# WINE研析報告

## 目錄

A、為何需要 Wine ? **2**

B、何謂 Wine ? **2**

C、Wine 的架構 **3**

D、DLL ( Dynamic-link library ) **7**

E、啟動 Wine **8**

F、啟動 Wine32 **8**

G、在 Ubuntu 18.04 安裝 Wine **9**

H、Wine 軟體測試 **9**

I、參考資料 **11**

## A、 為何需要 Wine ?

通常不同的應用軟體是針對個別的作業系統而設計的，且大多數的軟體無法在不是為其設計的作業系統上執行。例如： Windows 程式不能在 Linux 系統中運行，因為它包含 Linux 無法解析的指令。同樣的， Linux 程式不能在 Windows 系統下運行，因為 Windows 無法執行 Linux 所有的+指令。

因此對於想要在 Windows 和 Linux 系統中運行同一個軟體的人來說，這種情況是一個很難解決的問題。此問題其中一種的解決方案是在同一台電腦上安裝多個作業系統 ( Dual booting ) ，當需要使用 Windows 程式時，用戶將電腦轉換到 Windows 系統中以運行它；當需要使用 Linux 程式時，用戶再將電腦重新啟動到 Linux 系統上執行，這種方法不僅需要在電腦上不斷切換各別所需的作業系統，也不能同時運行多個作業系統在同一台電腦上。使用 Dual booting 這種做法還需為各別的作業系統切割單獨的硬碟分割區，且Windows 也不是免費的作業系統，需要而外的成本來滿足所需。第二種解決方法是在原有電腦的作業系統上建立虛擬機，虛擬機是在原有的硬體和作業系統上模擬出一套硬體並在上面運行一個完整的作業系統，但若使用虛擬機會發現它的性能並不太好且複雜，因為運行虛擬機會需要更多的RAM等硬體來支援，以犧牲原先電腦硬體所具備的性能，若需要運行遊戲等需要大量圖形支援或需要大量硬體加速的軟體時，會發現跟實體機相比虛擬機的表現會遜色不少。因此為了讓應用程式運行在其不支援的作業系統上且又要有良好的表現，需要另尋解決方法。

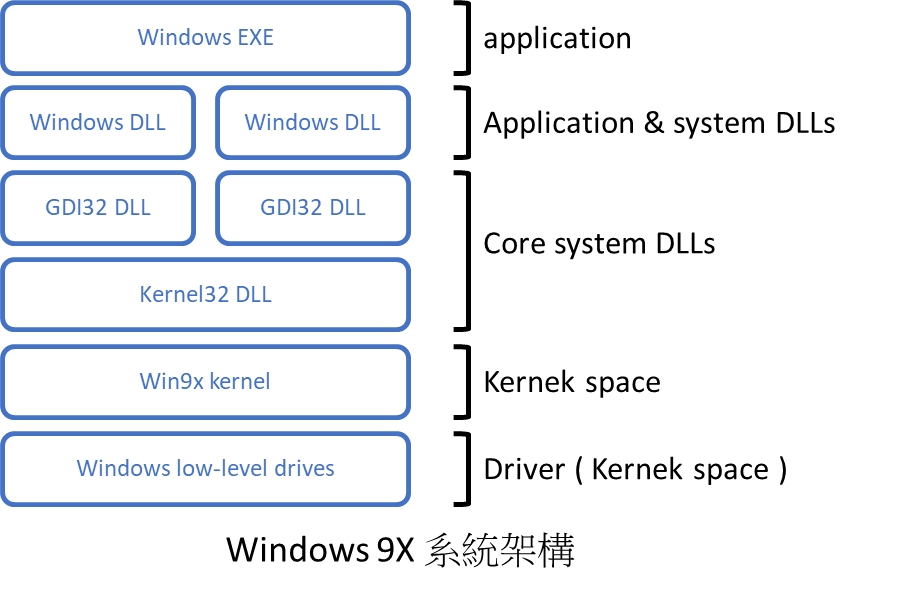
對於需要快速、簡單且擁有良好的 3D 圖形支援需求的使用者來說，Wine 可能是最好的選擇之一， Wine 能幫助 Windows 軟體在 Linux 作業系統上執行，當然 Wine 並不能支援所有的 Windows 程式，但若需要的應用程式能藉由 Wine 而成功在 Linux 上運行，則可以試試使用它，因為它是所有方法中最簡單且能獲得最佳效能的辦法。

