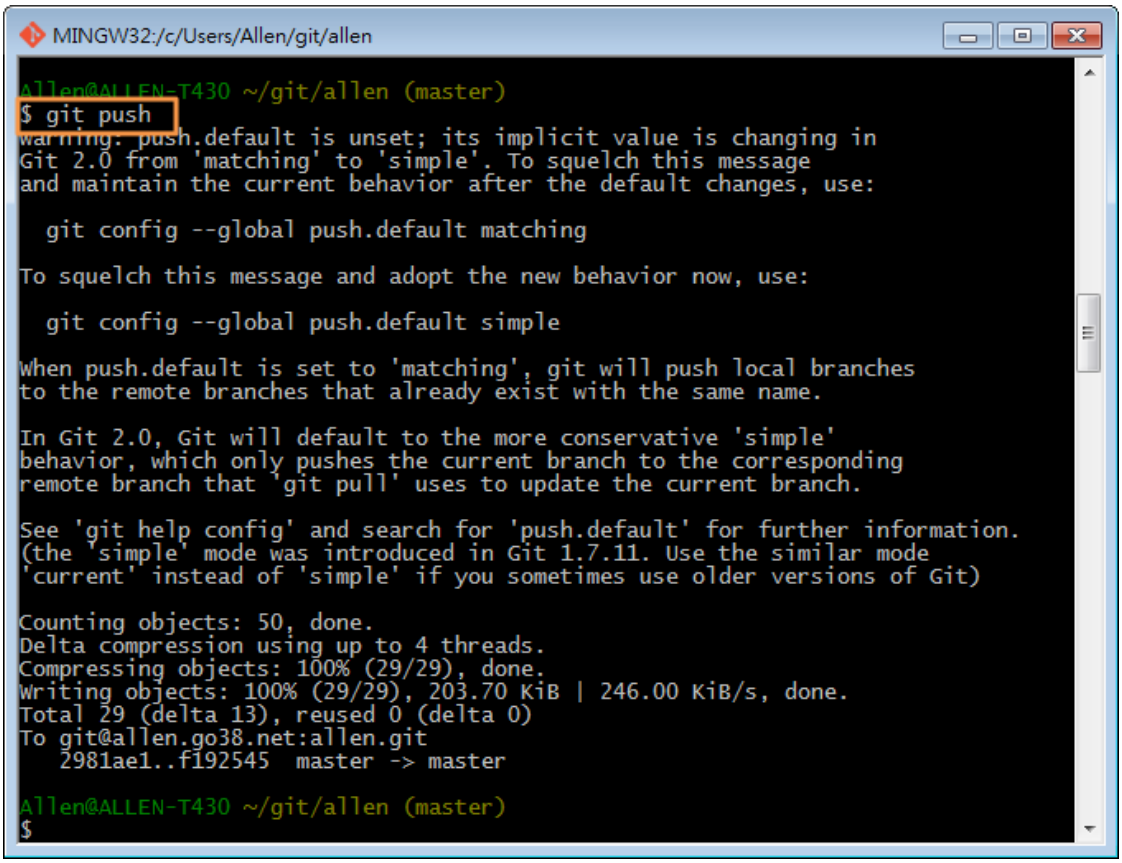
- 在這裡可以使用git status去顯示目前在本機檔案庫(Repo)內異動的狀況



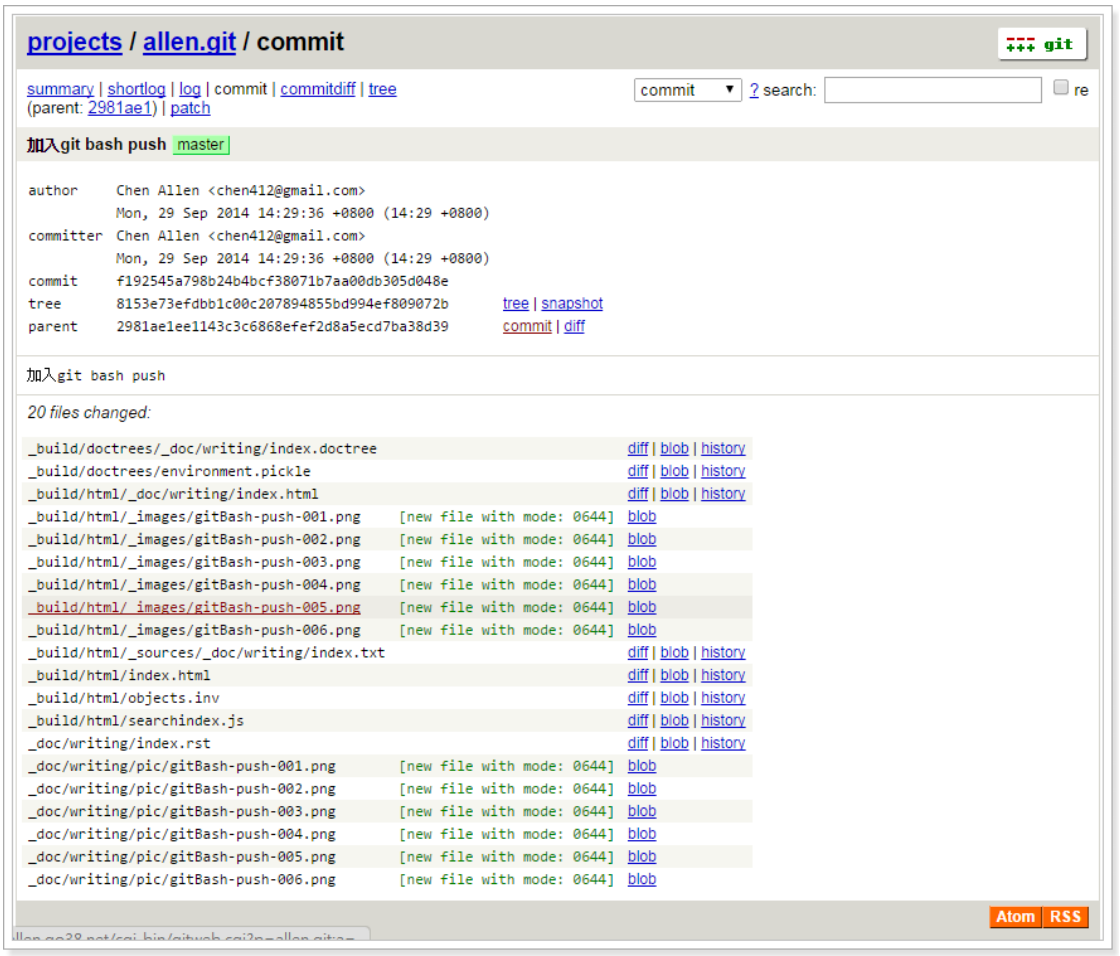
- 使提交(Commit)指令，『git commit -m ``記錄訊息''』，在使用Git提交一定要加入『記錄訊息』



- 最後使用上傳推送(Push)指令將所提交(Push)的內容上傳到伺服器中，『git push』



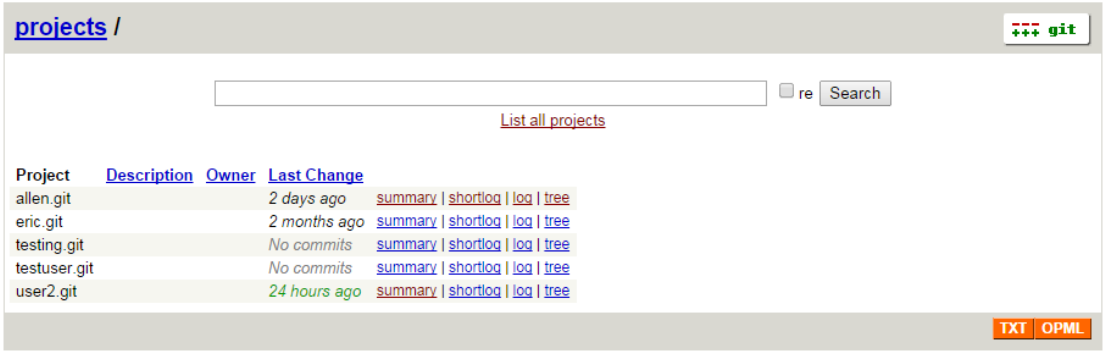
- 這時可以到 gitweb 上看到該檔案庫(Repo)的更新內容



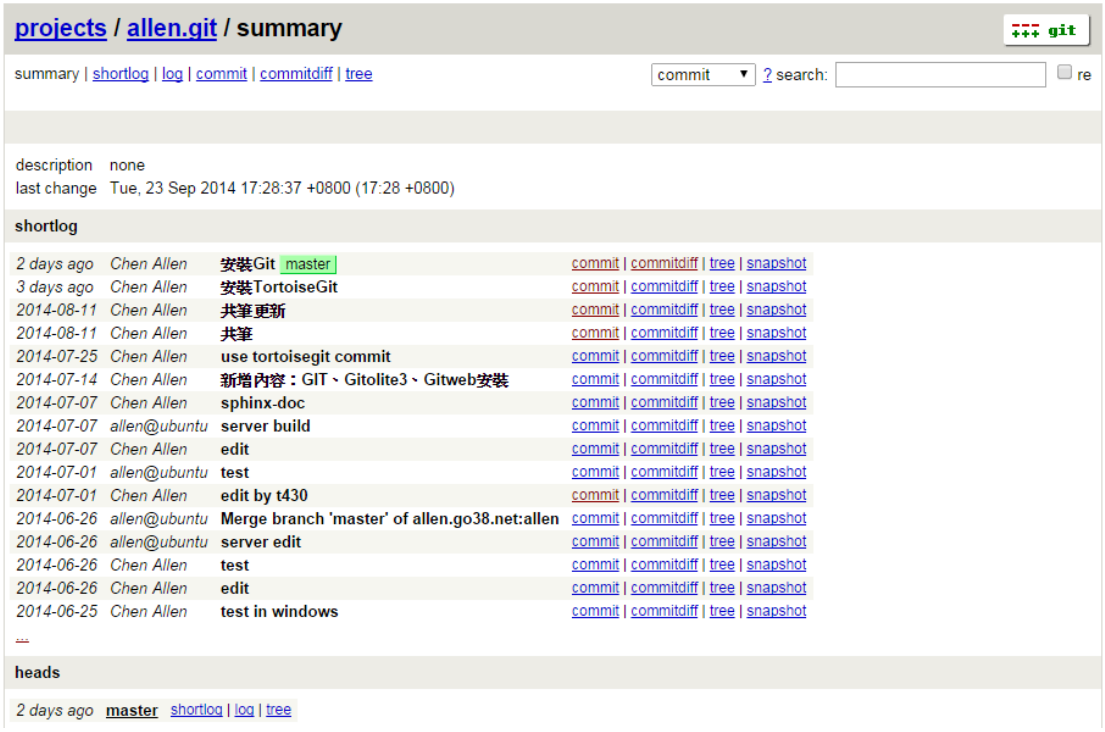
5.5 GitWeb 線上觀看修改內容

在伺服器上有安裝 gitweb 此網站會將有開放的檔案庫(Repo)顯示在上面，下面有一些gitweb執行的畫面

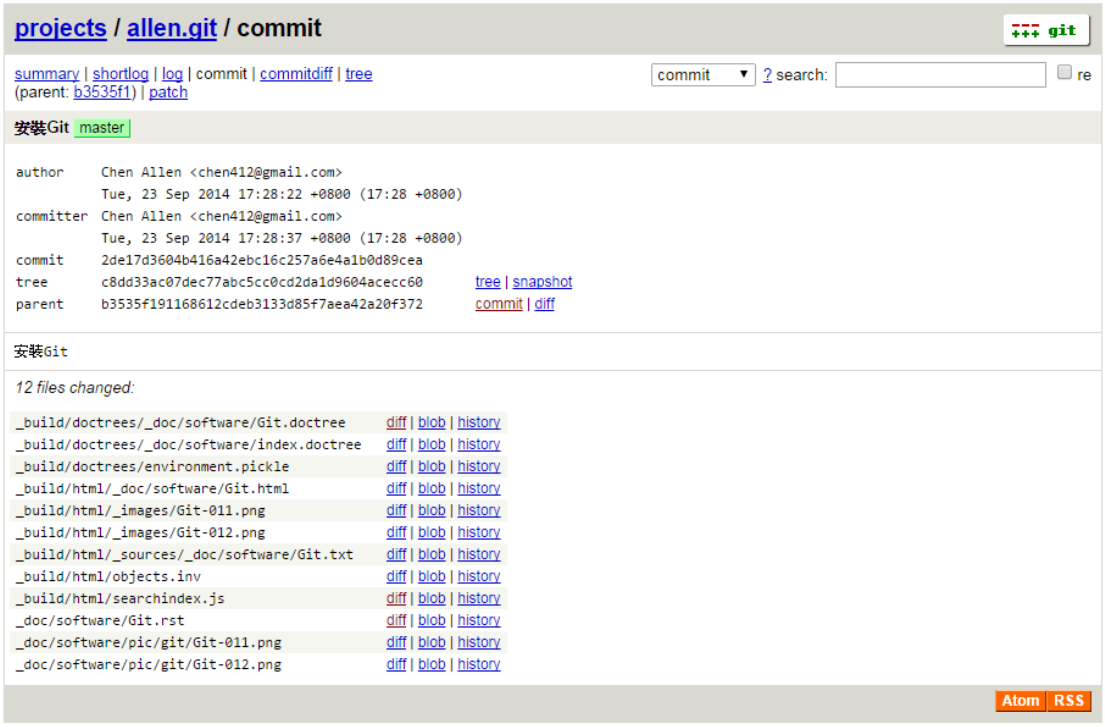
- 首頁 ，這裡會顯示所有檔案庫(Repo)的清單，後面會有一些連結可以不同的方式顯示記錄(summary | shortlog | log)，檔案內容(tree)



