

# PacVis: 可視化 pacman 本地數據 庫

---

## 目錄

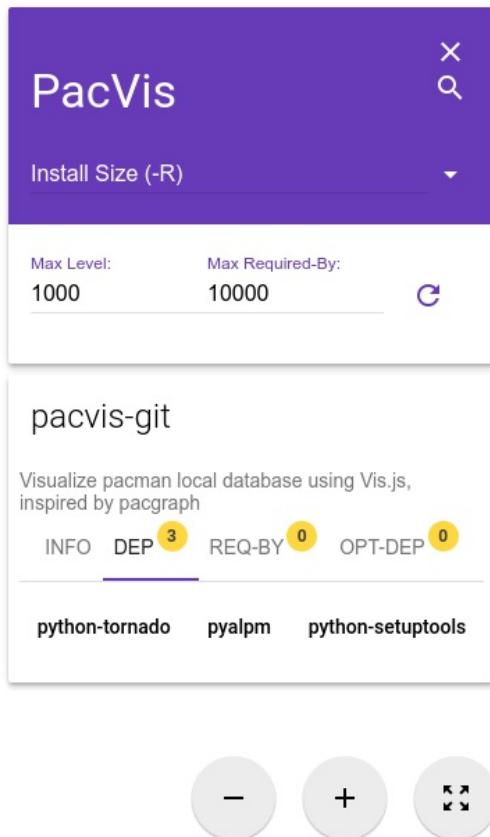
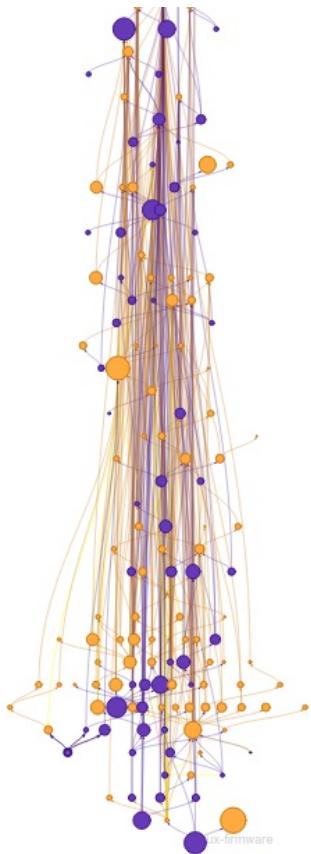
### Contents

- 我為什麼要做 PacVis
- PacVis的老前輩們
  - pactree
  - pacgraph

- 於是就有了 PacVis
- PacVis 的圖例和用法
- 從 PacVis 能瞭解到的一些事實
  - 依賴層次
  - 循環依賴
  - 有些包沒有依賴關係
  - 只看依賴關係的話 Linux 內核完全不重要
  - pacman -Qtd 不能找到帶有循環依賴的孤兒包
- PacVis 的未來

---

## PacVis



# 我為什麼要做 PacVis

我喜歡 Arch Linux，大概是因為唯有 Arch Linux 能給我對整個系統「瞭如指掌」的感覺。在 Arch Linux 裏我能清楚地知道我安裝的每一個包，能知道系統裏任何一個文件是來自哪個包，以及我為什麼要裝它。或許對 Debian/Fedora/openSUSE 足夠熟悉了之後也能做到這

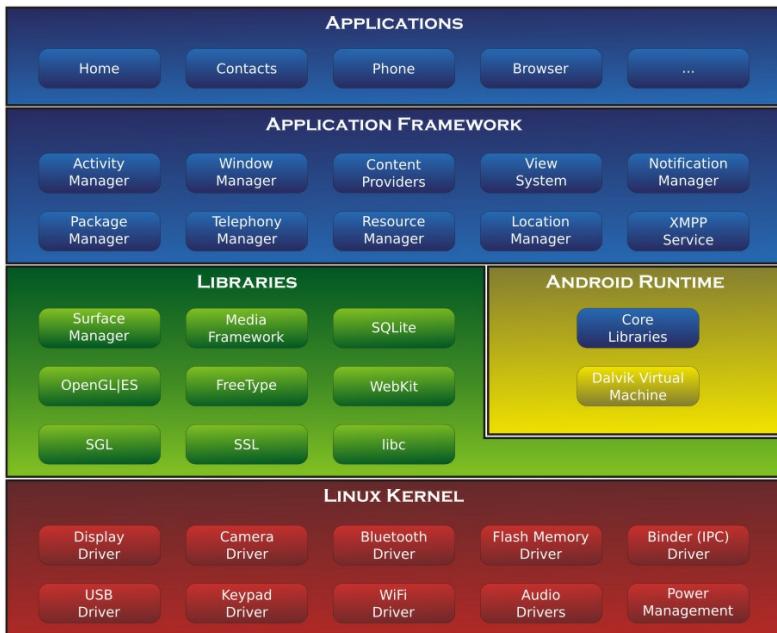
兩點，不過他們的細緻打包的結果通常是包的數量比 Arch 要多個 3 到 10 倍，並且打包的細節也比 Arch Linux 簡單的 PKGBUILD 要複雜一個數量級。