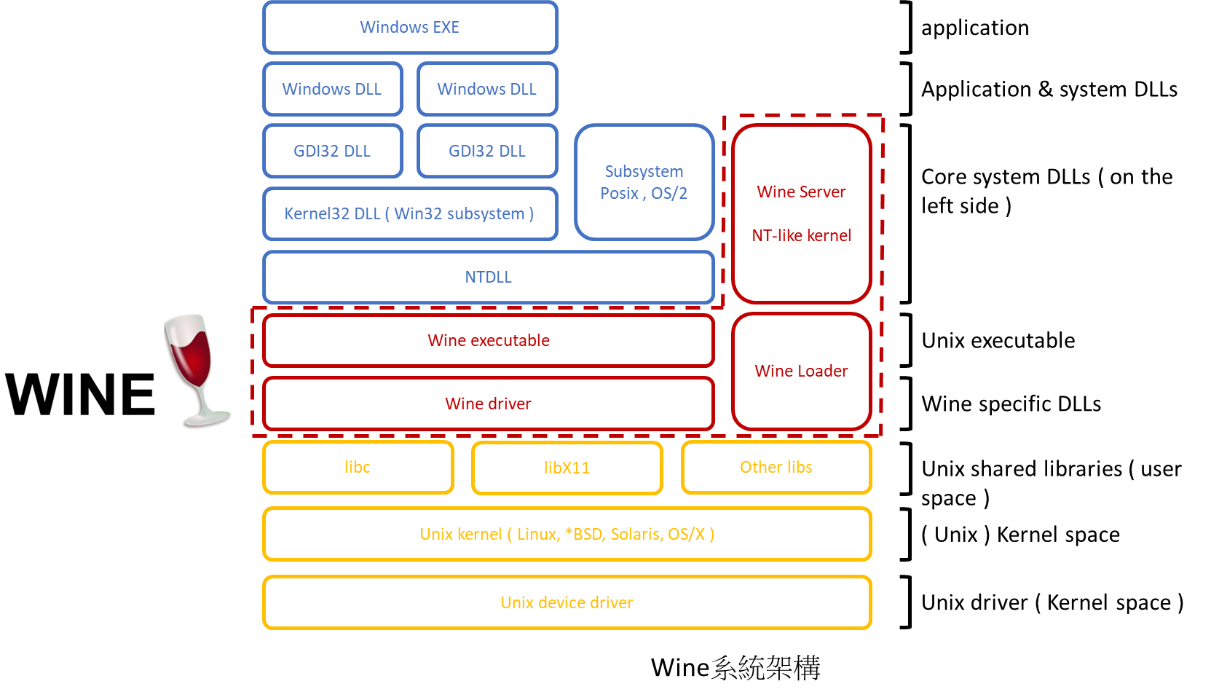
## B、 何謂 Wine ?

Wine ( Wine Is Not Emulator ) ( Windows Environment ) 的目標是簡單並重新創建 Windows 的操作功能，並使 Windows 應用軟體能在基於 UNIX 的作業系統下操作，也就是能夠在符合 POSIX 標準的作業系統上運行 Windows 應用程式，符合 POSIX 的作業系統如： Linux、macOS、BSD ( Berkeley Software Distribution ) 等。而 Wine 的主要任務就是要在非 Windows 作業系統下運行 Windows 可執行文件， Windows 可執行的文件包括 DOS可執行文件、Windows NE可執行文件 ( Windows New Executable 也就是Windows 3.x 文件格式 )、Windows PE ( Windows Portable Executable ) 可執行文件，當然 Wine 除了要運行這些 Windows 可執行文件，還必須可以運行 Winelib 可執行文件。Winelib 可執行文件是使用 Windows API 編寫但編譯為 Unix 可執行文件的應用軟體。但要在 UNIX 系統中模擬 Windows 作業系統必須有很多問題需要克服解決，且 Wine 為了符合開源授權的合法性，並不對 Windows 進行反組譯，而是利用黑箱測試法來開發，並參照未完全開放的開發文件逐步實作所有的 Win32 APIs。 所謂的黑箱測試法又稱功能測試法，就是單純考慮Input和Output的mapping，只考慮正確性，不考慮程式的內部邏輯結構及運行方法。