- 點選summary進入的頁面只會顯示比較近期的記錄，一樣後方也有一些連結，可以顯示commit的記錄，還有commitdiff與上一個版本的差異內容，snapshot可以打包下載這個版本的內容



- 點選commit後會進詳細的內容，例如有幾個檔案新增、修改、刪除等，可點選diff去看該檔案的版本差異內容

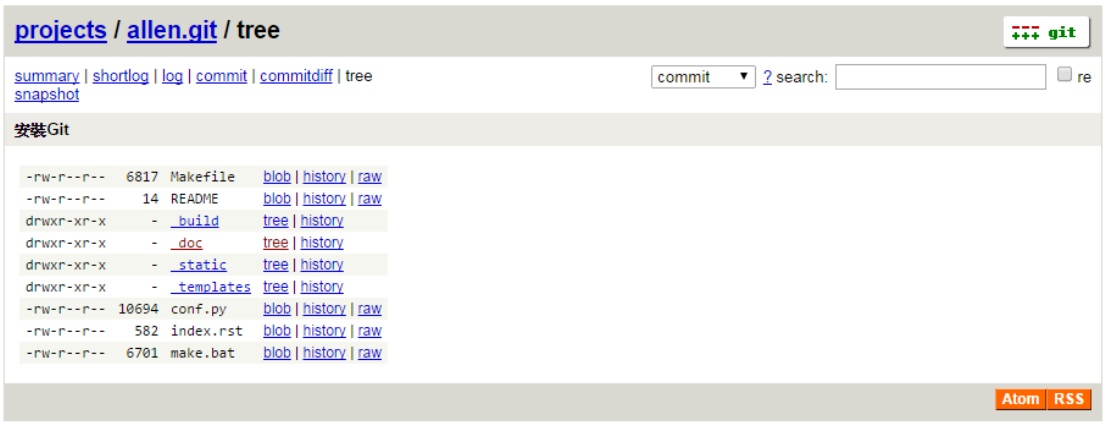


- 點選diff進入該檔案詳細的修改內容，如下圖，有『-』符號而且是紅色的文字代表被刪除，而有『+』符號並且是綠色文字代表新增內容，黑色文字就是沒有變動的內容

- 在專案名稱下方有一些超連結可點選觀看，下面的圖片爲點選『log』，會顯示比較完整的記錄項目

- 點選『tree』會顯示該檔案庫(Repo)下所有的檔案內容，如果爲檔案點選後可觀看檔案內容，若爲資

料夾則會進入該資料夾並顯示其檔案內容



CHAPTER

SIX

UBUNTU 安装

CHAPTER

SEVEN

軟體安裝

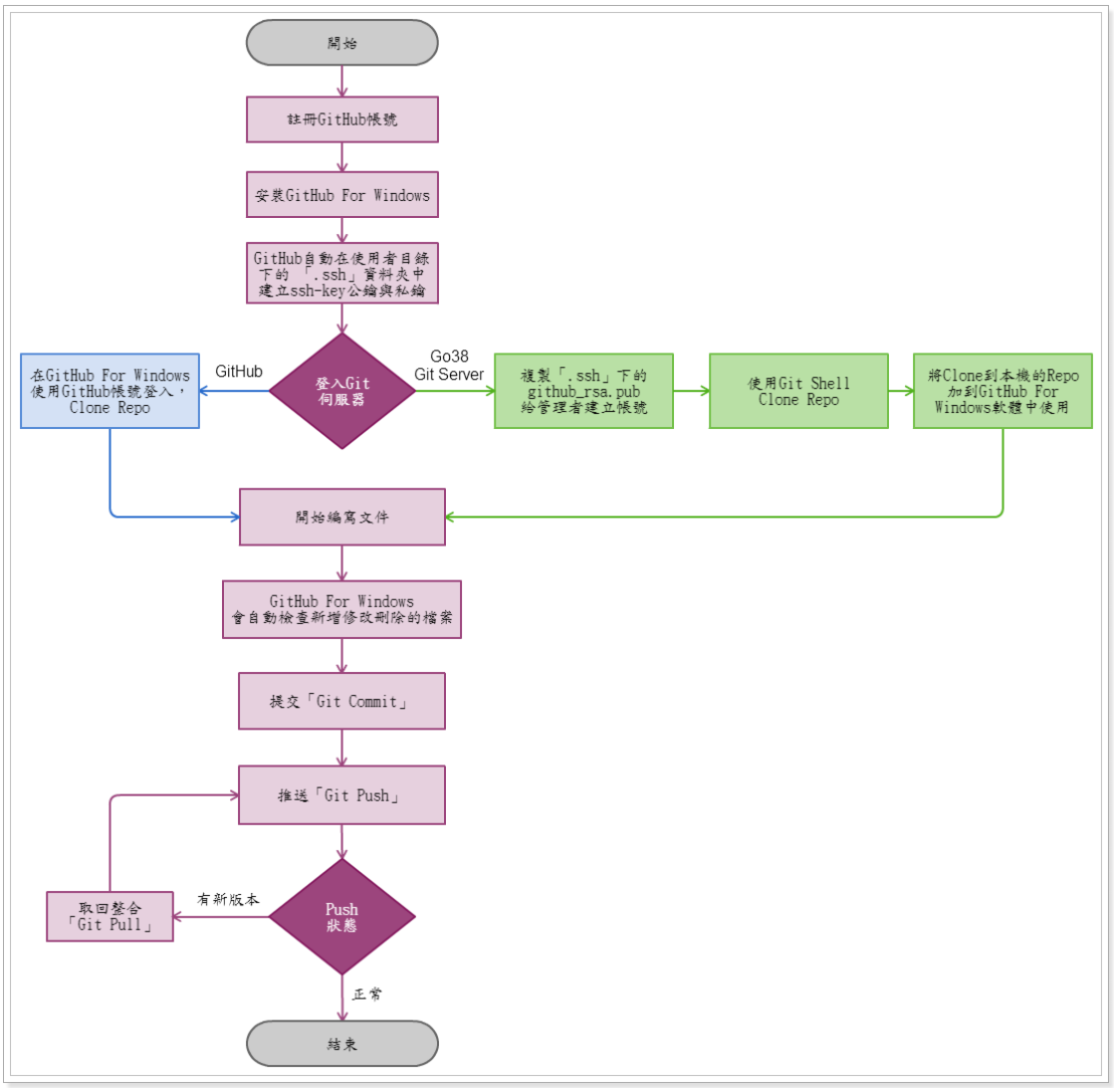
- TortoiseGit 安裝
- Git 安裝

CHAPTER

EIGHT

共筆寫作操作流程(GITHUB)

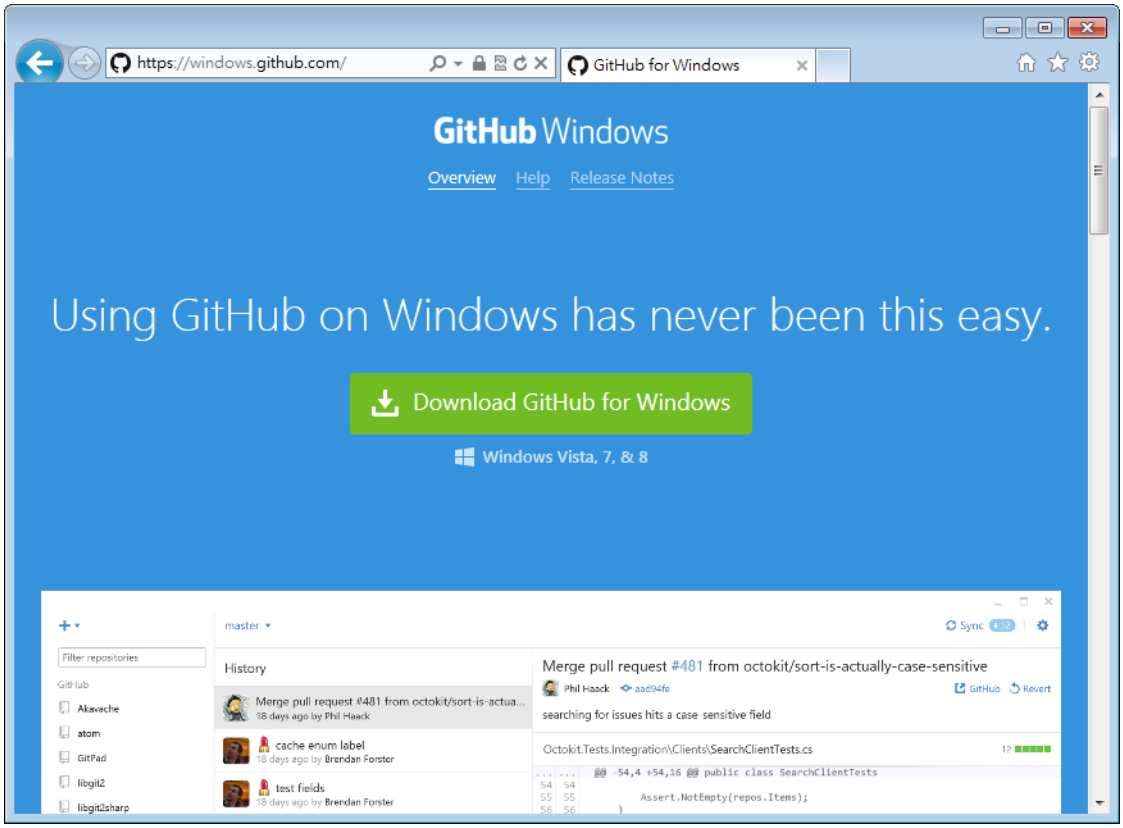
在上一版的「共同寫作操作流程」文件中是使用 TortoiseGit 存取Git伺服器上的儲存庫(Repo)，因爲最後使用 GitHub 平台，所以這篇主要是說明如何透過GitHub的軟體「GitHub Windows」共同存取寫作文件，當然也可以透過「GitHub Windows」存取Go38 Git伺服器上的儲存庫(Repo)，在「GitHub Windows 」軟體的操作上非常的簡單，而且也會自動產生ssh-key，節省了很多的步驟，而且所產生的ssh-key也可以直接供Go38的Git伺服器所使用，而Go38伺服器上的儲存庫(Repo)也可以使用「GitHub Windows」存取，下方的流程式爲共筆寫作的流程圖。