每一個裝過 Arch Linux 的人大概都知道，裝了 Arch Linux 之後得到的系統非常樸素，按照 ArchWiki 上的流程一路走下來的話，最關鍵的一條命令就是 `pacstrap /mnt base`，它在 `/mnt` 裏作為根調用 `pacman -S base` 裝上了整個 `base` 組，然後就沒有然後了。這個系統一開始空無一物，你需要的任何東西都是後來一點點用 `pacman` 手動裝出來的，沒有累贅，按你所需。

然而時間長了，系統中難免會有一些包，是你裝過用過然後忘記了，然後這些包就堆在系統的角落裏，就像家裏陳年的老傢俱，佔着地，落着灰。雖然 `pacman -Qtd` 能方便地幫你找出所有 **曾經作為依賴被裝進來，而現在不被任何包依賴** 的包，但是對於那些你手動指定的包，它就無能為力了。

於是我就一直在找一個工具能幫我梳理系統中包的關係，方便我：

1. 找出那些曾經用過而現在不需要的包
2. 找出那些體積大而且佔地方的包
3. 肆清系統中安裝了的包之間的關係



### Android 系統架構

關於最後一點「釐清包的關係」，我曾經看到過 macOS 系統架構圖 和 Android 的系統架構圖，對其中的層次化架構印象深刻，之後就一直在想，是否能畫出現代 Linux 桌面系統上類似的架構圖呢？又或者 Linux 桌面系統是否會展現完全不同的樣貌？從維基百科或者別的渠道能找到 Linux 內核、或者 Linux 圖形棧，或者某個桌面環境的架構，但是沒有找到覆蓋一整個發行版的樣貌的。於是我想，能不能從包的依賴關係中自動生成這樣一張圖呢。

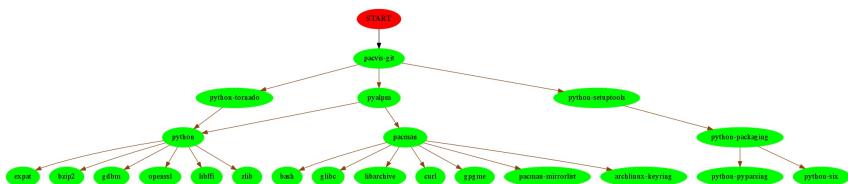
# PacVis的老前輩們

在開始寫 PacVis 之前，我試過一些類似的工具，他們都或多或少能解決一部分我的需要，又在某些方面有所不足。這些工具成爲了 PacVis 的雛形，啓發了 PacVis 應該做成什麼樣子。

# pactree

pactree 曾經是一個獨立的項目，現在則是 pacman 的一部分了。從手冊頁可以看出，pactree 的輸出是由某個包開始的依賴樹。加上 `--graph` 參數之後 pactree 還能輸出 dot 格式的矢量圖描述，然後可以用 dot 畫出依賴圖：

```
pactree pacvis-git -d3 --graph | dot -Tpng >pacvis-pactree.png
```



```
1 $ pactree pacvis-git -d3
2 pacvis-git
3   |-python-tornado
4     \-python
5       |-expat
6       |-bzip2
7       |-gdbm
8       |-openssl
9       |-libffi
10      \-zlib
11   |-pyalpm
12     |-python
13     \-pacman
14       |-bash
15       |-glibc
16       |-libarchive
17       |-curl
18       |-gpgme
19       |-pacman-mirrorlist
20         \-archlinux-keyring
21   \-python-setuptools
22     \-python-packaging
23       |-python-pyparsing
24         \-python-six
25 $ pactree pacvis-git -d3 --graph |
dot -Tpng >pacvis-pactree.png
```