Wine 為一個相容層，所謂的相容層就是將面向異質系統的系統呼叫轉換為面向主機系統的系統呼叫，所以理論上只是新增了一層 software layer ，在這一層提供一個用來從 Windows 進出到 UNIX 的開發工具包 ( Winelib ) ，也提供一個程式加載器，這個程式加載器允許不用修改任何 Windows 的二進制文件，就可以在 UNIX 底下運行。 Wine 不是虛擬機，也不是模擬 Windows 內部的邏輯，所以它並不提供虛擬 CPU 及 I/O 等硬體設備，而是通過Winelib及時將呼叫 Windows API 轉換為呼叫 POSIX API，免除了硬體性能和其他記憶體的占用，所以能完整發揮電腦的硬體性能。 Windows 應用軟體主要依賴 NTDLL、Kernel32、GID32 和 User32 這四個動態鏈結函式庫來執行， Wine 基本已經實現這些動態鏈結函式庫與 Windows API 的對應，所以隨著時間，有越來越多 Windows 的應用軟體可以藉著 Wine 在其他 UNIX 系統下執行，但 Windows 應用軟體的頻繁更新，也使得往往 Wine 可以執行的應用軟體不是當前最新版本，而是更之前的版本，所以對於需要最新版的使用者來說， Wine 可能不是你最好的選擇。

## C、 Wine 的架構



在 Windows 9X 系統中包含 Kernel 32 DLL 、 GDI32 DLL 、 User32 DLL 等系統函式庫，其中 Kernel32 DLL 提供一些基本 API ，如記憶體管理、輸入 / 輸出的操作和同步函式等， GDI32 DLL 提供有關圖形裝置介面有關的函式，如輸出到顯示卡和列印機的原生繪圖功能， User32 DLL 提供創建和管理 Windows圖形介面的功能。若比較起 Windows 9X 和 Windows NT 兩個系統的差異，除了兩個的驅動程式模型和系統位元 ( Windows 9X為16位元系統，Windows NT為32位元系統 ) 等不同外。在上結構圖中 ( Windows 9X 系統架構、Windows NT 系統架構 ) ，可以看出 Windows NT 架構比起 Windows 9X 架構多了 NTDLL 這個動態鏈結函式庫 。 NTDLL ( NT Layer DLL ) 含有 Windows 的 Native API， Native API 通常由系統內部使用，NTDLL 用來允許實現不同的子系統 ，如 win32 等 ，在 Windows NT 結構系統中的 kernel32 在 NTDLL.DLL 之上實現 Win32 子系統，也可以理解為 Windows kernel ( Kernel32.dll 和 user32.dll ) 所提供的 API 都從這個NTDLL.DLL中所導出。

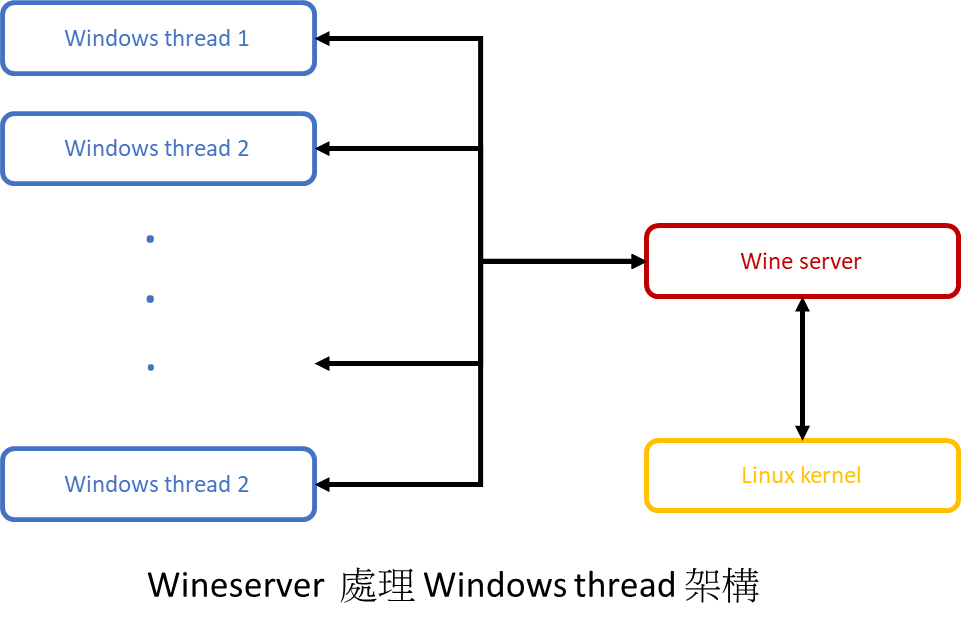
而從Wine 的架構圖中，比較起上述的Windows 9X 和 Windows NT 的系統架構，可以發現 Wine 更接近去實現Windows NT 的系統架構。 Wine 必須只少能夠完全取代三個 ( kernel32 DLL, GDI32 DLL, User32 DLL ) 動態鏈結函式庫的功能。 Wine 建構在 Unix shared libraries 上， Unix shared libraries 包含 libc 、 libx11 等函式庫， libc 為 Linux 下 ANSI C 函式庫， libx11 則是linux 系統下GUI的基礎，提供了客戶端應用程序在桌面上顯示數據所需的函式庫。