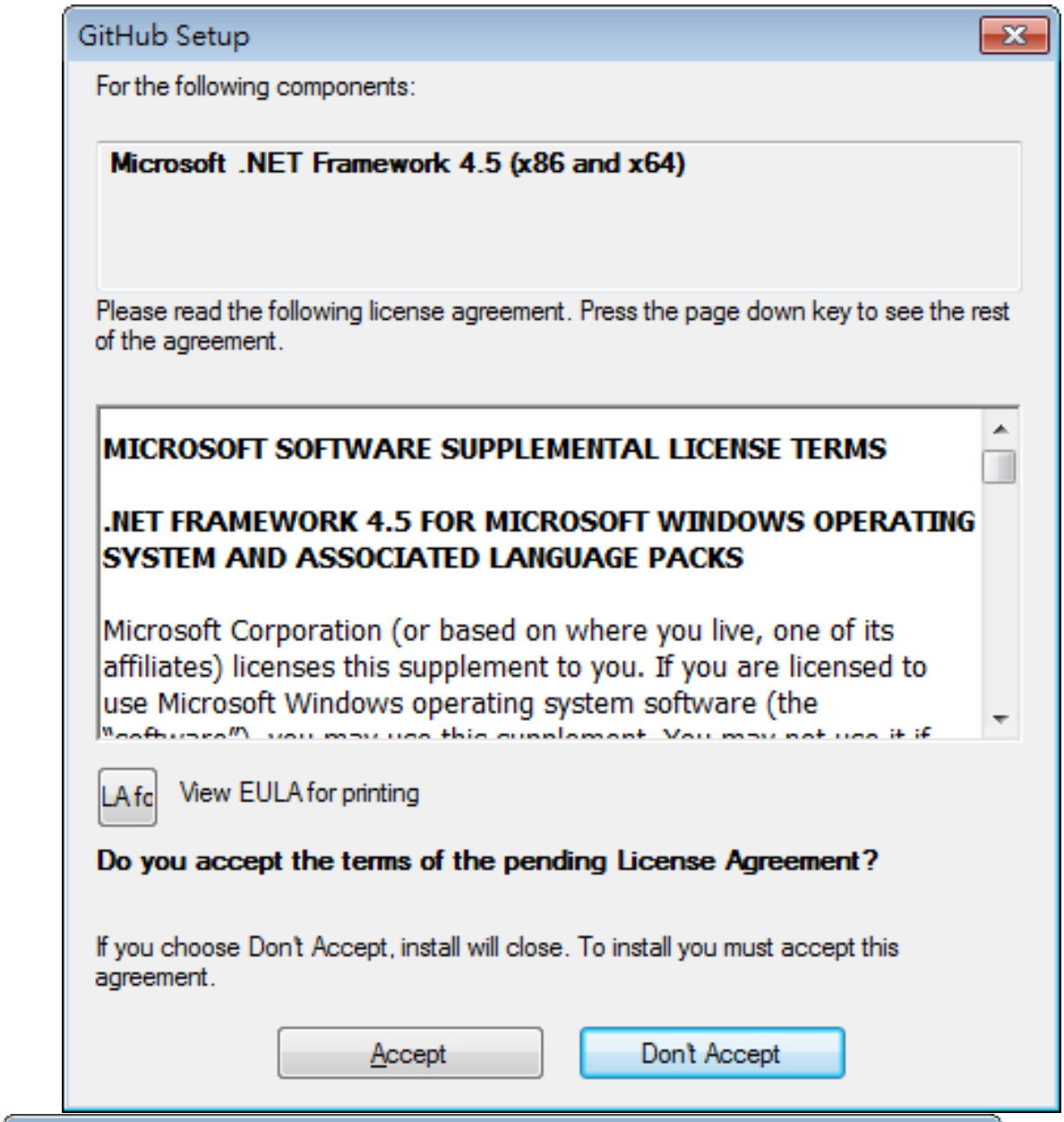
8.1 相關軟體安裝

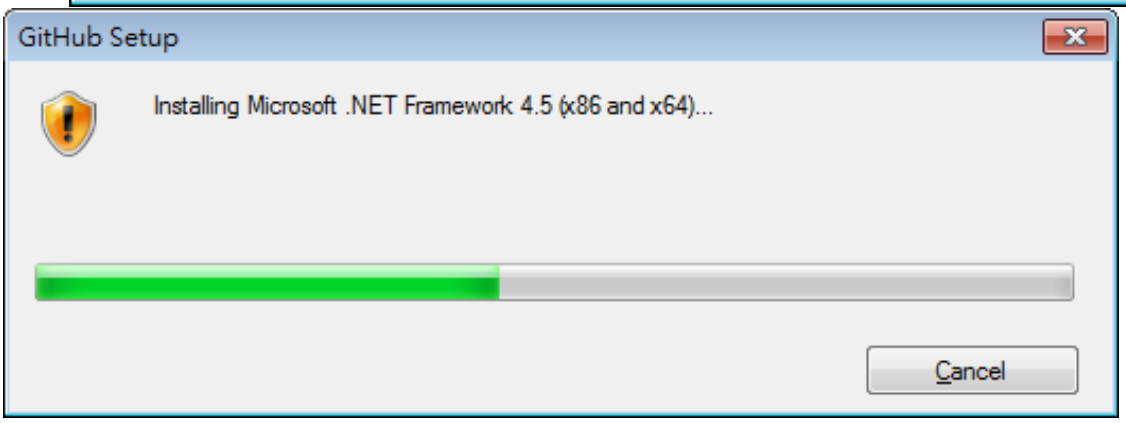
8.1.1 GitHub Windows

- 打開 GitHub Windows 網站，首頁就有『Download GitHub for Windows』連結，點選此連結下載



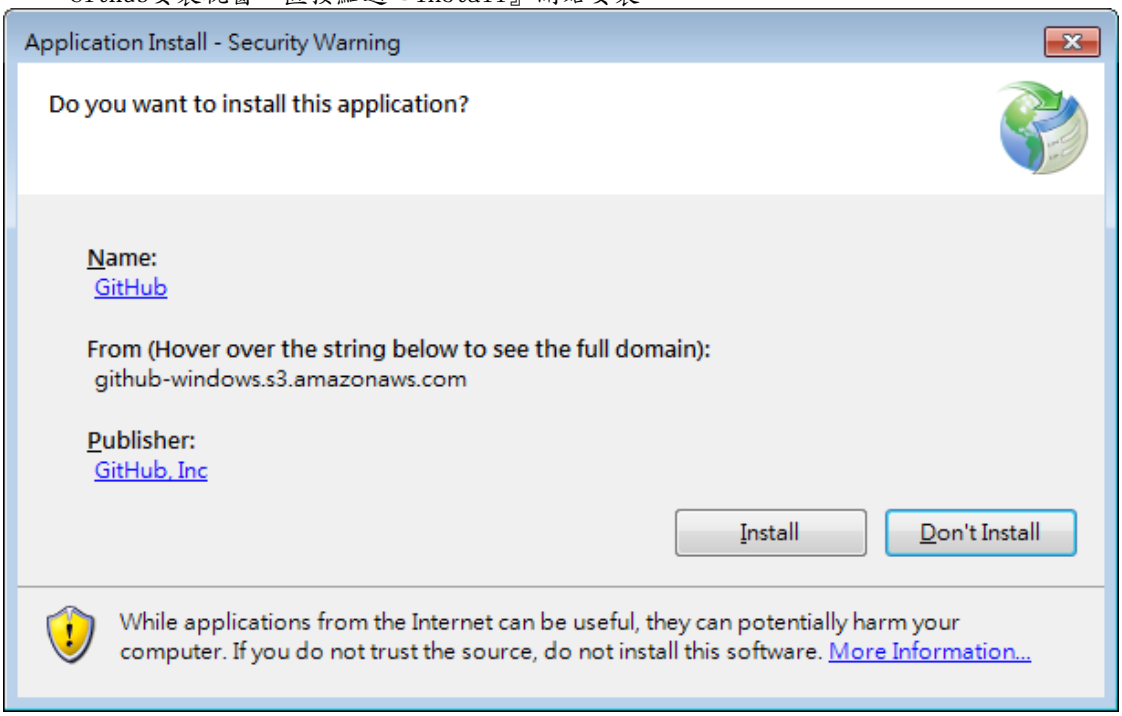
- 下載完後執行安裝程式『GitHubSetup.exe』，如果電腦已經安裝過GitHub Windows，執行後會直接啓動GitHub Windows程式；若沒安裝過，執行後會檢查電腦所需要的元件，如下圖在第一次安裝由於電腦中沒有『Microsoft .NET Framework 4.5(x86 and x64)』這個元件，會要求安裝，點選下方的『Accept』即可