從畫出的圖可以看出，因為有共用的依賴，所以從一個包開始的依賴關係已經不再是一棵 圖論意義上的樹 (Tree) 了。最初嘗試做 PacVis 的早期實現的時候，就是

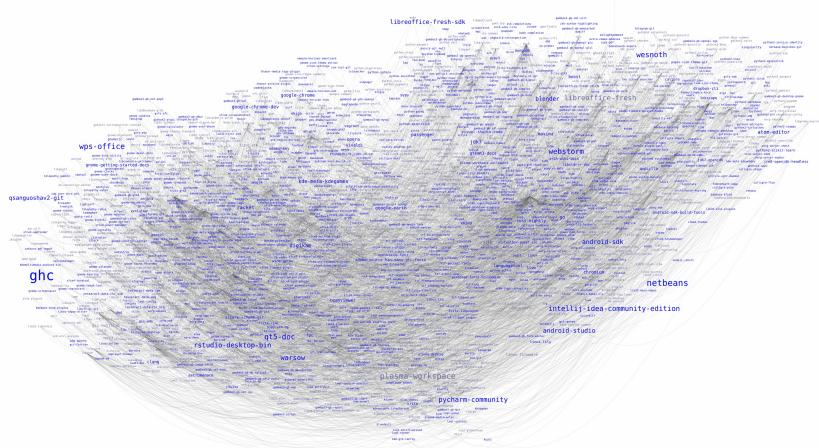
試圖用 bash/python 腳本解析 pactree 和 pacman 的輸出，在 pactree 的基礎上把整個系統中所有安裝的包全都包含到一張圖裏。當然後來畫出的結果並不那麼理想，首先由於圖非常巨大，導致 dot 的自動佈局要耗費數小時，最後畫出的圖也過於巨大基本上沒法看。

然而不得不說沒有 pactree 就不會有 PacVis，甚至 pacman 被分離出 alpm 庫也和 pactree 用 C 重寫的過程有很大關係，而 PacVis 用來查詢 pacman 數據庫的庫 pyalpm 正是 alpm 的 Python 綁定。因為 pactree 的需要而增加出的 alpm 庫奠定了 PacVis 實現的基石。

## pacgraph

pacgraph 的輸出

---



pacgraph 是一位 Arch Linux 的 Trusted User keenerd 寫的程序，和 PacVis 一樣也是用 Python 實現的。比起 pactree，pacgraph 明顯更接近我的需求，它默認繪製整個系統的所有安裝包，並且用聰明的佈局算法解決了 dot 佈局的性能問題。

`pacgraph` 的輸出是一個富有藝術感的依賴圖，圖中用不同的字體大小表示出了每個包佔用的磁盤空間。通過觀察 `pacgraph` 的輸出，我們可以清楚地把握系統全局的樣貌，比如一眼看出這是個桌面系統還是個服務器系統，並且可以很容易地發現那些佔用磁盤空間巨大的包，考慮清理這些包以節約空間。

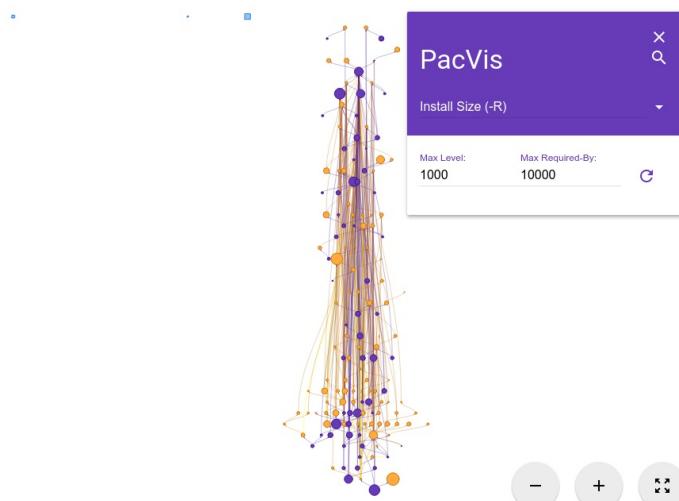
更棒的是 pacgraph 還提供了一個交互性的 GUI 叫做 pacgraph-tk，顯然通過 tk 實現。用這個 GUI 可以縮放觀察整幅圖的細節，或者選中某個包觀察它和別的包的依賴關係。