Wine 包含一個程式的 loader ，用來將程式及其相關的動態鏈結函式庫裝載到 linux 系統環境中，同時保留 windows 環境所需的上下文參數。Wine 對一些必須重新實現的 Windows 動態鏈結函式庫使用linux 函式庫來實現，也就是說 wine 是在程式執行時對 windows API 轉為 POSIX API，而不是使用模擬器來模擬系統環境。

Wine Drivers 用來支持 Windows 的設備驅動，也就是說 Wine 在Linux kernel上建立了一個提供Windows NTDLL 與 Kernel32 功能的代理層，這種設計方式決定了Windows程式在 Linux 中使用驅動必須滿足該設備在 Linux 中被支持或 Wine 已經實現了Windows驅動的API與Linux驅動接口黏合的代理功能。

Wine Executable 主要是加載 Windows 映像文件，並在 Linux kernel 中創建一個 Linux process來運行應用程式。

Wine server 主要在 Linux 用戶空間模仿 Windows 操作系統環境，也就是提供作業系統核心模擬的功能，為 Wine 提供與 Windows kernel 在 Windows 系統上相同的服務，提供自行實現的 Windows NT 動態鏈結庫來替代原有的 Windows NT 動態鏈結庫，而在 Windows 中的 win32 子系統的 API，大多數是直接轉發到 Windows NT 動態鏈結庫來處理的，而 Wine 提供了兼容的Kernel32 和 Windows NT 動態鏈結庫，所以就可以實現用戶級動態鏈結庫的兼容，如 GDI32 、 USER32等。 Wine server 另一個重點是提供 Windows process 和 thread 等運行時應用程式所需的各種服務請求，包括 process 之間的通訊、同步機制等。 當 Wine 啟動時，會隨之建立一個 Linux Socket，之後所有 Wine process 都會使用此 socket 與 Wine Server 溝通，當所有 Wine process 結束時 Wine Server 便會自動關閉。 Wine process 的 thread 都有各自的請求緩衝區，此緩衝區與 Wine Server 是共享的。當 thread 需要與其它 thread 或 process 同步時， Wine 會先填充請求緩衝區，並通過 socket 發送一段命令， Wine Server 根據此命令執行相關的操作，並在操作結果返回之前將 Wine Server 狀態設為等待。而一樣的，當 thread 需要請求 Win32 API 服務時，透過 socket 來通知 Wine server ， Wine server 執行相關的操作並在操作結果返回給 Windows thread 之前將其狀態設為等待。 Wine Server 是一個單獨的 Linux process ，每次起動任何 Windows 程式之前，都會被優先啟動，若出了問題也會在砍掉它後才重新起動，它沒有自己的 thread ，而是在一個大的無限循環中往復檢查用戶 process 是否發送了請求命令，因此內部不存在競態條件 ( race condition )。 所以簡單來說， Wine Server管理 Wine 啟動的各類進程，其功能相當於 Wine 的內核。



