

构建一致性指导书



reproducible-builds Sig

目录 Table of Contents

1 源代码.....	3
时间宏__DATE__和__TIME__	3
编译路径.....	4
链接顺序.....	5
引入不同的头文件	6
2 gcc编译器.....	7
GCC编译器configure配置信息归档.....	7
代码覆盖率测试选项.....	8
GCC静态库链接顺序	8
3 python.....	9
pyc路径差异消除指导.....	9
pyc时间戳差异消除指导	10
python 2to3差异消除指导.....	11
pyc集合(set)差异消除指导	12
同一py文件编译pyc的大量差异消除指导	12
Python字符串引用导致差异消除指导.....	13
Python字符串驻留导致差异消除指导	16
环境变量不统一差异.....	19
4 源代码版本控制工具.....	19
SVN	19
Git.....	21
5 ar生成的.a静态链接库文件.....	21
文件修改时间属性差异消除指导	21
uid和gid差异消除指导	22
文件权限属性差异消除指导	23
6 压缩、打包工具	23
tar	23
cpio	24
gzip	24

编译路径

差异现象：

当两次代码构建路径有差异时，而且会导致最终生成的二进制文件存在路径差异，以及由路径差异导致的build-id差异和地址偏移等。

00000000 24 07 00 00 00 00 00 00 24 07 00 00 00 00 00 00 \$.....\$.....	00000000 44 07 00 00 00 00 00 00 44 07 00 00 00 00 00 00 D.....D.....
00000190 50 E5 74 64 04 00 00 00 FC 05 00 00 00 00 00 00 P?td....?.....	00000190 50 E5 74 64 04 00 00 00 18 06 00 00 00 00 00 00 P?td....?.....
000001A0 FC 05 40 00 00 00 00 00 FC 05 40 00 00 00 00 00 00 ?.@.....?.....	000001A0 18 06 40 00 00 00 00 00 18 06 40 00 00 00 00 00 ?.@.....?.....
00000280 47 4E 55 00 A5 C0 8C 77 0D AC 66 5E 8D 0E A7 7C GNU.???w.??^?..?	00000280 47 4E 55 00 86 8F 73 99 6F 93 7A 24 0E 7A 31 GNU.???o?z\$.z1
00000290 04 0A E6 9F 42 E7 DA 2A 01 00 00 00 01 00 00 00 ????B??^.....	0000028F E5 21 B8 54 61 17 AA EB B0 01 00 00 00 01 00 00 ????Ta.???.....
0000044F 49 C7 C0 00 05 40 00 48 C7 C1 50 05 40 00 48 C7 I???@.H??P.@.H?	0000044F 49 C7 C0 00 05 40 00 48 C7 C1 60 05 40 00 48 C7 I???@.H??P.@.H?
0000052F 89 E5 BE 04 05 40 00 BF E9 05 40 ?? ??.@.??..@	0000052F 89 E5 BA 05 00 00 00 BE E4 05 40 00 BF 01 06 40 ???....??.@.??..@
0000053A 00 B8 00 00 00 00 E8 C8 FE FF FF B8 00 00 00 00 00 .?....?? ?..	0000053F 00 B8 00 00 00 00 E8 C6 FE FF FF B8 00 00 00 00 .?....?? ?..
0000054A 5D C3 0F 1F 40 j? ..@	0000054F 5D C3 66 2E 0F 1F 84 00 00 00 00 00 0F 1F 44 00 j?f....?.....D.
0000055F 41 54 4C 8D 25 A8 08 20 00 55 48 8D 2D A8 08 20 ATL?%?. .UH?-?..	0000056F 41 54 4C 8D 25 98 08 20 00 55 48 8D 2D 98 08 20 ATL?%?. .UH?-?..
0000057F 5D FE FF FF 48 85 ED 74 1E 0F 1F 84 00 00 00 00 00 1? H??t....?..	0000058F 4D FE FF FF 48 85 ED 74 1E 0F 1F 84 00 00 00 00 00 M? H??t....?..
000005CF 00 01 00 02 00 2F 6F 70 74 2F 6D 61 79 70 2F/opt/mayp/	000005DF 00 01 00 02 00 2F 6F 70 74 2F 6D 61 79 70 2F 62/opt/mayp/b
000005DE 66 69 6C 65 70 61 74 68 2E filepath.	000005EF 65 70 74 65 73 74 2F 66 69 6C 65 70 61 74 68 2E eptest/filepath.
000005F7 2E 0A 00 00 00 01 18 03 38 30;0.	0000060F 2C 29 25 64 20 2E 0A 00 00 01 18 03 38 34 00 ,%d....;4.
00000602 00 05 00 00 00 04 FE FF FF 7C 00 00 00 44 FE? ...D?	0000061E 00 05 00 00 00 E8 FD FF FF 80 00 00 00 28 FE? ...D?
00000612 FF FF 4C 00 00 00 31 FF FF FF A4 00 00 00 54 FF L..1 ?..T	0000062E FF FF 50 00 00 00 15 FF FF FF A8 00 00 00 48 FF P....? ?...H
00000622 FF FF C4 00 00 00 C4 FF FF FF 0C 01 00 00 ?..? ?..	0000063E FF FF C8 00 00 00 B8 FF FF FF 10 01 00 00 00 00 ?..? ?..
0000062F 00 14 00 00 00 00 00 00 01 7A 52 00 01 78zR..x	0000064E 00 00 14 00 00 00 00 00 00 01 7A 52 00 01 78zR..x

