从零开始构建 Linux 发行版

杨瑒 Adam

上海大学
开源社区

目录

- 枯燥的构建基础与理不清的细节
- 包管理器、依赖树与发行版
- 尬聊时光

枯燥的构建基础与理不清的细节

枯燥的构建基础与理不清的细节

- 编译器： gcc, g++
- 辅助构建工具
  - autotools, Makefile
  - CMake, Meson, ...
- 脚本语言
  - shell (bash)
  - sed, awk
  - perl
  - python

枯燥的构建基础与理不清的细节

- 我们曾以为……

```
./configure --prefix=/usr
make -j$(nproc)
make install
```

枯燥的构建基础与理不清的细节

- 我们曾以为……

```
./configure --prefix=/usr
make -j$(nproc)
make install
```

Too Young, Too Naive

枯燥的构建基础与理不清的细节

- 现实情况是……

```
#!/bin/bash

# Check whether currently is in chroot env
[ -d "${ROOTFS_DIR}" ] && {
    (>&2 echo "Executing chroot failed! Exit!")
    exit 1
}

# Configure and build
cd /_source/

# Create a compatibility symlink to avoid references to /$TCNAME in our final glibc
ln -sfv "/${TCNAME}/lib/gcc" /usr/lib

case $(uname -m) in
  i?86)
    GCC_INCDIR=/usr/lib/gcc/$(uname -m)-pc-linux-gnu/${GCC}/include
    ln -sfv ld-linux.so.2 /lib/ld-lsb.so.3
  ;;
  x86_64)
    GCC_INCDIR=/usr/lib/gcc/x86_64-pc-linux-gnu/${GCC}/include
    ln -sfv ../lib/ld-linux-x86-64.so.2 /lib64
    ln -sfv ../lib/ld-linux-x86-64.so.2 /lib64/ld-lsb-x86-64.so.3
  ;;
esac

rm -f /usr/include/limits.h

# Configure and build
rm -rf /_build && mkdir -p /_build
cd /_build

CC="gcc -isystem ${GCC_INCDIR} -isystem /usr/include" \
  /_source/configure                                  \
    --prefix=/usr                                     \
    --disable-werror                                  \
    --enable-kernel=3.2                               \
    --enable-stack-protector=strong                   \
    libc_cv_slibdir=/lib

unset GCC_INCDIR

make -j$(nproc)

case $(uname -m) in
  i?86)   ln -sfnv "${PWD}/elf/ld-linux.so.2"        /lib ;;
  x86_64) ln -sfnv "${PWD}/elf/ld-linux-x86-64.so.2" /lib ;;
esac

# Failures to be ignored:
# - misc/tst-ttyname is known to fail in the LFS chroot environment.
# - inet/tst-idna_name_classify is known to fail in the LFS chroot environment.
# - posix/tst-getaddrinfo4 and posix/tst-getaddrinfo5 may fail on some architectures.
# - nss/tst-nss-files-hosts-multi test may fail for reasons that have not been determined
# - rt/tst-cputimer{1,2,3} tests depend on the host system kernel.
#   Kernels 4.14.91–4.14.96, 4.19.13–4.19.18, and 4.20.0–4.20.5 are known
#   to cause these tests to fail.
# - The math tests sometimes fail when running on systems where the CPU is not a
#   relatively new Intel or AMD processor.
make check

# Create dynamic linker configuration
touch /etc/ld.so.conf

# Skip 'test-installation.pl' test
sed '/test-installation/s@$(PERL)@echo not running@' -i /_source/Makefile

# Install glibc
make install
```