Wine server 在處理多個 Windows threads 請求服務時：

1. Windows thread 1 向 Wine server發出服務請求。

2. Wine server 收到請求後將thread 1 設為睡眠狀態，並處理其發出的請求， 若需要 Linux kernel 服務時，則再向 Linux kernel 發出請求。

3. 當 Wine server 在處理 thread 1的請求時，若有其他thread ( 如： thread 2 ) 發出服務請求，則將其設為睡眠狀態。

4. Wine server 完成 thread 1 的服務請求時，喚醒 thread 1 並處理 thread 2 的服務請求。

5. 當完成thread 2 的服務請求，喚醒 thread 2 以此類推來完成服務請求。 Wine server 輪尋所有 socket 連線並完成收到的服務請求。

而在 Wine 載入執行檔前，由於每個程式都需要使用自己的記憶體空間， Wine 要先依照不同的執行檔格式需求建立 Memory layout ，為了能夠在 Linux 中製造出跟 Win32 一樣的記憶體位址，除了試圖將相關的動態鏈結庫載入到正確的記憶體位址外， Wine 也要避免動態鏈結預先把 Wine 所需的函式庫配置到錯誤的記憶體位址，於是修改預設的 ELF ( Executable and Linking Format ) 初始化程序，以系統呼叫將相關的函式庫配置到正確的位址。

且在 Windows 中，每個 process 和 thread 會有一塊Environment Block ( PEB–Process Environment block or TEB – Thread Environment Block)，這些資料中包含了 TLS slots ( Thread Local Storage )、message queue、error code等資料，這塊資料結構提供錯誤處理等所需資訊，此外還包括SHE ( Structured Exception Handing )，用來拋出 “ This program has performed an illegal operation and will be terminated ” 等類似訊息。在建立 PEB / TEB 後，才初始化 process heap，載入執行檔，依照執行檔指示來建立 stack，最後才將控制權交給執行檔的 entrypoint 。上述這兩個工作均由 NTDLL 和 kernel32 來處理，NTDLL 和 kernel32 也會將執行續資訊傳遞給 Wine server。

由上述而言， Wine 實現 Windows NT 架構，因此提供 NTDLL，而實現 Win32 基礎的 Kernel32 則在NTDLL 上。總結來說 Wine 就是運用 API 轉換技術實作出 Linux 對應到 Windows 相對應得函數來調用動態鏈結函式庫以運行Windows程式。

## D、 DLL ( Dynamic-link library )

在 Wine 中，動態鏈結函式庫分為兩個部分，一個為 Wine builtin DLLs ，另一個為 Wine native DLLs，Wine builtin DLLs 也就是 Wine 提供的動態鏈結函式庫，Native DLLs 則是 Windows作業系統中的動態鏈結函式庫。

Wine builtin DLLs 包含了動態鏈結函式庫的 resource、file descriptor、constructor ，當此動態鏈結函式庫被載入記憶體時，該 constructor 被調用且在WINE DLL加載器（preloader）上註冊該DLL的 file descriptor 。 DLL 的 file descriptor 在記憶體中創建一個PE-header ( PE - Portable Executable) 用於提供DLL的入口點、資源、模塊的依賴性等信息。

Native DLLs保證實現 100% 的兼容性，但並非每個應用程式都在 native DLLs 下都有很好的表現，若函式庫試圖呼叫未在 Wine 中完全實現的函式功能，則 native DLLs 可能會比 builtin DLLs 的表現來得更糟，此時就必須轉而加載 builtin DLLs 。 Native DLLs 也有與上述 builtin DLLs 類似的結構， Wine 利用這些訊息來處理動態鏈結函式庫的導入與導出。當一個應用進程需要加載Native DLL時，它依次按下列方式查找：1.通過已註冊的DLL列表；2.根據WINEDLL-PATH的環境變量進行搜索；3.加載Windows原始的DLL。DLL實例化後被dlopen() 映射進記憶體中，dlopen() 是 Linux 中一個用來打開動態鏈結函式庫的函數。

從上述可知， Wine 可以使用自身提供的動態鏈結函式庫，也可以加載 Windows 系統中的動態鏈結函式庫，一切都已如何完美實現 Windows 程式為考量基準。