pacgraph 還支持通過參數指定只繪製個別包的依賴關係，就像 pactree 那樣。

不過 pacgraph 也不是完全滿足我的需要。如我前面說過，我希望繪製出的圖能反應 **這個發行版的架構面貌**，而 pacgraph 似乎並不區別「該包依賴的包」和「依賴該包的包」這兩種截然相反的依賴關係。換句話說 pacgraph 畫出的是一張無向圖，而我更想要一張有向圖，或者說是 **有層次結構的依賴關係圖**。

## 於是就有了 PacVis

PacVis 剛打開的樣子



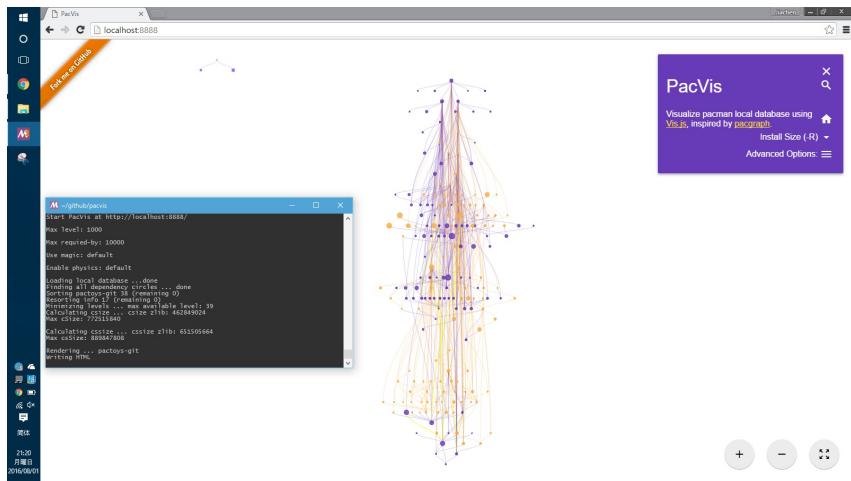
總結了老前輩們的優勢與不足，我便開始利用空餘時間做我心目中的 PacVis。前後斷斷續續寫了兩個月，又分為兩個階段，第一階段做了基本的功能和雛形，第二階段套用上 <https://getmdl.io/> 的模板，總算有了能拿得出手給別人看的樣子。

於是乎前兩天在 AUR 上給 pacvis 打了個 pacvis-git 包，現在想在本地跑 pacvis 應該很方便了，用任何你熟悉的 aurhelper 就可以安裝，也可以直接從 aur 下載 PKGBUILD 打包：

```
1 ~$ git clone aur@aur.archlinux.org:pacvis-git.git
2 ~$ cd pacvis-git
3 ~/pacvis-git$ makepkg -si
4 ~/pacvis-git$ pacvis
5 Start PacVis at http://localhost:8888
8/
```

按照提示說的，接下來打開瀏覽器訪問 <http://localhost:8888/> 就能看到 PacVis 的樣子了。僅僅作為嘗試也可以直接打開跑在我的服務器上的 demo: <https://pacvis.farseerfc.me/>，這個作為最小安裝的服務器載入速度大概比普通的桌面系統快一點。

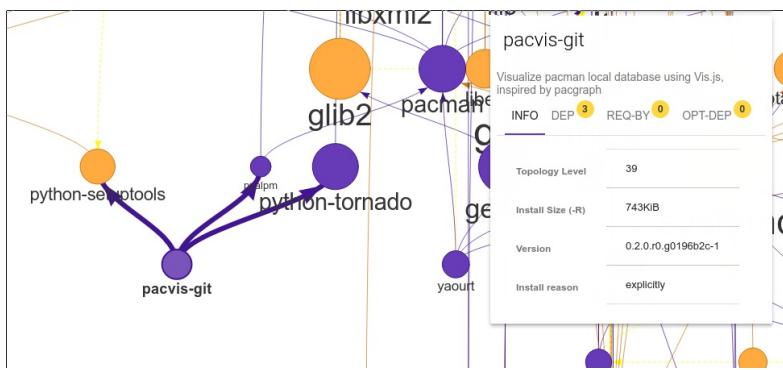
在 Windows msys2 跑 PacVis



另外補充一下，因為 PacVis 只依賴 pyalpm 和 tornado，所以在別的基於 pacman 的系統上跑它應該也沒有任何問題，包括 Windows 上的 msys2 裏（儘管在 msys2 上編譯 tornado 的包可能要花些功夫）。