枯燥的构建基础与理不清的细节

- 现实情况是……
- 为什么会出现这样的问题？

```
#!/bin/bash

# Check whether currently is in chroot env
[ -d "${ROOTFS_DIR}" ] && {
    (>&2 echo "Executing chroot failed! Exit!")
    exit 1
}

# Configure and build
cd /_source/

# Create a compatibility symlink to avoid references to /$TCNAME in our final glibc
ln -sfv "/${TCNAME}/lib/gcc" /usr/lib

case $(uname -m) in
  i?86)
    GCC_INCDIR=/usr/lib/gcc/$(uname -m)-pc-linux-gnu/${GCC}/include
    ln -sfv ld-linux.so.2 /lib/ld-lsb.so.3
  ;;
  x86_64)
    GCC_INCDIR=/usr/lib/gcc/x86_64-pc-linux-gnu/${GCC}/include
    ln -sfv ../lib/ld-linux-x86-64.so.2 /lib64
    ln -sfv ../lib/ld-linux-x86-64.so.2 /lib64/ld-lsb-x86-64.so.3
  ;;
esac

rm -f /usr/include/limits.h

# Configure and build
rm -rf /_build && mkdir -p /_build
cd /_build

CC="gcc -isystem ${GCC_INCDIR} -isystem /usr/include" \
  /_source/configure                                  \
    --prefix=/usr                                     \
    --disable-werror                                  \
    --enable-kernel=3.2                               \
    --enable-stack-protector=strong                   \
    libc_cv_slibdir=/lib

unset GCC_INCDIR

make -j$(nproc)

case $(uname -m) in
  i?86)   ln -sfnv "${PWD}/elf/ld-linux.so.2"        /lib ;;
  x86_64) ln -sfnv "${PWD}/elf/ld-linux-x86-64.so.2" /lib ;;
esac

# Failures to be ignored:
# - misc/tst-ttyname is known to fail in the LFS chroot environment.
# - inet/tst-idna_name_classify is known to fail in the LFS chroot environment.
# - posix/tst-getaddrinfo4 and posix/tst-getaddrinfo5 may fail on some architectures.
# - nss/tst-nss-files-hosts-multi test may fail for reasons that have not been determined
# - rt/tst-cputimer{1,2,3} tests depend on the host system kernel.
#   Kernels 4.14.91–4.14.96, 4.19.13–4.19.18, and 4.20.0–4.20.5 are known
#   to cause these tests to fail.
# - The math tests sometimes fail when running on systems where the CPU is not a
#   relatively new Intel or AMD processor.
make check

# Create dynamic linker configuration
touch /etc/ld.so.conf

# Skip 'test-installation.pl' test
sed '/test-installation/s@$(PERL)@echo not running@' -i /_source/Makefile

# Install glibc
make install
```

枯燥的构建基础与理不清的细节

我们对你们提供的技术方案进行了深入的探讨，
对其感到非常惊艳，但很抱歉不能将其纳入。
其中最主要的原因是，我们不仅仅需要支持 x86, x64 平台，
还同时需要支持 ARM, ARM64, MIPS, PowerPC, Sparc 等等平台。

枯燥的构建基础与理不清的细节

什么？

你就因为那 40亿 个用手机
的人把我的方案拒绝了？

大家都不爱遵守的 Linux Standard Base 标准

- LSB 标准
- https://refspecs.linuxfoundation.org/lsb.shtml

The LSB defines a binary interface for application programs that are compiled and packaged for LSB-conforming implementations on many different hardware architectures.

大家都不爱遵守的 Linux Standard Base 标准

The LSB defines

a binary interface (ABI, for differenct software)

for application programs that are compiled and packaged for LSB-conforming implementations

on many different hardware

architectures.

大家都不爱遵守的 Linux Standard Base 标准

“... Some in the Debian project are questioning the value of maintaining LSB compliance—it has become, they say, a considerable amount of work for little measurable benefit.”

“I've held an LSB BoF last year (2014) at DebConf, and discussed src:lsb with various people back then, and what I took back was 'roughly no one cares'.”

大家都不爱遵守的 Linux Standard Base 标准

- 软件库 **头文件接口** 与其软链接**名称**
- ELF 格式
  - Object format
  - 调试信息格式
  - …
- libc system call
- 文件与目录的用户权限
- 部分语言与地区配置
- 系统初始化、系统服务接口

大家都不爱遵守的 Linux Standard Base 标准

Library	Runtime Name	Library	Runtime Name
libGL	libGL.so.1	libXext	libXext.so.6
libGLU	libGLU.so.1	libXft	libXft.so.2
libICE	libICE.so.6	libXi	libXi.so.6
libQtCore	libQtCore.so.4	libXrender	libXrender.so.1
libQtGui	libQtGui.so.4	libXt	libXt.so.6
libQtNetwork	libQtNetwork.so.4	libXtst	libXtst.so.6
libQtOpenGL	libQtOpenGL.so.4	libasound	libasound.so.2
libQtSql	libQtSql.so.4	libatk-1.0	libatk-1.0.so.0
libQtSvg	libQtSvg.so.4	libcairo	libcairo.so.2
libQtXml	libQtXml.so.4	libcairo-gobject	libcairo-gobject.so.2
libSM	libSM.so.6	libfreetype	libfreetype.so.6
libX11	libX11.so.6	...	...