Wine 並沒有一個順序法則來決定 builtin DLLs 和 native DLLs 哪個優先使用，必需針對不同的程式測試來作出個別的加載順序。 Wine 允許用 native DLLs 來替換 builtin DLLs ， 但其中 kernel32.dll 、 GDI32.dll 、 User32.dll 、 NTDLL.dll這四個動態鏈結函式庫是不能用 native DLLs 來替換的，必須使用 Wine 提供的，也就是必須是 builtin DLLs。Wine對於動態鏈結函式庫的默認加載順序為，在一般情況下，Native DLLs 有優先權，但對於那些 Wine 能完全實現的所有 DLL 或者有已知有 Native DLLs 無法提供的動態鏈結函式庫的程式，則優先使用 builtin DLLs 來執行。

## E、 啟動 Wine

由於 Linux 早期版本提供的 thread API 不夠強大，不足以實現 Windows thread API 的部分， WINE 利用 Linux kernel thread 重新建構了 Windows thread API函數，此部分在 WINE 中稱為 WINE-kthread 。隨著 Linux 的發展， Linux 後來的版本中出現了功能強大的 NPTL 線程子系統， NPTL ( Native POSIX Thread Library ) 是一個能夠使使用POSIX Threads編寫的程式在Linux kernel 上更有效地執行的軟體。，WINE利用此系統提供的API函數黏合了Windows thread API 函數，此部分稱為WINE-pthread。 WINE在啟動前首先檢查用戶使用的 thread 模式並執行由 WINE-pthread 或 WINE-kthread 實現的WINE preloader。它是一個沒有 main 主函數的 ELF ( Executable and Linkable Format ) 文件，用於在新建立的WINE 進程中設置Win32類型的 process 地址空間佈局後轉入 WINE 的 main 函數， WINE 被啟動。 Preloader 通過這種方式傳遞一些必要信息到 libwine（WINE的動態庫）中，並調用函數 Wine-init 進行debug功能和地址空間方面的處理。以後順次執行：

1． WINE-process-init建立Win32 process運行所必需的結構，如 PEB、TEB、process heap 以及與 WINE server 的連接等。

2． Wine-kernel-init依次執行，從 WINE server 獲得新 process 啟動時必需的訊息；初始化新 process 的註冊以及其它相關部分；確定被加載的文件類型以後該應用程式被載入記憶體中。

3．新載入Windows process運行的開始。首先，start-process建立SHE，調用NtdllL dr-Initializer Thuck 執行 process 初始化的其他部分，如進行重定位以及調用動態鏈結函式庫的構造函數等。最後，執行轉入被載入應用程序的入口點，應用程序開始執行，WINE提供的API就可以被正確地調用了，隨後當執行從應用程序返回時，調用 ExitProcess 函數結束該進程。

## F、 啟動 Win32

從命令行啟動一個 Wine process 時， Wine 首先檢查作業系統使用的 thread 模式，啟動 Wine preloader 。 Wine 從 Linux 中取得諸如運行環境、命令行參數等必要的資源後依次執行下列動作：

1． NTDLL.dll 被Linux的標準共享庫裝載器載入內存，在Linux中建立PEB、TEB、進程堆以及與WINE Server的連接等。

2．利用Windows動態裝載能力加載 Kernel32 ， WINE-kernel-init 處理所有裝載與執行的邏輯過程，包括初始化命令行參數，在文件系統中尋找相應的可執行文件。如果沒有找到， Wine 裝載器停止裝載並返回出錯信息；否則，將找到的文件載入內存，建立一個 heap 供新建立的可執行文件的 process作用， heap大小由該文件結構中PE-header定義。