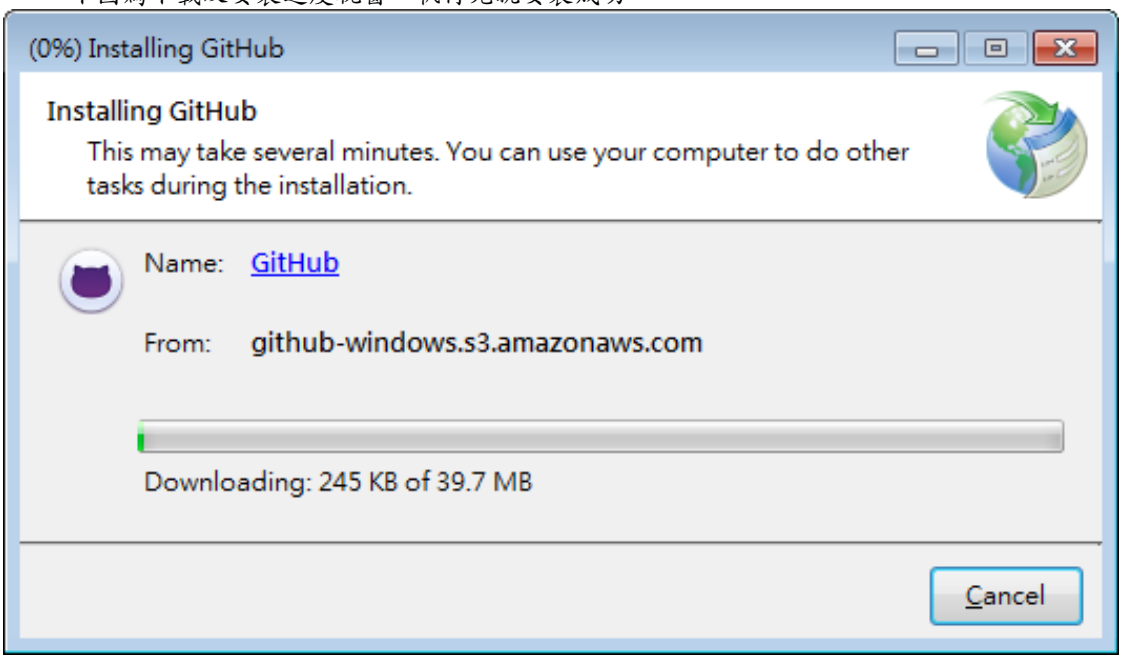


- 如果作業系統已安裝過『Microsoft .NET Framework 4.5(x86 and x64)』元件，執行後就直接出現

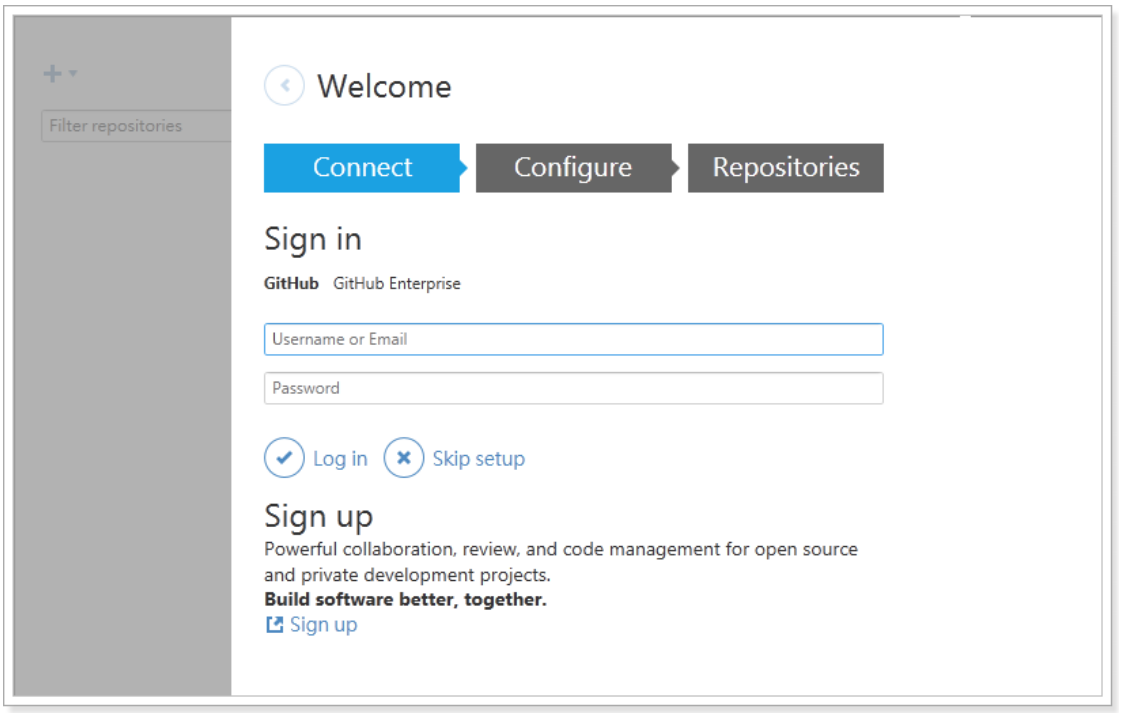
GitHub安裝視窗，直接點選『Install』開始安裝



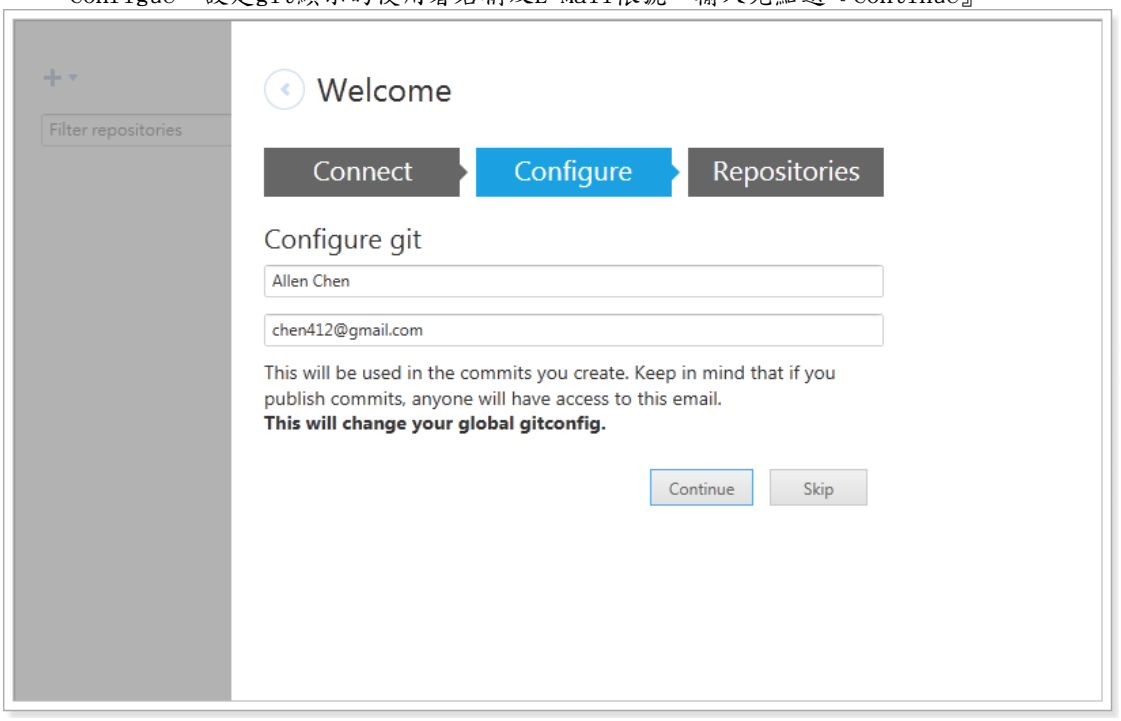
- 下圖爲下載及安裝進度視窗，執行完就安裝成功



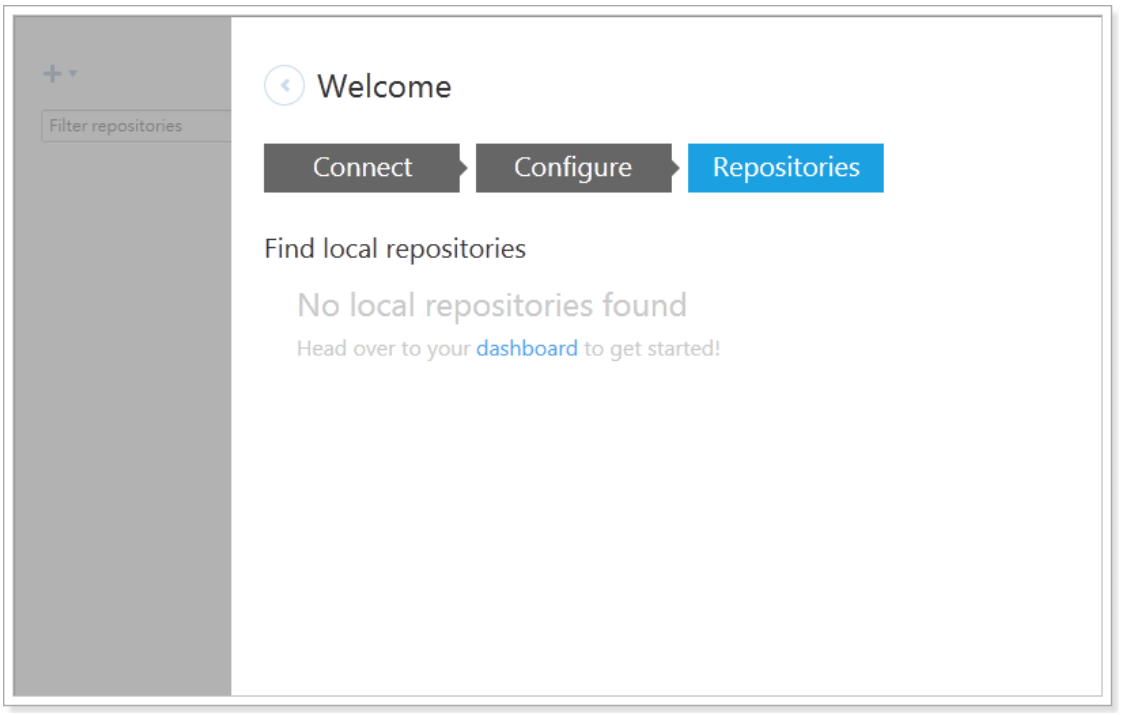
- Connect，安裝完第一次執行GitHub Windows的畫面如下，輸入GitHub帳號及密碼，點選『Log in』登入與GitHub平台連線



- Configue，設定git顯示的使用者名稱及E-Mail帳號，輸入完點選『Continue』



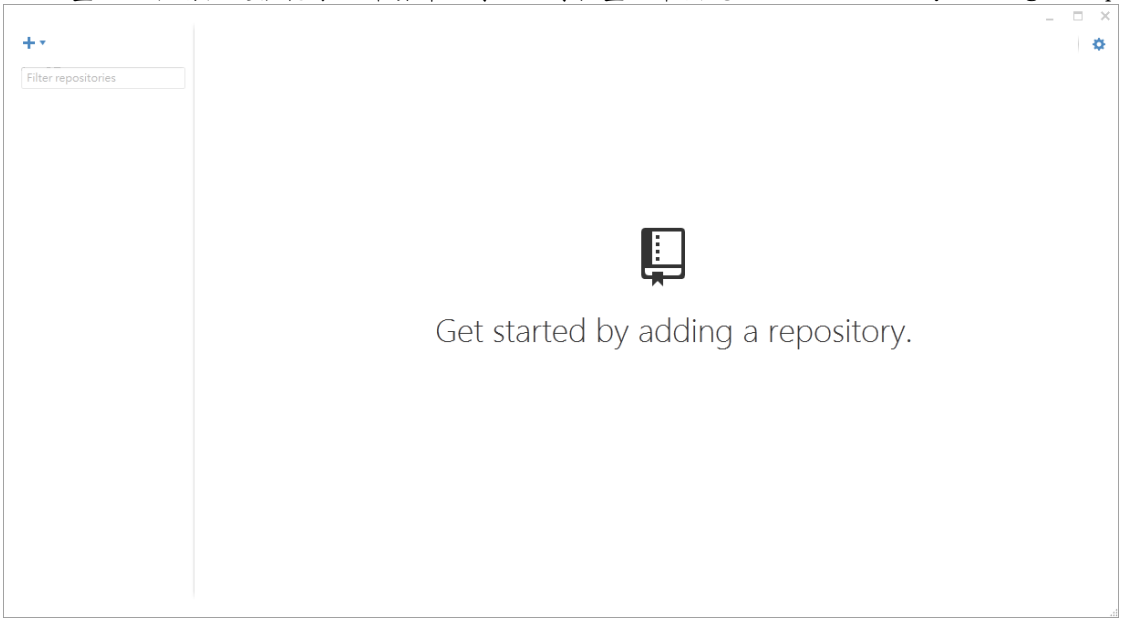
- Repositories，這一步軟體會自動尋找本機的儲存庫(Repo)，如果沒有找到，直接點選『dashboard』，進入軟體主畫面。