大家都不爱读字的 Linux Standard Base 标准

```
#define EOF (-1)
#define P_tmpdir    "/tmp"
#ifndef SEEK_SET
#define SEEK_SET    0
#endif
#ifndef SEEK_CUR
#define SEEK_CUR    1
#endif
#define FOPEN_MAX   16
#ifndef SEEK_END
#define SEEK_END    2
#endif

// ... dozens of lines ...

typedef struct {
    off_t __pos;
    mbstate_t __state;
} fpos_t;
typedef struct {
    off64_t __pos;
    mbstate_t __state;
} fpos64_t;

typedef struct _IO_FILE FILE;

// ... dozens of lines ...

extern int asprintf(char **__ptr, const char *__fmt, ...);
extern void clearerr(FILE * __stream);
extern void clearerr_unlocked(FILE * __stream);
extern int dprintf(int __fd, const char *__fmt, ...);
extern int fclose(FILE * __stream);
extern FILE *fdopen(int __fd, const char *__modes);
extern int feof(FILE * __stream);
extern int feof_unlocked(FILE * __stream);
extern int ferror(FILE * __stream);
extern int ferror_unlocked(FILE * __stream);
extern int fflush(FILE * __stream);
extern int fflush_unlocked(FILE * __stream);
extern int fgetc(FILE * __stream);
extern int fgetc_unlocked(FILE * __stream);
extern int fgetpos(FILE * __stream, fpos_t * __pos);
extern int fgetpos64(FILE * __stream, fpos64_t * __pos);
extern char *fgets(char *__s, int __n, FILE * __stream);

// ... dozens of lines ...
```

大家都不爱遵守的 Linux Standard Base 标准

LSB 4.1 release consisted of "1493 components, 1672 libs, 38491 commands, 30176 classes and 716202 interfaces"

... and ...

LSB 5.0 is even bigger

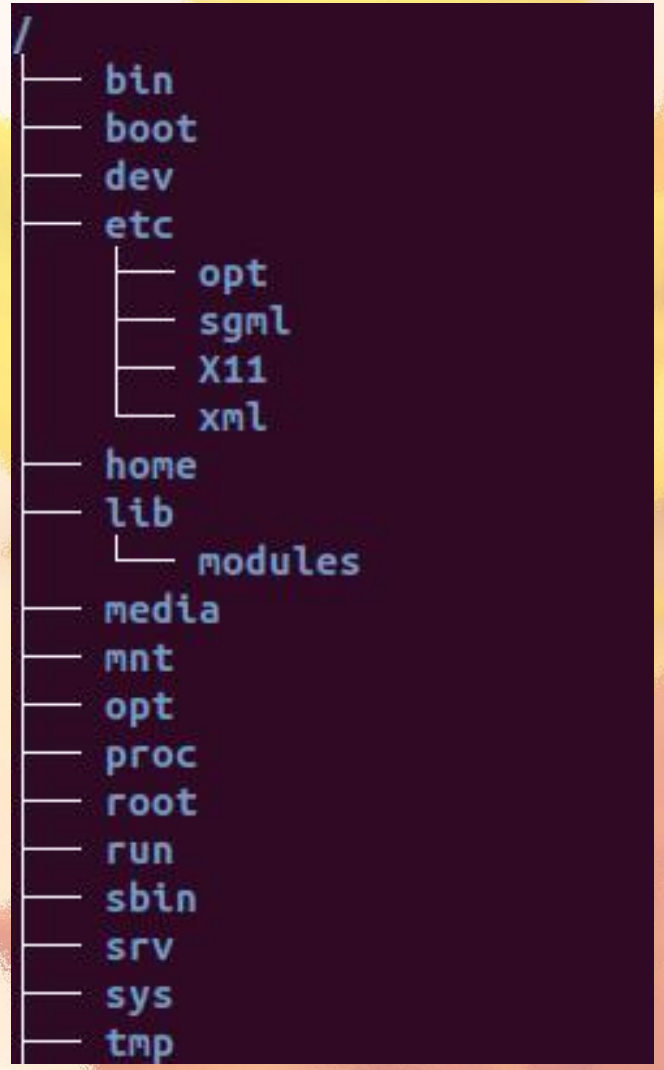
好像知道又似是而非的 Filesystem Hierarchy Standard 标准