3． NtdllL dr-Initializer Thuck 執行其他部分如解析所有在可執行文件中引用的 DLL 與 TLS 槽的初始化。

4．執行轉入到可執行文件的入口點，可執行文件開始運行。

## G、 在 Ubuntu 18.04 安裝 Wine

1. 安裝 Wine

FOR 64bit : $ sudo apt install wine64

FOR 32bit : $ sudo apt install wine32

2. 檢查安裝

wine --version

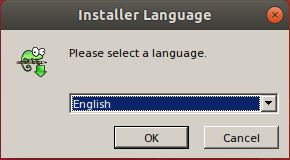
## H、 Wine 軟體測試

### 測試 Notepad++

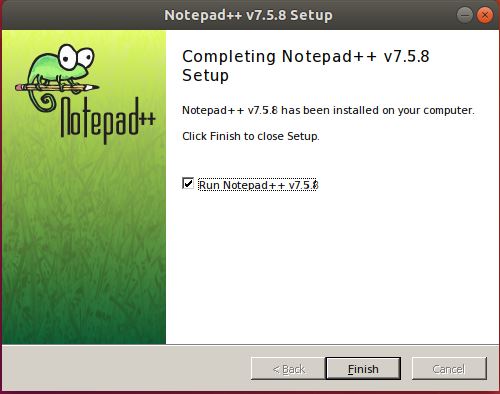
1. 從 Notepad++ 官方網站下在 Notepad++ 執行檔 ( npp.7.5.8.Installer.exe )

2. 利用 wine 安裝和開啟 Notepad++

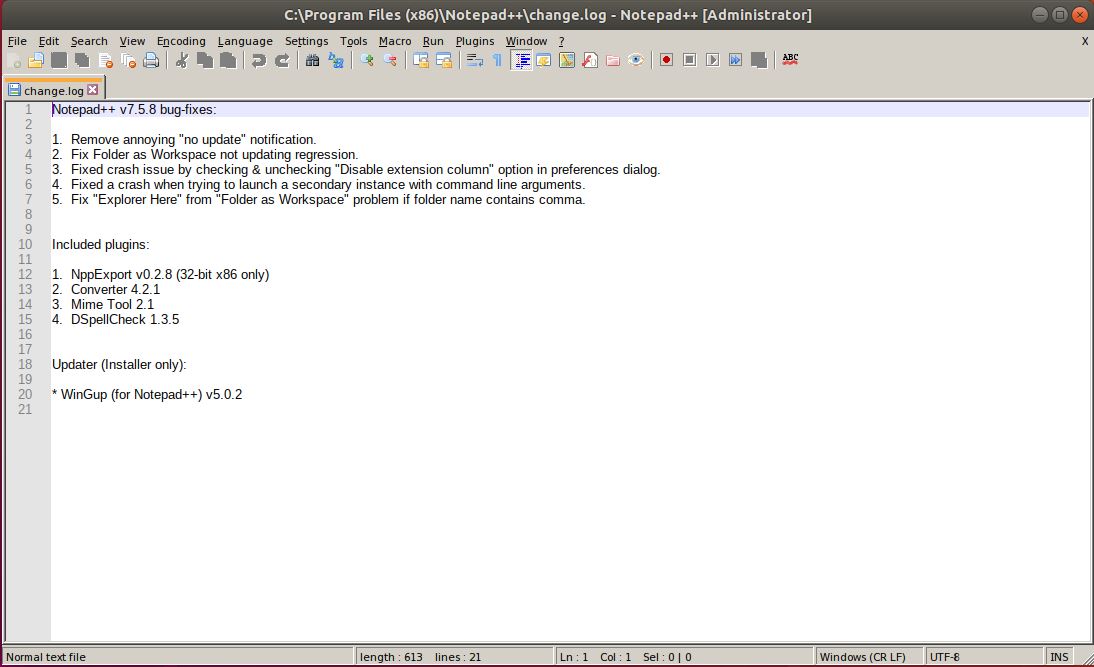
wine npp.7.5.8.Installer.exe



開始安裝 Notepad++



安裝 Notepad++ 完成



開始執行 Notepad++

## I. 參考資料

a. 書籍

1. Linux企業應用案例精解 第8章 虛擬化技術應用

作者：李晨光 出版社：清華大學出版社 出版日期：2012年4月

b. 網頁

1. [Wine Developer's Guide](https://wiki.winehq.org/Wine_Developer%27s_Guide)

2. [IBM 體驗 Wine Windows 到 Linux 的轉換](https://www.ibm.com/developerworks/cn/linux/l-wine/index.html)

3. [Wine – 在 Linux 中使用 Windows 程式](http://blog.nutsfactory.net/2011/04/07/wine-using-windows-program-in-linux/)

4. [基於龍芯的Windows軟件運行環境的設計與實現](http://journal.just.edu.cn/jweb_zkb/article/2014/1673-4807-28-4-387.html)

5. [Wine開發入門](https://www.jianshu.com/p/a907dae3a78d)

6. [Wine中PE格式文件的加載](https://blog.csdn.net/chrisnotfound/article/details/79957441)