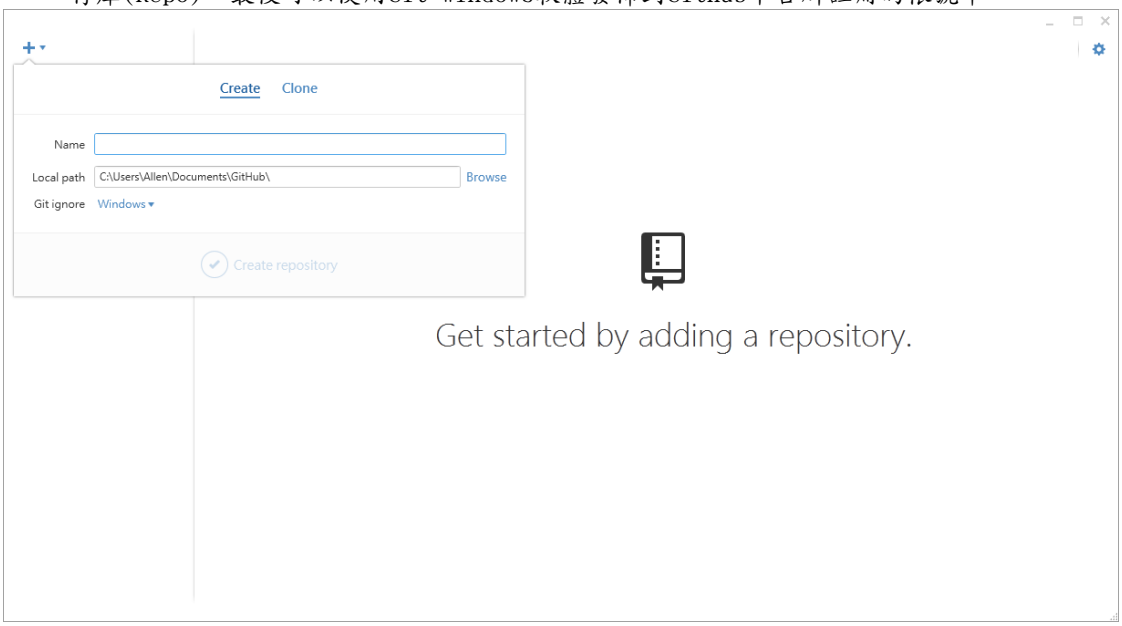
8.2 存取 Github Repo

8.2.1 複製Clone Repo

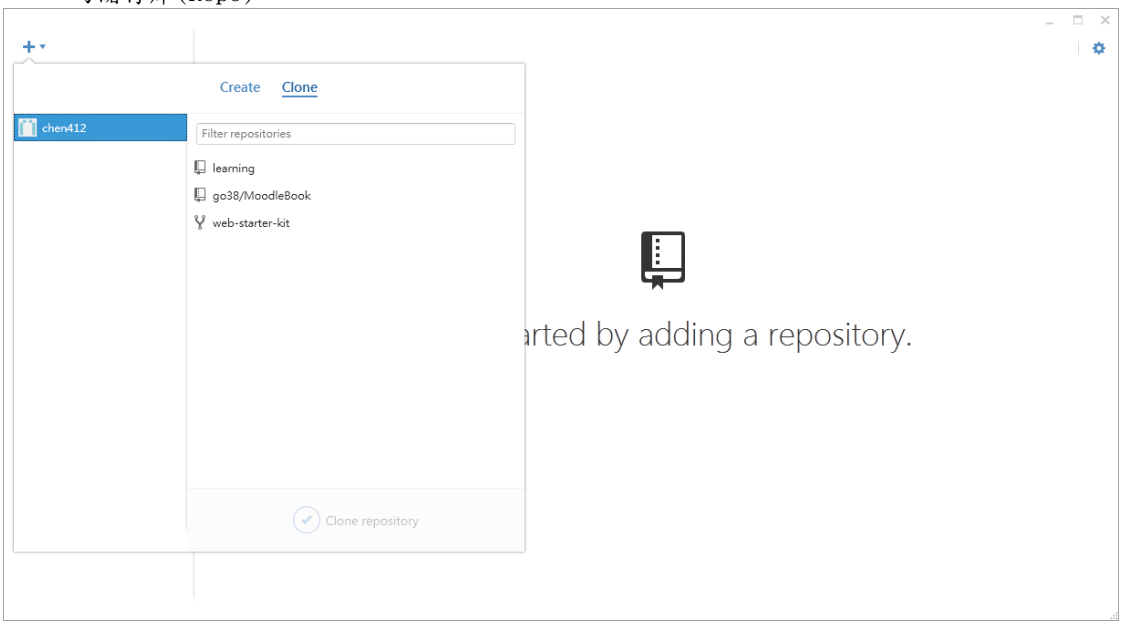
- 當安裝完GitHub Windows後，在還沒有複製(Clone)儲存庫(Repo)的狀況下，會看到如下面GitHub初始畫面，啓動後是很乾淨，什麼都沒有，只有在畫面中出現『Get started by adding a repository.』



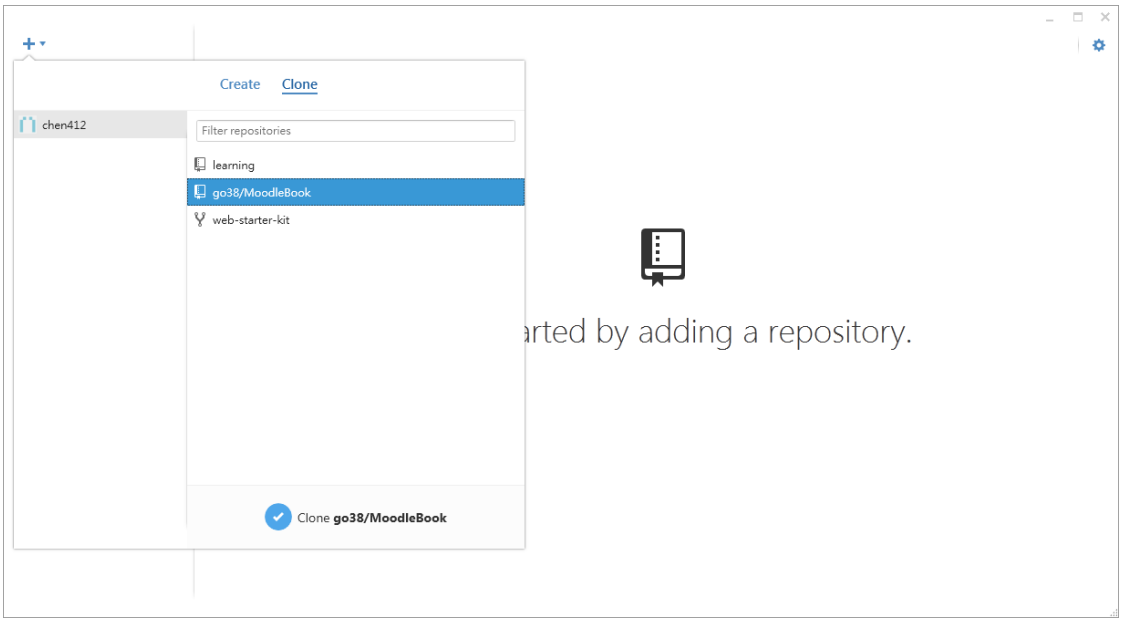
- 接下來點選左上角的『+』後會出現一個功能視窗，這個視窗主要的功能就是新增儲存庫(Repo)，可以看到視窗最上方有兩個選項『Creat』和『Clone』，下圖爲『Create』的畫面，設定Name和Local path後，點選下方的『Create repository』，就可以在所選的本機資料夾路徑中建立一個新的Git儲存庫(Repo)，最後可以使用Git Windows軟體發佈到GitHub平台所註冊的帳號中。



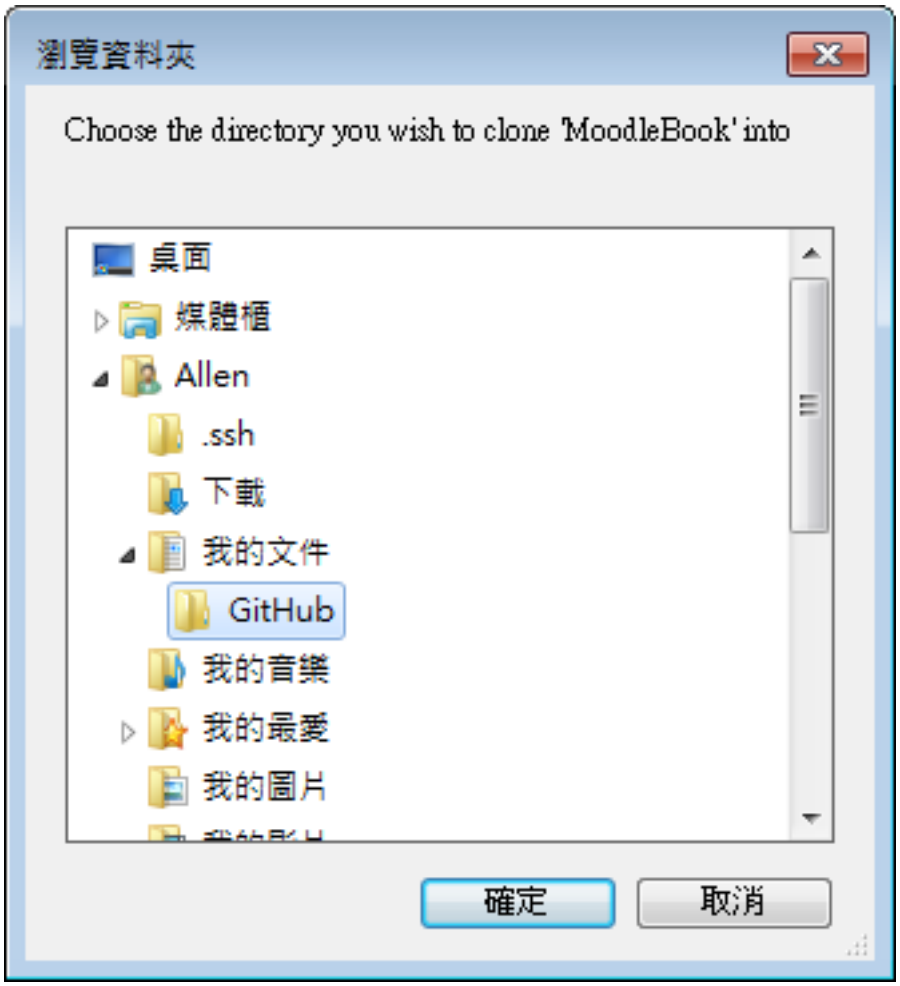
- 下圖爲『Clone』的畫面，當點選到『Clone』後，下方會自動載入安裝時所登入的GitHub帳號中所有的儲存庫(Repo)



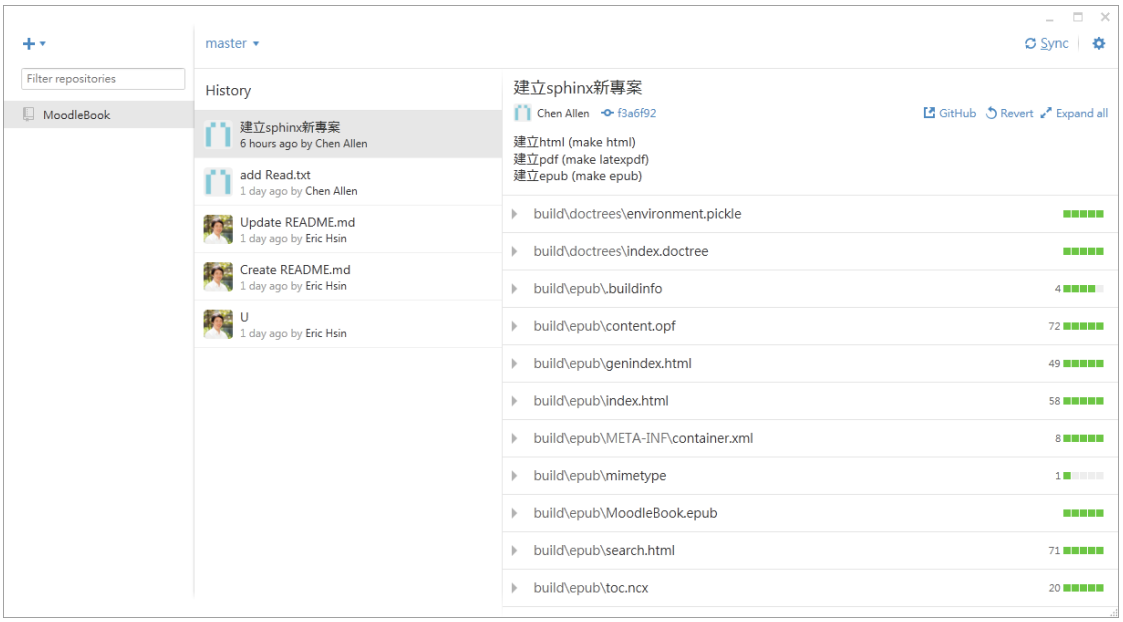
- 只要點選名稱，如『go38/MoodleBook』，再點選下方的『Clone repository』



- 在複製(Clone)之前會出『瀏覽資料夾』的視窗，在此處點選儲存庫(Repo)要放在資料夾，如畫面中爲預設的路徑(C:/Users/使用者名稱/我的文件/GitHub)