常见的由编译路径的不同导致的差异有两种场景，一种是源码中引用__FILE__宏。并且在编译时引用了该源代码文件的全路径，如：

```
gcc -c ${buildpath}/helloworld.c
```

如果两次构建的\${buildpath}不同，则该路径会被编译器解释为全路径，从而引入路径差异。

另一种是调试信息中会记录源代码文件编译时的路径，会导致二进制程序中调试信息段会存在大量的偏移量差异。

消除方案1：

消除的差异最彻底的方法是在源代码中不使用__FILE__宏。也可以修改构建脚本，不再使用全路径的方式编译，修改为相对路径，如：

```
cd ${buildpath}
gcc -c ./helloworld.c
```

注：如使用cmake自动生成Makefile，cmake会自动扩展为全路径。该方案无法消除。

消除方案2：

当__FILE__用于调测和定位问题，则可以使用__FUNCTION__替换。好处一方面是函数名通常更能体现处理动作，会更易于定位，另一方面是通用性强，整改更简单；缺点是和以前的维护习惯不同，另外C++还可能出现函数重载，会出现多处函数名一样，这种情况仅在C语言下可以用。

消除方案3：

在cmake或makefile中生成一个文件名的宏，如__FILENAME__，文件绝对路径能够获取到，此时把去掉路径的文件名定义成这个宏，就能在c/cpp中使用。该方式优点是日志记录和之前保持一致，维护习惯不变；坏处是不支持头文件里面直接引用__FILENAME__，因为gcc的编译单元是c/cpp，头文件是在编译的时候加工到c/cpp中的，没有单独的实体。

```
set(FILENAME_MACRO "-D__FILENAME__=\"$(lastword $(subst /, $(abspath $<)))\"")
string(APPEND CMAKE_C_FLAGS " ${FILENAME_MACRO}")
string(APPEND CMAKE_CXX_FLAGS " ${FILENAME_MACRO}")
```

消除方案4：

如果是C++，利用模板元编程的方式，在公共头文件中定义统一的__FILENAME__（或其他名字）宏，通过编译器推导，把目录去掉，在编译期获得纯粹的文件名，这对头文件也适用。

```
template<std::ptrdiff_t index>
inline constexpr const char* basename(const char* path)
{
    return ((path[index] == '/') || (path[index] == '\\'))
        ? (path + index + 1) : basename<index - 1>(path);
}

template<>
inline constexpr const char* basename(const char* path)
{
    return ((path[index] == '/') || (path[index] == '\\'))
        ? (path + index + 1) : basename<index - 1>(path);
}

template<>
inline constexpr const char* basename<0>(const char* path)
{
    return path;
}

#ifndef __FILENAME__
#define __FILENAME__ basename<sizeof(__FILE__) - 1>(__FILE__)
#endif
```

消除方案5:

如果代码中没有使用__FILE__宏，而路径差异仅出现在调试信息中，可以删除调试信息，例如，在编译代码时使用gcc的-s参数，或者在编译脚本中调用strip工具删除调试信息。

消除方案6:

高版本的编译器也开始增加构建一致性的能力，如：

-fdebug-prefix-map=OLD=NEW 可以从调试信息中删除目录前缀。（适用于所有GCC版本，Clang 3.8）

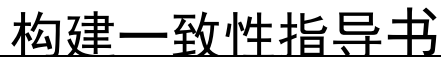
-fmacro-prefix-map=OLD=NEW 与相似-fdebug-prefix-map，但是解决了由于例如__FILE__在assert调用中使用宏而导致的一致性。（自GCC 8和Clang 10起可用）

例如：在顶级目录规则中，将-fdebug-prefix-map=BUILDPATH=.和-fmacro-prefix-map=BUILDPATH=.添加到 CFLAGS, CXXFLAGS, OBJCFLAGS, OBJCXXFLAGS, GCJFLAGS, FFLAGS and FCFLAGS这几个变量中。

链接顺序

差异现象:

直接比较链接顺序有差异的二进制文件，差异现象非常大。



当多个中间文件链接成可执行程序或动态库时，通常是用通配符来适配所有中间文件，如：`gcc *.o -o output.so`

通配符会被构建环境上的环境变量影响，从而在最终链接生成的二进制文件链接顺序不同，导致最终生成的二进制有差异。

1. 使用固定的链接顺序，在编写Makefile时，将中间文件的链接顺序写成固定的顺序，如：`gcc a.o b.o c.o -o output.so`
2. 使用固定的链接顺序，在编写Makefile时，对输入文件进行排序，例如：

修改为：

差异现象:

[illegible]

gcc在编译时按照如下顺序寻找所需要的头文件:

- 第6页,共24页Page 6.

3)然后找gcc的环境变量 C_INCLUDE_PATH, CPLUS_INCLUDE_PATH, OBJC_INCLUDE_PATH

4)再找内定目录: /usr/include, /usr/local/include

5)最后找gcc的一系列自带目录, 如:

CPLUS_INCLUDE_PATH=/usr/lib/gcc/x86_64-redhat-linux/4.8.5/include

消除方案:

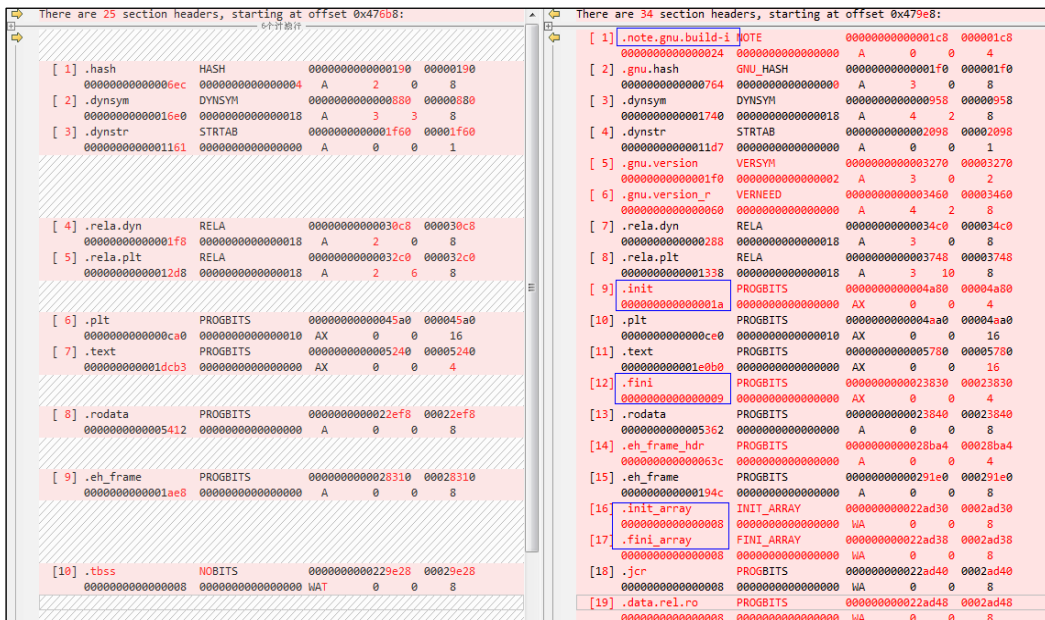
屏蔽/usr/local/include/该路径下的zlib.h, 编译.o一致。

2 gcc编译器

GCC 编译器 configure 配置信息归档

差异现象:

在构建工具管理规范中, 优选的GCC版本要高于linux系统自带的GCC版本, 所以产品需要优先编译GCC, 而在编译GCC时如果configure配置信息不一致, 会导致同一个GCC版本编译生成的二进制文件差异非常巨大。在如下差异截图中, 右侧的二进制文件比左侧的文件多了多个section, 如.note.gnu.build-id段, fini段, init段等。