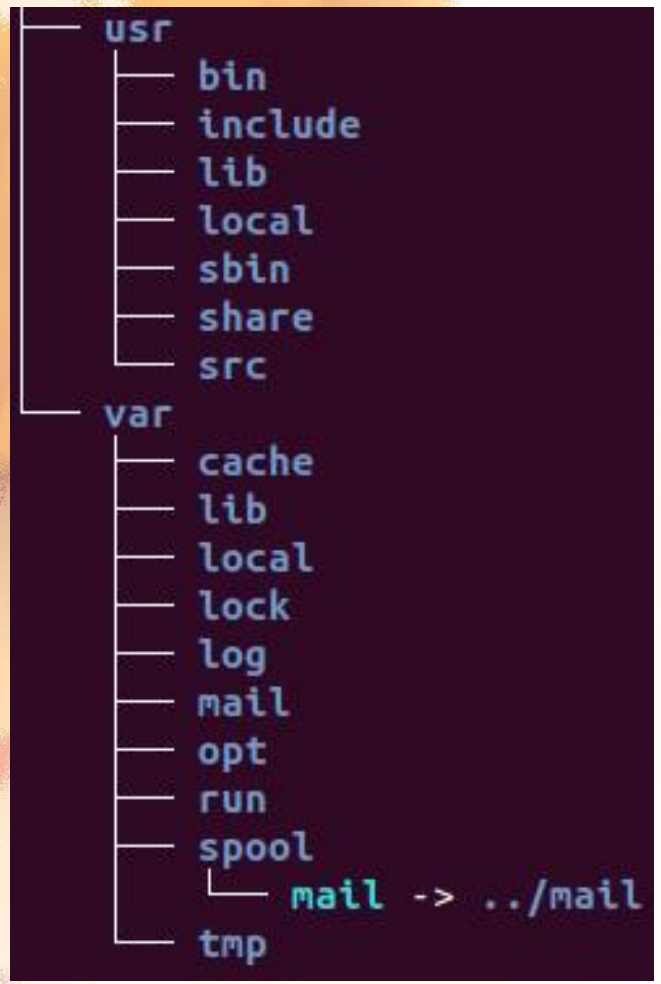
- FHS 标准
- https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html

The Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Linux distributions.

好像知道又似是而非的

FHS 标准





好像知道又似是而非的 Filesystem Hierarchy Standard 标准

- /bin vs. /usr/bin
- /usr vs. /usr/local

- ~/.* vs. ~/.config
- /usr/local vs. ~/.local

好像知道又似是而非的 Filesystem Hierarchy Standard 标准

- /bin vs. /usr/bin
- /usr vs. /usr/local

- ~/.* vs. ~/.config
- /usr/local vs. ~/.local + PATH?

好像知道又似是而非的 Filesystem Hierarchy Standard 标准

- /bin vs. /usr/bin
- /usr vs. /usr/local

- ~/.* vs. ~/.config
- /usr/local vs. ~/.local + PATH?

- “sudo pip install” ... or ... “pip install --user” ?

好像知道又似是而非的 Filesystem Hierarchy Standard 标准

- /bin vs. /usr/bin
- /usr vs. /usr/local

- ~/.* vs. ~/.config
- /usr/local vs. ~/.local + PATH?

- “sudo pip install” ... or ... “pip install --user” ?
  - Oh, yeah ... I think we should ....

好像知道又似是而非的 Filesystem Hierarchy Standard 标准

- /bin vs. /usr/bin
- /usr vs. /usr/local

- ~/.* vs. ~/.config
- /usr/local vs. ~/.local + PATH?

- “sudo pip install” ... or ... “pip install --user” ?
  - Oh, yeah ... I think we should .... But, how many?

从零构建 Linux 发行版

- 枯燥、复杂的完整手册
- http://www.linuxfromscratch.org/

从零构建 Linux 发行版 —— 核心步骤

1. 编译器么……	交叉编译 Cross-compile 创建目标 Toolchain 子集
2. 当然是要用自己编译自己	使用 Cross-toolchain 自举干净、无 (直接) 依赖的 Toolchain
3. 依赖一家人要整整齐齐	创建目标子系统，纳入所有编译依赖
4. 遵守 FHS 标准	创建目标系统目录树、绑定 Host 的设备树、chroot
5. 装你爱装的	编译定制化目标环境

为什么要自己对自己交叉编译？

- 依赖问题
  - as, ld, ...
  - ar, ranlib, ...
  - cpp, gcc, g++, ...
  - objcopy, readelf ...
- 自举概念