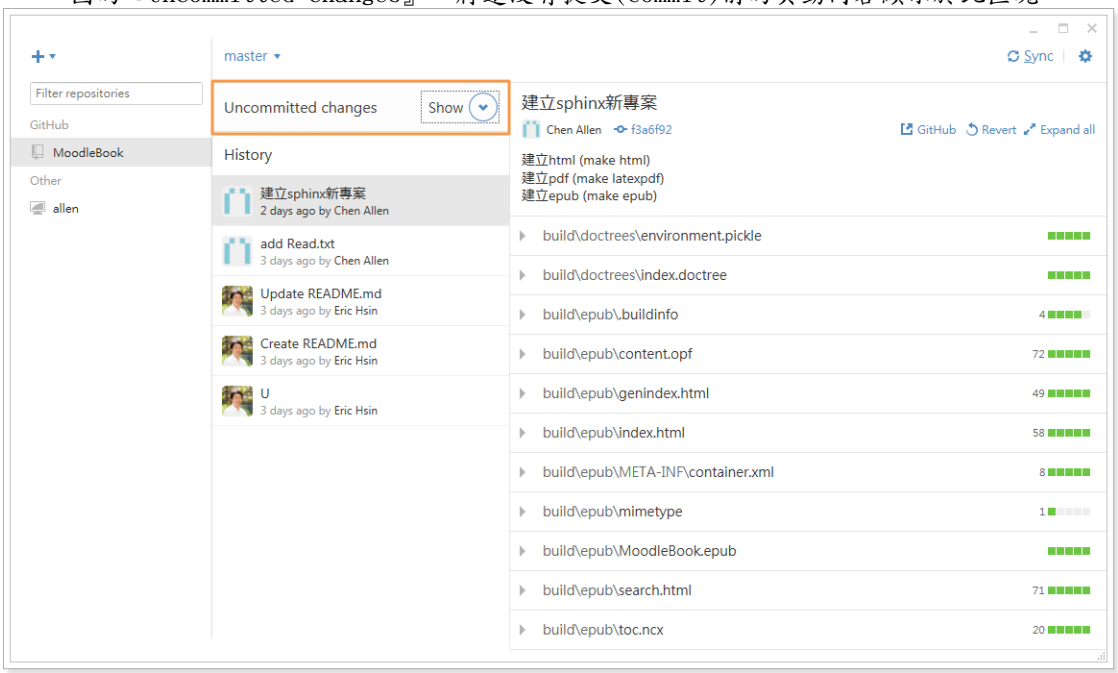


- 確認資料夾後就會開始複製(Clone)到本機電腦，完成後會看到如下的畫面，畫面主要分爲三個區塊，左邊爲本機的儲存庫(Repo)，如果有多個儲存庫(Repo)可點選做切換，而中間的『History』就是依左方所點選儲存庫(Repo)的歷史推送(Push)記錄，點選中間的歷史推送記錄後，在右方會顯示該次的異動內容，另外在『History』上方有個『master』，這爲目前的分支(Branch)名稱，點選後可以切換分支或建立新的分支。



8.2.2 檔案新增刪除修改

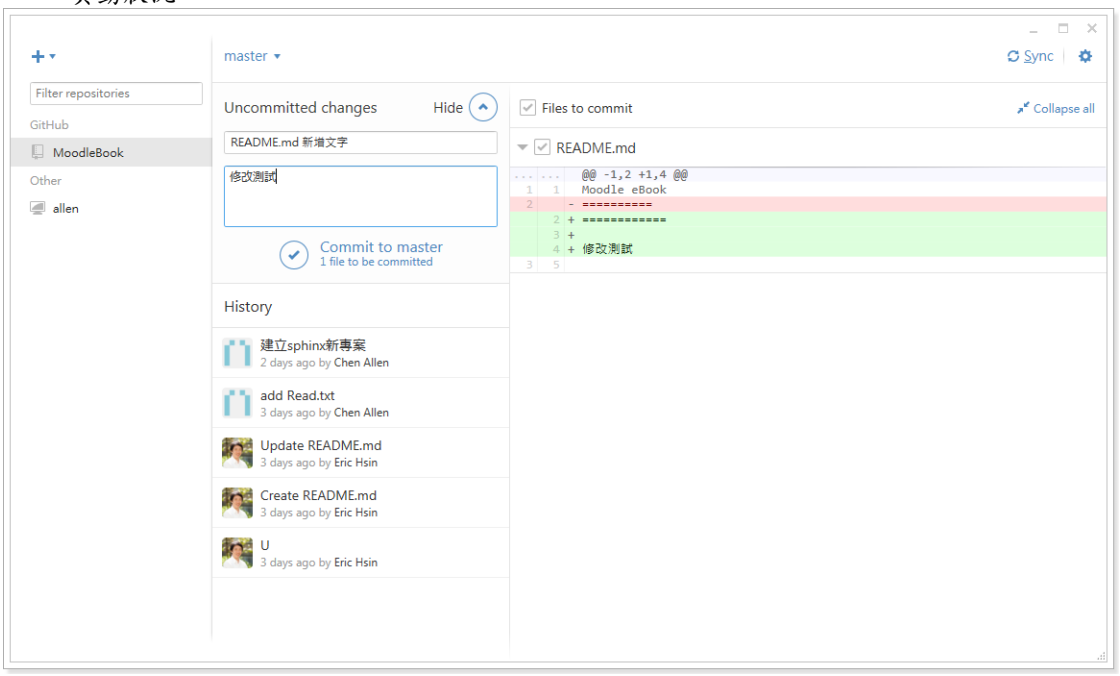
- 把共筆的儲檔庫(Repo)複製(Clone)到本機資料夾中後就可以開始編寫內容，當想把編寫好的內容放到遠端時，如果是使用Git Shell文字模式的狀況下，需要自行輸入指令，如要把新增的檔案加到儲存庫(Repo)則輸入「git add .」，「git add filename」，若要把該檔案從儲存庫中刪除，使用「git rm filename」，那如果使用TortoiseGit軟體，也需要在該檔案點選右鍵去找到新增或刪除該檔案的功能選項，但使用GitHub Windows軟體，不需要在要新增或刪除的檔案去操作功能選項，也不需要自行下Git指令，只要打開GitHub Windows軟體，它會自動把檔案異動內容做處理並顯示在軟體中，如下圖的『Uncommitted changes』，將還沒有提交(Commit)前的異動內容顯示於此區塊。



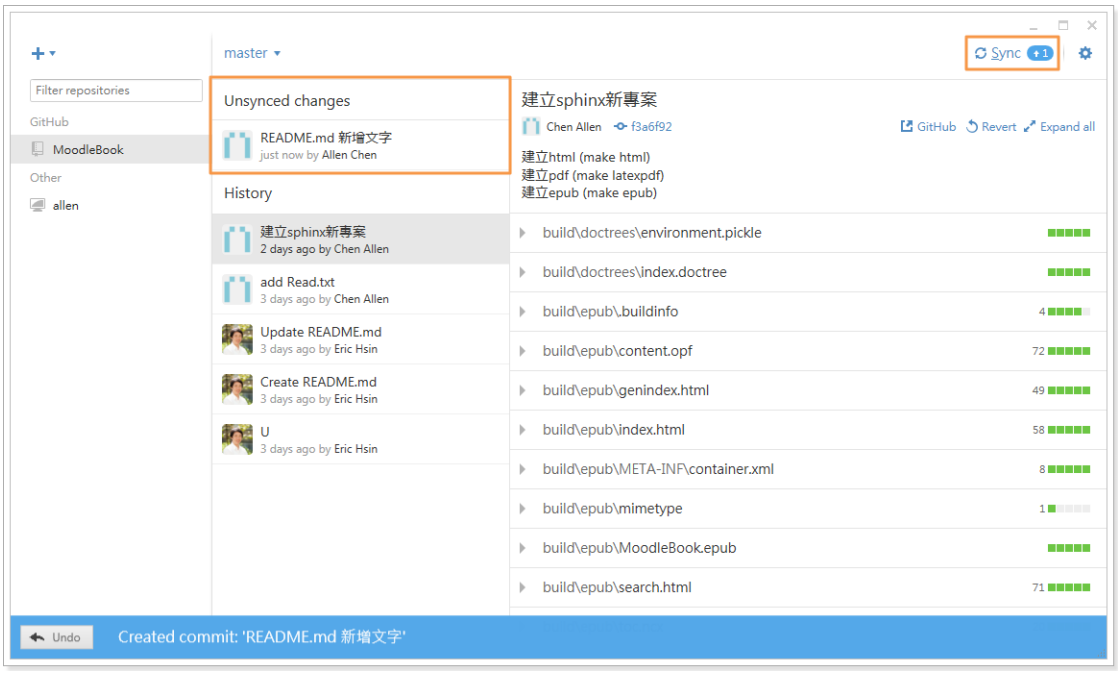
8.2.3 提交 Commit Repo

- 儲存庫(Repo)中的檔案異動完畢，可以點選『Show』，下方會出現摘要簡介(Summary)及說明(Description)，至少要輸入摘要簡介(Summary)這個輸入欄位，讓其他編輯者了解此次提交(Commit)的目的，那如果要寫的比較詳細，請填寫說明(Description)的輸入欄位，如果在提交前想要看一下檔案的異動詳細內容，可在右方的區塊看到所列出的檔案，每個檔案都可以點選觀看該檔案的修改內容減號『-』代表刪除，加號『+』代表新增，填寫完並且確認完檔案後點選下方的『Commit to master』即可，「master」是分析的名稱。

  上面的操作轉換到Git Shell文字模式下，需要輸入『git commit』，然後會自動在文字編輯器下開起一個檔案，需要在上面輸入這次提交(commit)的訊息，最後儲存檔案結束，如果要快一點的話可以用『git commit -m ``摘要簡介(Summary)''』即可，此外可以執行『git status』列出此次提交的檔安異動狀況。



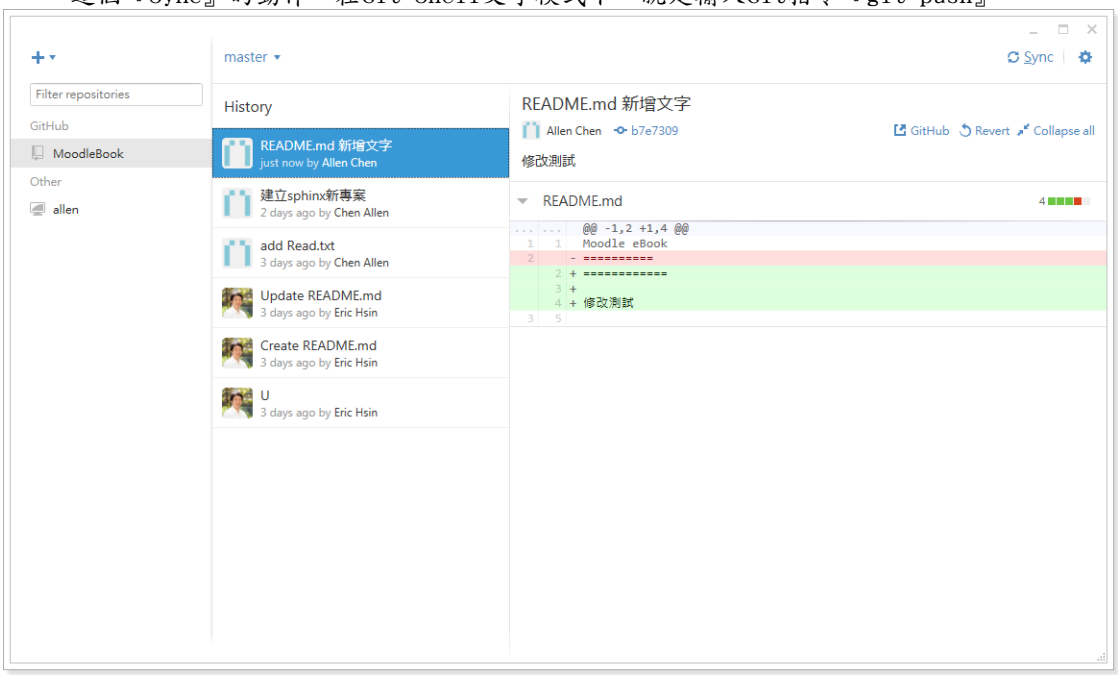
- 當點選『Commit to 分析名稱』後，『Uncommitted changes』的文字就會轉變成『Unsynced changes』，那右上方的『Sync』後方就會出現一個數字，該數字代表本機的儲存庫(Repo)執行了幾次的提交(Commit)動作，並且尚未推送(Push)到遠端伺服器中。下方有個『Undo』的按鈕，可以取消這一次的提交(Commit)動作。



8.2.4 推送 Push Repo

- 在軟體畫面右上方的『Sync』後方有數字出現，代表有提交(Commit)動作，而且尚未執行推送(Push)，使用GitHub Windows執行推送(Push)也是非常的簡單，只要在『Sync』上點選一下，就會自動推送到遠端的伺服器中，完成推送(Push)後，『Unsynced changes』下的項目內容就會出現在『History』中，而且『Sync』後方的數字也會不見。