## PacVis 的圖例和用法

操作上 PacVis 仿照地圖程序比如 Google Maps 的用法，可以用滾輪或者觸摸屏的手勢 縮放、拖拽，右上角有個側邊欄，不需要的話可以點叉隱藏掉，右下角有縮放的按鈕和回到全局視圖的按鈕，用起來應該還算直觀。



*pacvis-git* 包的依賴

先解釋圖形本身，整張圖由很多小圓圈的節點，以及節點之間的箭頭組成。一個圓圈就代表一個軟件包，而一條箭頭代表一個依賴關係。縮放到細節的話，能看到每個小圓圈的下方標註了這個軟件包的名字，鼠標懸浮在圓圈上也會顯示響應信息。還可以點開軟件包，在右側的邊欄裏會有更詳細的信息。

比如圖例中顯示了 pacvis-git 自己的依賴，它依賴 pyalpm, python-tornado 和 python-setuptools，其中 pyalpm 又依賴 pacman。圖中用 **紫色** 表示手動安裝的包，**橙色** 表示被作為依賴安裝的包，箭頭的顏色也隨着包的顏色改變。

值得注意的是圖中大多數箭頭都是由下往上指的，這是因為 PacVis 按照包的依賴關係做了拓撲排序，並且給每個包賦予了一個拓撲層級。比如 pacvis-git 位於 39 層，那麼它依賴的 pyalpm 就位於 38 層，而 pyalpm 依賴的 pacman 就位於 37 層。根據層級關係排列包是 PacVis 與 pacgraph 之間最大的不同之處。

除了手動縮放， PacVis 還提供了搜索框，根據包名快速定位你感興趣的包。以及在右側邊欄中的 Dep 和 Req-By 等頁中，包的依賴關係也是做成了按鈕的形式，可以由此探索包和包之間的關聯。

最後稍微解釋一下兩個和實現相關的參數：

#### **Max Level**

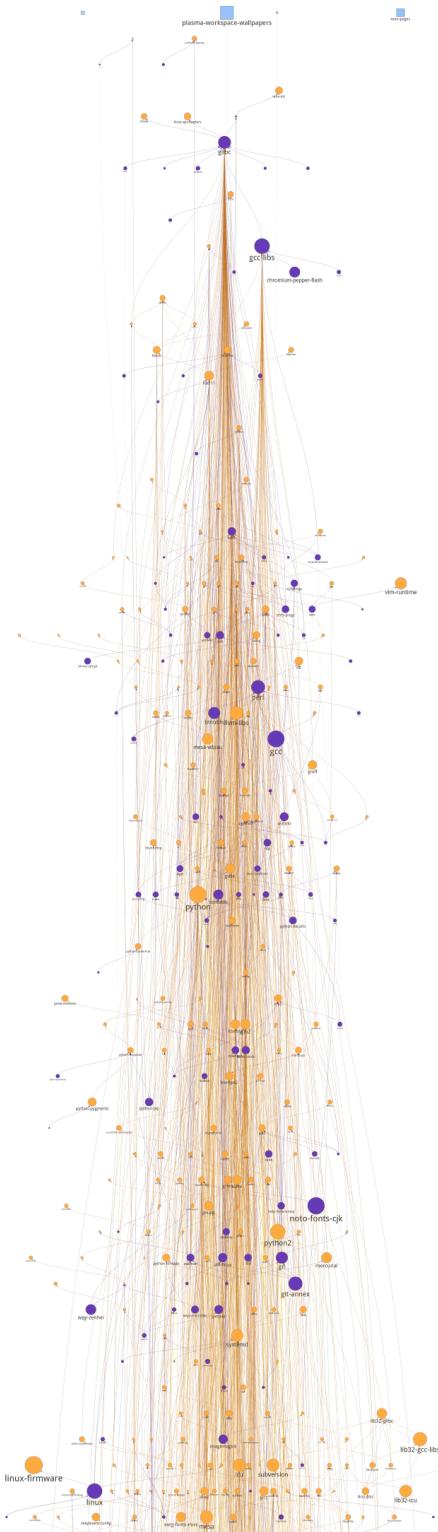
這是限制 PacVis 載入的最大拓撲層。系統包非常多的時候 PacVis 的佈局算法會顯得很慢，限制層數有助於加快載入，特別是在調試 PacVis 的時候比較有用。