从零构建 Linux 发行版 —— 核心步骤

1. 编译器么……	交叉编译 Cross-compile 创建目标 Toolchain 子集
2. 当然是要用自己编译自己	使用 Cross-toolchain 自举干净、无 (直接) 依赖的 Toolchain
3. 依赖一家人要整整齐齐	创建目标子系统，纳入所有编译依赖
4. 遵守 FHS 标准	创建目标系统目录树、绑定 Host 的设备树、chroot
5. 装你爱装的	编译定制化目标环境

依赖一家人要整整齐齐

binutils	M4	gawk	sed
gcc	ncurses	gettext	tar
linux-headers	bash	grep	texinfo
libstdc++	bison	gzip	util-linux
glibc	bzip2	make	xz
tcl	coreutils	patch	
expect	diffutils	perl	
DejaGNU	findutils	python	

超管把你关进门，却给其他人开了窗？

- Chroot

```
chroot "${ROOTFS_DIR}" "/_toolchain/bin/env" -i                \
    HOME=/root                                                 \
    PATH="/bin:/usr/bin:/sbin:/usr/sbin:/_toolchain/bin"       \
    "/_toolchain/bin/bash"
```

超管把你关进门，却给其他人开了窗？

Chroot

(chroot) $ ls /bin

bash bzdiff bzgrep bzmore chown dd dnsdomainname false gunzip
kill ... ...

// Ah... Good ...

超管把你关进门，却给其他人开了窗？

Chroot

(chroot) $ ls /bin

bash bzdiff bzgrep bzmore chown dd dnsdomainname false gunzip
kill ... ...

// Ah... Good ...

(chroot) $ ls /dev

(chroot) $ ls: cannot access '/dev': No such file or directory

// Oh... No ...

超管把你关进门，却给其他人开了窗？

Chroot

(chroot) $ ls /bin

bash bzdiff bzgrep bzmore chown dd dnsdomainname false gunzip kill ... ...

// Ah... Good ...

(chroot) $ ls /dev

(chroot) $ ls: cannot access '/dev': No such file or directory

// Oh... No ...

Docker

```
/dev
|-- console
|-- core -> /proc/kcore
|-- fd -> /proc/self/fd
|-- full
|-- mqueue
|-- null
|-- ptmx -> pts/ptmx
|-- pts
|   |-- 0
|   `-- ptmx
|-- random
|-- shm
|-- stderr -> /proc/self/fd/2
|-- stdin -> /proc/self/fd/0
|-- stdout -> /proc/self/fd/1
|-- tty
|-- urandom
`-- zero
```

超管把你关进门，却给其他人开了窗？

```
mkdir -p ${ROOTFS_DIR}/{dev,proc,sys,run}

# Device ID references
# https://www.kernel.org/doc/html/latest/admin-guide/devices.html
mknod -m 600 "${ROOTFS_DIR}/dev/console" c 5 1
mknod -m 666 "${ROOTFS_DIR}/dev/null" c 1 3

# mount virtual kernel filesystem
mount --bind /dev "${ROOTFS_DIR}/dev"
mount -t devpts devpts ${ROOTFS_DIR}/dev/pts
mount -t proc proc ${ROOTFS_DIR}/proc
mount -t sysfs sysfs ${ROOTFS_DIR}/sys
mount -t tmpfs tmpfs ${ROOTFS_DIR}/run
```