  這個『Sync』的動作，在Git Shell文字模式下，就是輸入Git指令『git push』。

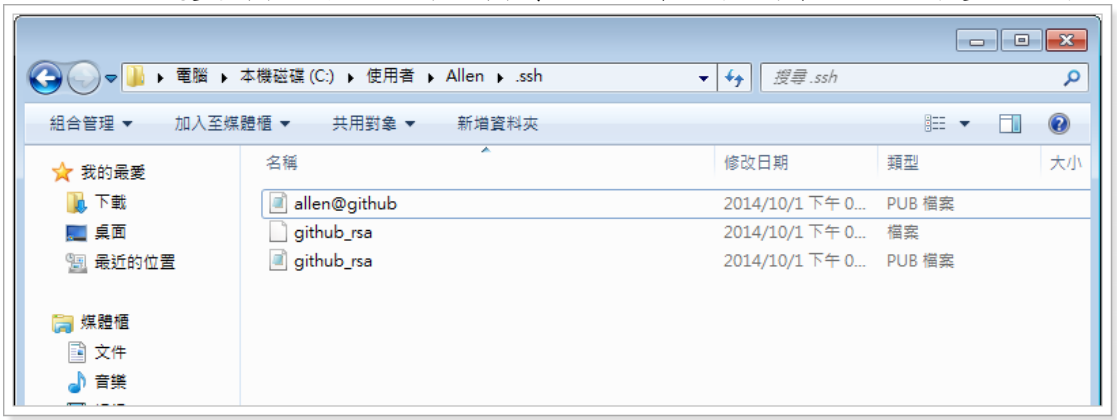


8.3 存取 Go38 Git Repo

使用GitHub Windows存取GitHub平台的存取很簡單方便，因爲GitHub Windows本來就是專爲GitHub平台操作使用的軟體，那如果要操作其他Git伺服器的儲存庫(Repo)，例如Go38伺服器上的Git儲存庫(Repo)，要如何操作使用呢？以下會說明該如何操作，主要有幾個步驟和GitHub Widnwos不同，如下

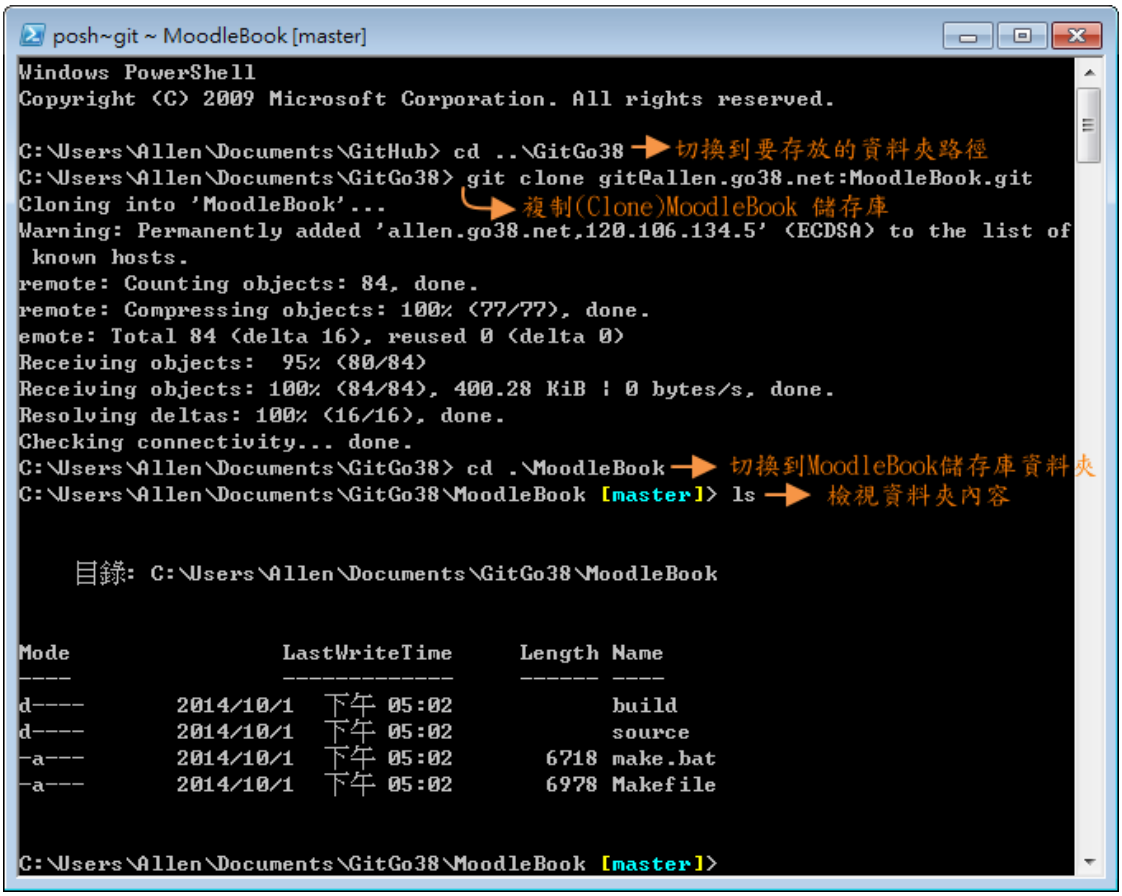
8.3.1 寄送公鑰(Public Key)

- 因爲在安裝GitHub Windows軟體時，會自動產生一組公鑰「github_rsa.pub」與私鑰「github_rsa」在使用者家目錄下的「.ssh」資料夾中，打開「.ssh」資料夾，將公鑰「github_rsa.pub」複製一份，然後變更檔案名稱，然後將該檔案傳送給Go38管理者，將會使用此公鑰建立使用者。



8.3.2 複製Clone Repo

- 當使用者建立完成後要複製(Clone)Go38上的儲存庫(Repo)，這裡的操作方式和GitHub平台不同，GitHub Windows可以直接使用GitHub的帳號登入並直接點複製(Clone)，但這裡我們需要使用Git Shell的文字模式執行『git clone git@allen.go38.net:MoodleBook.git』，將Go38上的MoodleBook儲存庫(Repo)複製(Clone)到本機電腦中，如下圖

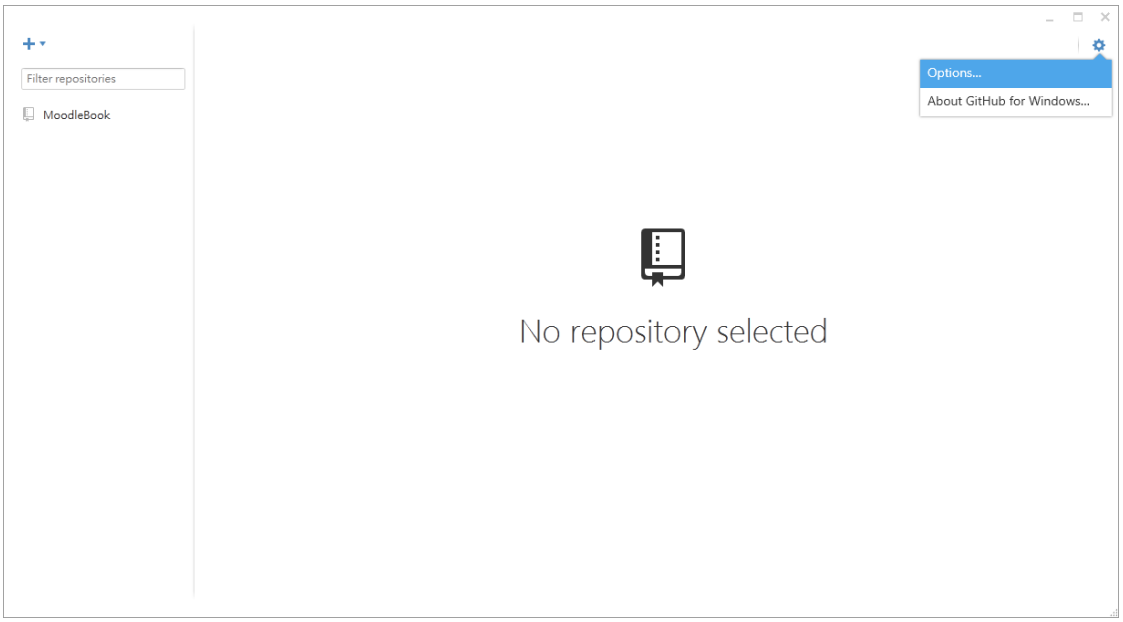


8.3.3 加入至GitHub Windows

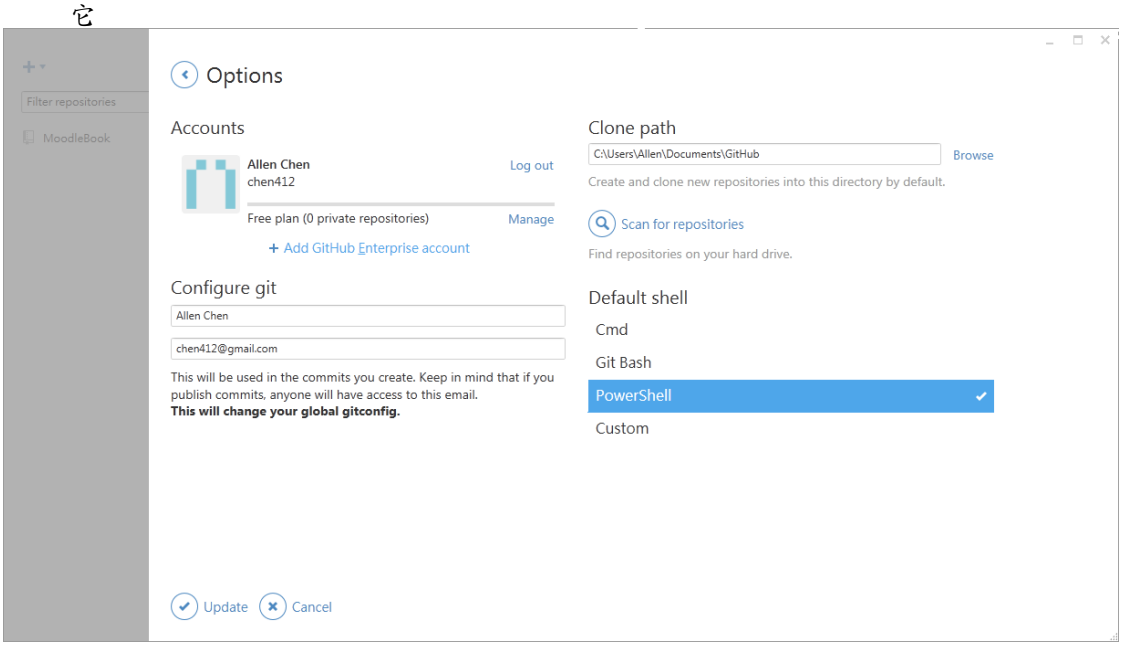
打開GitHub Windows軟體，怎麼看好像都沒有方法可以加入從別的伺服器中複製(Clone)下來的儲存庫(Repo)，在這有找到兩種加入的方法，如下

方法一：Scan for repositories

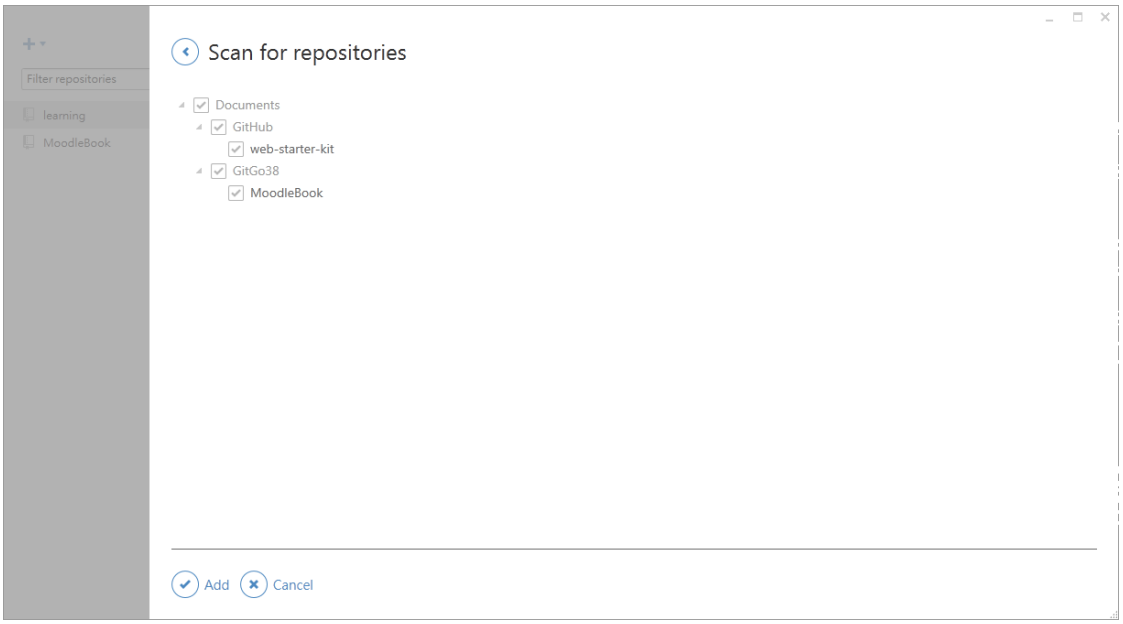
- 點選GitHub Windows右上角的齒輪(tools and options)，點選功能選單中的『Options...』