#### **Max Required-By**

這是限制 PacVis 繪製的最大被依賴關係。稍微把玩一下 PacVis 就會發現系統內絕大多數的包都直接依賴了 glibc 或者 gcc-libs 等個別的幾個包，而要繪製這些依賴的話會導致渲染出的圖中有大量長直的依賴線，不便觀察。於是可以通过限制這個值，使得 PacVis 不繪製被依賴太多的包的依賴關係，有助於讓渲染出的圖更易觀察。

## 從 PacVis 能瞭解到的一些事實

一個 KDE 桌面的 PacVis 結果全圖， 放大 (17M)



glibc

gcc-libs

libx11

bash

vim

perl

gcc

guile

python coreutils

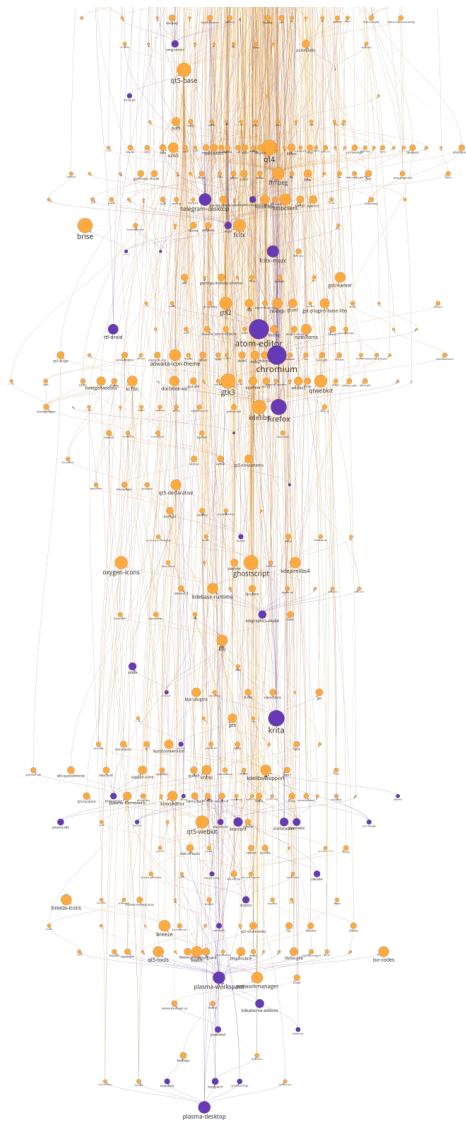
glib2

freetype2

pacman

systemd

linux



qt5-base

qt4

gtk2

chromium

gtk3

firefox

qt5-declarative

kdegraphics-okular

sddm

krita

plasma-framework

plasma-workspace

plasma-desktop

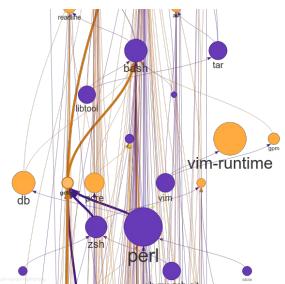
稍微玩一下 PacVis 就能發現不少有趣現象，上述「絕大多數包依賴 glibc」就是一例。除此之外還有不少值得玩味的地方。

# 依賴層次

系統中安裝的包被明顯地分成了這樣幾個層次：

- glibc 等 C 庫
- Bash/Perl/Python 等腳本語言
- coreutils/gcc/binutils 等核心工具
- pacman / systemd 等較大的系統工具
- gtk{2,3}/qt{4,5} 等 GUI toolkit
- chromium 等 GUI 應用
- Plasma/Gnome 等桌面環境

大體上符合直觀的感受，不過細節上有很多有意思的地方，比如 zsh 因為 gdbm 間接依賴了 bash，這也說明我們不可能在系統中用 zsh 完全替代掉 bash。再比如 python（在 Arch Linux 中是 python3）和 python2 和 pypy 幾乎在同一個拓撲層級。