包管理器、依赖树与发行版

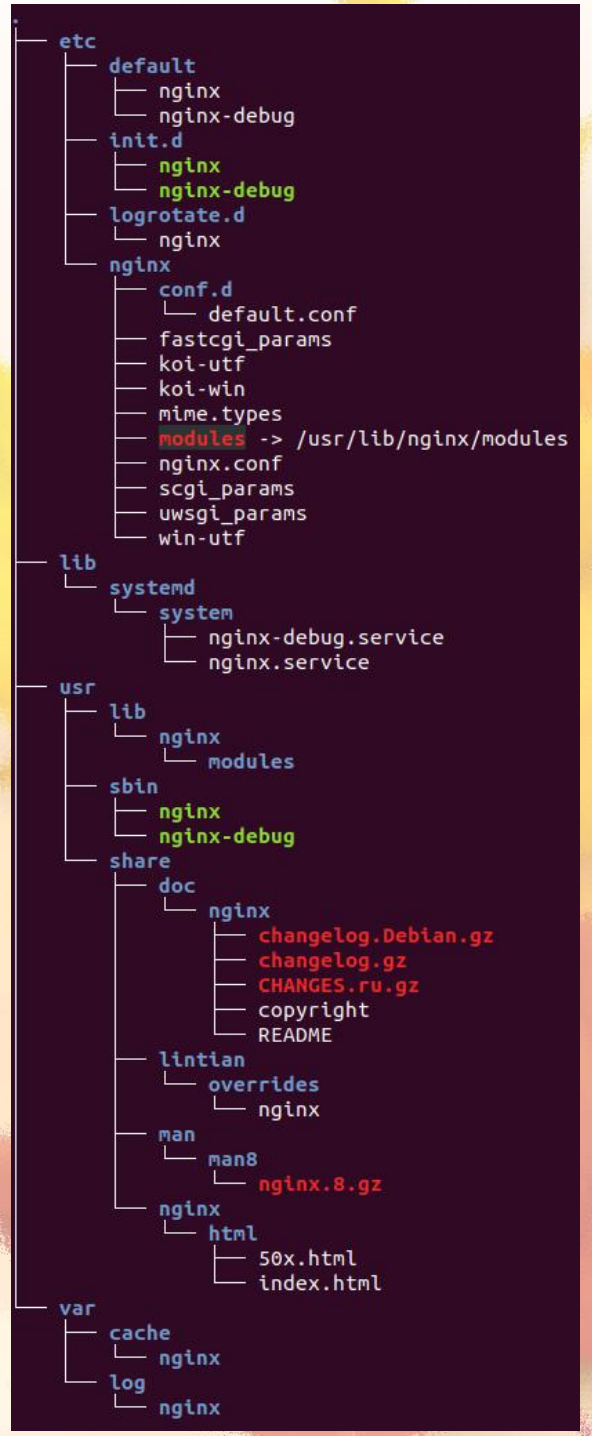
喂，老古董包管理器们！

- Debian: apt + deb
- Redhat: yum/dnf + rpm
- SUSE: zypper + rpm

喂，老古董包管理器们！

- Brain: tar.gz
- Debian: apt + deb
- Redhat: yum/dnf + rpm
- SUSE: zypper + rpm

Debian DEB



Redhat RPM

喂，老古董包管理器们！

包管理器管理了什么？

- 依赖树
  - 运行依赖
  - 构建依赖
  - 测试依赖
- 如何构建
- 如何运行
- 相关配置文件 (systemd, Sys V, etc.)
- 默认用户数据初始化等

理清细节却复杂到蛋疼的包构建

- 什么是依赖？
  - 硬件算不算依赖？
  - (狭义的) 运行时软件依赖
  - (广义的) 运行时配置依赖
- 如何定义依赖树
  - 依赖获取
  - 预检 host 的依赖

古董包管没解决的问题——新人包管解决好了吗？

- Suckless, Sta.li
- Snap, Flatpak, AppImage
- Nix
- Docker

发行版的开发者们都在做什么？

- 如何支持更新的软件——冻结与版本管理
- 如何支持更新的硬件
  - Linus Torvalds
  - Greg Kroah-Hartman
  - 发行版内核开发者们
- Security Patches
- 新的软件包
- 下一个发行版

你的错误设计为什么要我买单？

- 重构
- 向前兼容
- 跨依赖兼容

- Glibc-2.27 deprecated libio.h, and further 2.28 removed it.
  https://sourceware.org/ml/libc-announce/2018/msg00000.html

- sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' lib/*.c

尬聊时光 Q&A

- 常收好人卡的老实人包管理器
  - 找到了包，但包里有虫 (让你黑屏之 GPU 驱动包)
  - 没找到包，也没找到依赖包 (这个软件曾在20年前的 IBM OS/2 系统上工作过)
  - 找到了依赖包，但版本不对 (哦豁，这原来是用 Python 2 写的)
- 梦中情人包管理器该长啥样子？
  - Bottom-Up: 硬件控制与支持的颗粒度
  - Top-Down: 软件抽象层次颗粒度

尬聊时光 Q&A

- Perfect Software Solo:
  - 依赖想有就有、通力合作
  - 硬件完美工作、性能榨干
- 二进制独裁
  - You role it all
  - You provide all the features your customers (feel) in demand
  - You write all the libraries (you think) they need
  - You control all the hardware devices yourself
  - and ... you pray that your "software" works and upgradeable

尬聊时光 Q&A

谢谢！