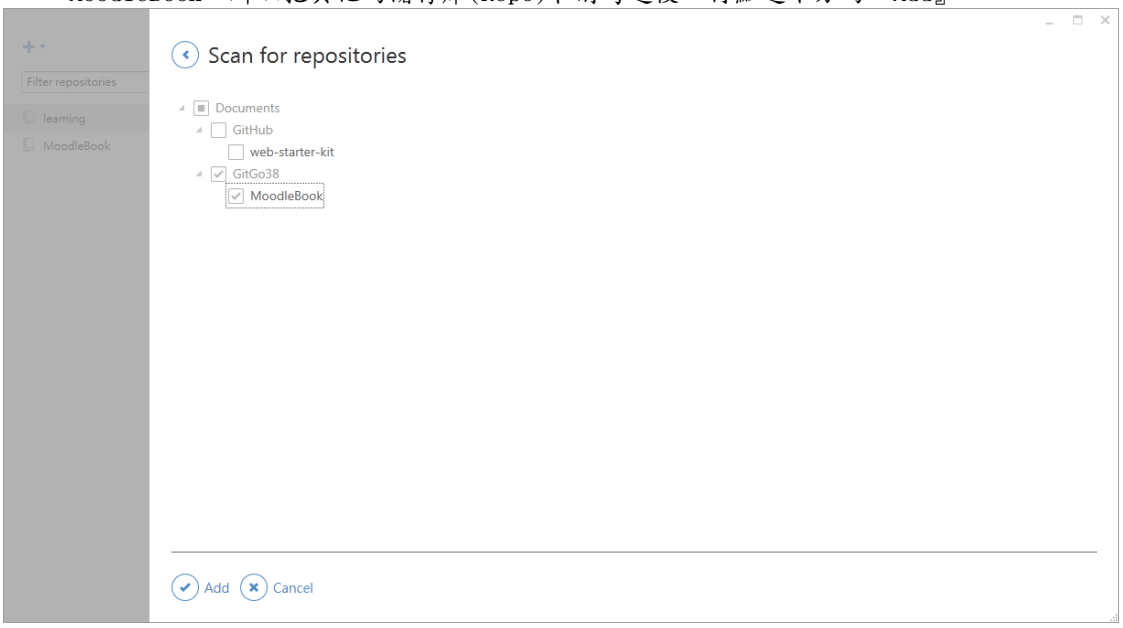
- 點選後會出現如下圖的Options視窗，尋找畫面中的放大鏡圖案『Scan for repositories』，並點選它



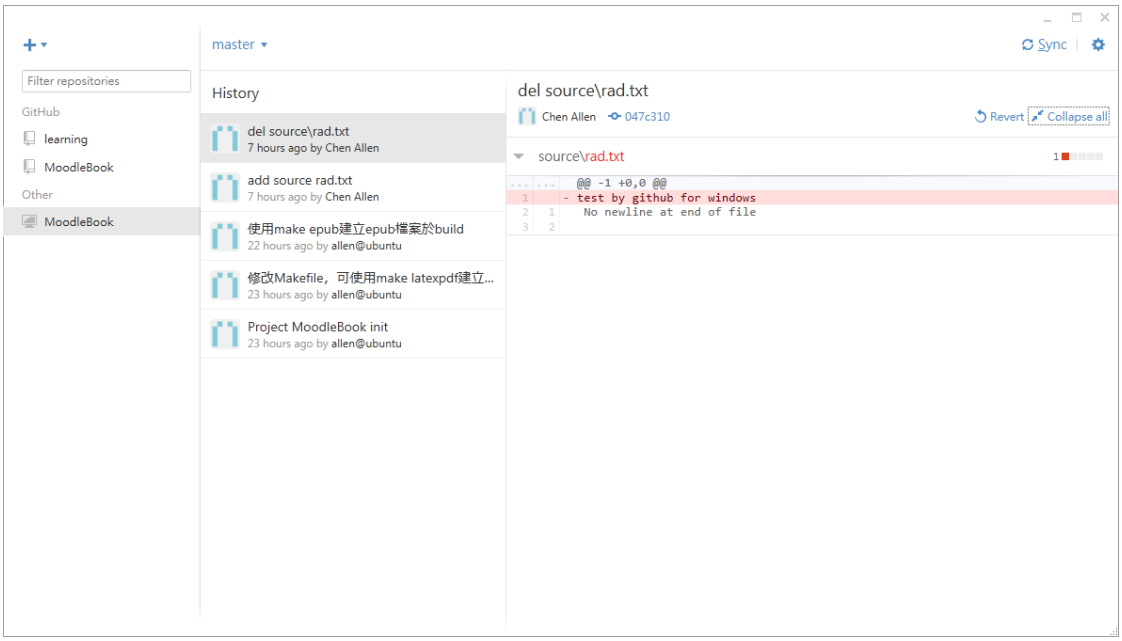
- 此時就會出現如下圖『Scan for repositories』的畫面，畫面中會出現在本機所找到的儲存庫(Repo)，經過測試GitHub Windows只會找使用者家目錄「C:Users使用者名稱」以下的資料夾，如果是放在其他資料夾下沒辦法被自動搜尋到的儲存庫(Repo)可使用方法二加入。



- 在本機電腦中可能有很多的儲存庫(Repo)，如下圖就有找到兩個，預設都是勾選的，因爲只要加入MoodleBook，所以把其他的儲存庫(Repo)取消勾選後，再點選下方的『Add』

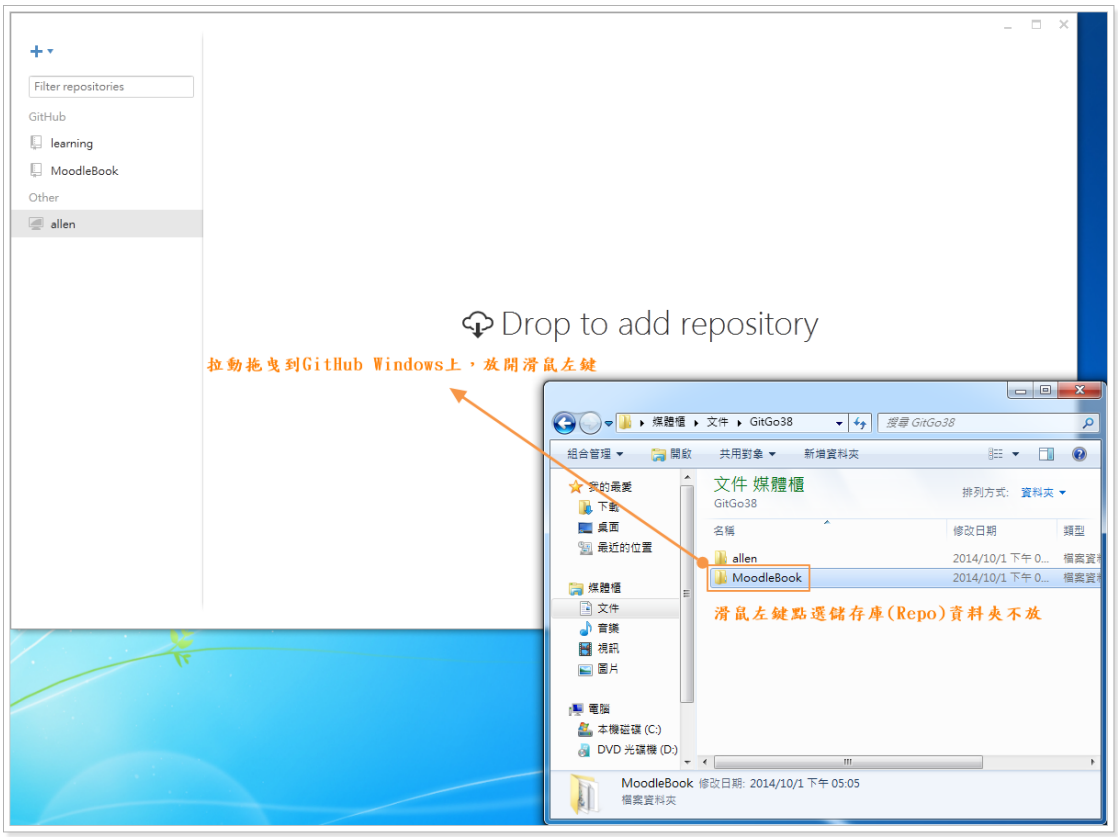


- 下圖爲加入Go38的MoodleBook存儲庫(Repo)後的狀況，會發現所加入的MoodleBook存儲庫(Repo)會放在標籤『Other』下方，而GitHub的儲存庫會被歸類在『GitHub』標籤下

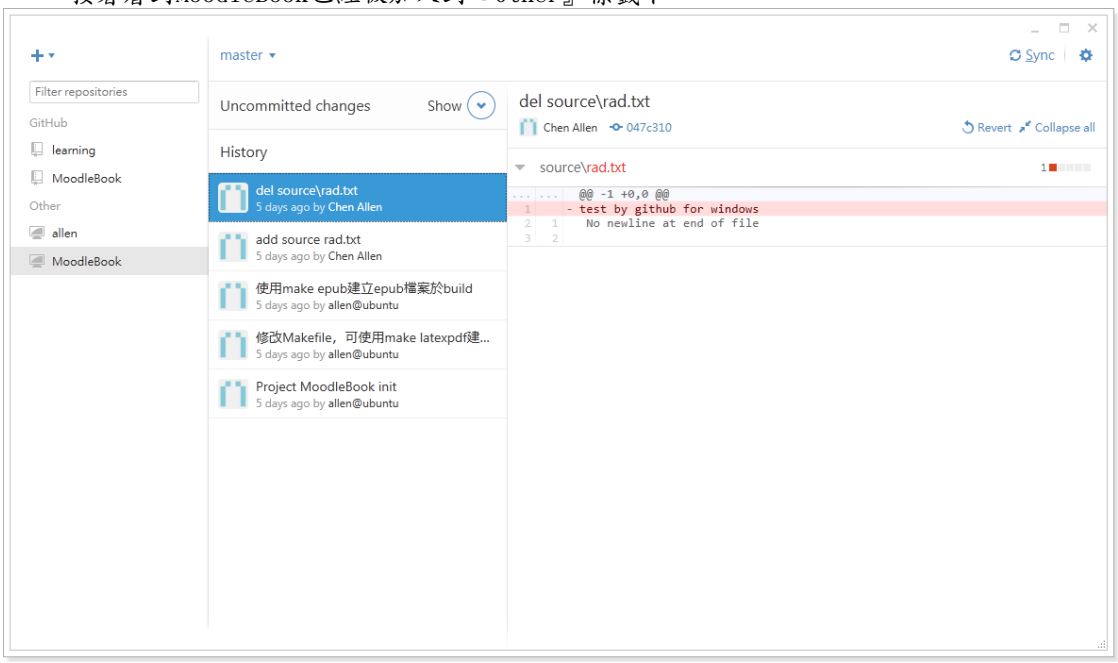


方法二：拖曳資料夾

- 第二種方法最簡單，只要打開檔案管理員，找到存放儲存庫(Repo)的資料夾，然後點選滑鼠左鍵不放，然後拉動拖曳儲存庫(Repo)的資料夾到GitHub Windows上，會看到出現『Drop to add repository』，然後放開滑鼠左鍵即可



- 接著看到MoodleBook已經被加入到『Other』標籤下



CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search