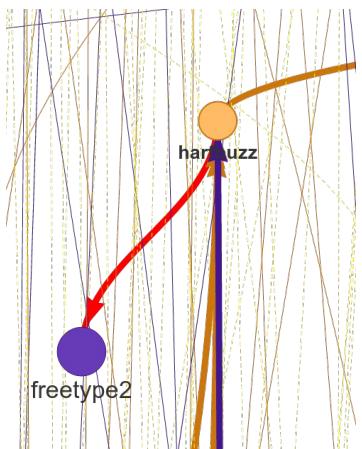


*zsh 因為 gdbm 間接依賴了 bash*

不過偶爾顯示的依賴層級不太符合直觀，比如 qt5-base < qt4 < gtk2 < gtk3。qt5 因為被拆成了數個包所以比 qt4 更低級這可以理解，而 gtk 系比 qt 系更高級這一點是很多人（包括我）沒有預料到的吧。

# 循環依賴

有些包的依賴關係形成了循環依賴，一個例子是 freetype2 和 harfbuzz，freetype2 是繪製字體的庫，harfbuzz 是解析 OpenType 字形的庫，兩者對對方互相依賴。另一個例子是 KDE 的 kio 和 kinit，前者提供類似 FUSE 的資源訪問抽象層，後者初始化 KDE 桌面環境。



*freetype2 和 harfbuzz 之間的循環依賴*

因為這些循環依賴的存在，使得 PacVis 在實現時不能直接拓撲排序，我採用環探測 算法找出有向圖中所有的環，並且打破這些環，然後再使用拓撲排序。因此我在圖中用紅色的箭頭表示這些會導致環的依賴關係。

# 有些包沒有依賴關係



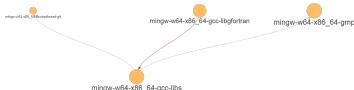
*man-pages* 和 *licenses* 沒有依賴關係

有些包既不被別的包依賴，也不依賴別的包，而是孤立在整張圖中，比如 *man-pages* 和 *licenses*。這些包在圖中位於最頂端，拓撲層級是 0，我用 藍色 正方形特別繪製它們。

## 只看依賴關係的話 Linux 內核完全不重要

所有用戶空間的程序都依賴着 glibc，而 glibc 則從定義良好的 syscall 調用內核。因此理所當然地，如果只看用戶空間的話，glibc 和別的 GNU 組件是整個 GNU/Linux 發行版的中心，而 Linux 則是位於依賴層次中很深的位置，甚至在我的 demo 服務器上 Linux 位於整個圖中的最底端，因為它的安裝腳本依賴 mkinitcpio 而後者依賴了系統中的衆多組件。

## pacman -Qtd 不能找到帶有循環依賴的孤兒包



*msys2* 中帶有循環依賴的孤兒包

這是在 msys2 上測試 PacVis 的時候發現的，我看  
到在渲染的圖中有一片羣島，沒有連上任何手動安裝的  
包。這種情況很不正常，因為我一直在我的所有系統中  
跑 pacman -Qtd 找出孤兒包並刪掉他們。放大之後我  
發現這些包中有一條循環依賴，這說明 pacman -Qtd  
不能像語言的垃圾回收機制那樣找出有循環依賴的孤兒  
包。

## PacVis 的未來

---

目前的 PacVis 基本上是我最初開始做的時候設想的  
樣子，隨着開發逐漸又增加了不少功能。一些是迫於佈  
局算法的性能而增加的（比如限制層數）。

今後準備再加入以下這些特性：

1. 更合理的 optdeps 處理。目前只是把 optdeps 關係在圖上畫出來了。
2. 更合理的 **依賴關係抉擇**。有時候包的依賴關係並  
不是直接根據包名，而是 provides 由一個包提  
供另一個包的依賴。目前 PacVis 用 alpm 提供的  
方式抉擇這種依賴，於是這種關係並沒有記錄在圖  
上。
3. 目前的層級關係沒有考慮包所在的倉庫  
(core/extracommunity/...) 或者包所屬的組。加  
入這些關係能更清晰地表達依賴層次。

4. 目前沒有辦法只顯示一部分包的關係。以後準備加入像 pactree/pacgraph 一樣顯示部分包。

如果你希望 PacVis 出現某些有趣的用法和功能，也請給我提 issue。

---