Index	Section Name	Type	Address	Size	Flags
[1]	.hash	HASH	000000000000190	00000190	A 2 0 8
[2]	.dynsym	DYNSYM	00000000000016e0	00000880	A 3 3 8
[3]	.dynstr	STRTAB	0000000000001f60	00001f60	A 0 0 1
[4]	.rela.dyn	RELA	0000000000001f8	000030c8	A 2 0 8
[5]	.rela.plt	RELA	00000000000012d8	000032c0	A 2 6 8
[6]	.plt	PROGBITS	000000000000c0	000045a0	AX 0 0 16
[7]	.text	PROGBITS	0000000000001dc3	00005240	AX 0 0 4
[8]	.rodata	PROGBITS	0000000000005412	00022ef8	A 0 0 8
[9]	.eh_frame	PROGBITS	0000000000001ae8	00028310	A 0 0 8
[10]	.tbss	NOBITS	0000000000000008	00029e28	WAT 0 0 8
[11]	.note.gnu.build-id	NOTE	000000000000024	000001c8	A 0 0 4
[12]	.gnu.hash	GNU_HASH	000000000000764	000001f0	A 3 0 8
[13]	.dynsym	DYNSYM	0000000000001740	00000958	A 4 2 8
[14]	.dynstr	STRTAB	00000000000011d7	00000298	A 0 0 1
[15]	.gnu.version	VERSYM	00000000000001f0	00003270	A 3 0 2
[16]	.gnu.version_r	VERNEED	000000000000060	00003460	A 4 2 8
[17]	.rela.dyn	RELA	0000000000000288	000034c0	A 3 0 8
[18]	.rela.plt	RELA	0000000000001338	00003748	A 3 10 8
[19]	.init	PROGBITS	000000000000001a	00004a80	AX 0 0 4
[20]	.plt	PROGBITS	0000000000000ce0	00004aa0	AX 0 0 16
[21]	.text	PROGBITS	00000000000001e0b0	00005780	AX 0 0 16
[22]	.fini	PROGBITS	0000000000000009	00023830	AX 0 0 4
[23]	.rodata	PROGBITS	00000000000005362	00023840	A 0 0 8
[24]	.eh_frame_hdr	PROGBITS	000000000000063c	00028ba4	A 0 0 4
[25]	.eh_frame	PROGBITS	0000000000000194c	000291e0	A 0 0 8
[26]	.init_array	INIT_ARRAY	000000000000022ad30	0002ad30	WA 0 0 8
[27]	.fini_array	FINI_ARRAY	000000000000022ad38	0002ad38	WA 0 0 8
[28]	.jcr	PROGBITS	000000000000022ad40	0002ad40	WA 0 0 8
[29]	.data.rel.ro	PROGBITS	000000000000022ad48	0002ad48	WA 0 0 8

GCC的configure配置会影响编译生成的二进制文件, 例如GCC的configure中有--enable-linker-build-id和--enable-finitfini-array等配置, 会导致生成的ELF格式文件中会比没有这些配置的ELF多几个section。

通过GCC的-v参数, 可以查看已安装GCC的configure信息。

Total24

消除方案:

3 python

pyc 路径差异消除指导

差异现象:

python的源代码在编译时, 会将.py文件的路径信息保存在生成的pyc中, 导致产生差异。

00000180 00 00 28 00 00 00 00 73 16 00 00 00 2F 65 .(...s... / e	00000180 00 00 28 00 00 00 00 73 10 00 00 00 2E 2F 74 65 .(...s.../te
0000018E 78 70 6F 72 74 2F 68 6F 6D 65 2F 74 6F 6D 63 61 xport/home/tomca	00000190 73 74 2F 74 6F 6D 63 61 s t /tomca
0000047E 73 16 00 00 00 2F 65 78 70 6F 72 74 2F 68 s... / export/h	00000478 73 10 00 00 00 2E 2F 74 65 73 s.../te s t
0000048C 6F 6D 65 2F 74 6F 6D 63 61 74 2E 70 79 74 0C 00 ome/tomcat.pyt..	00000483 2F 74 6F 6D 63 61 74 2E 70 79 74 0C 00 /tomcat.pyt..
000005CC 00 73 16 00 00 00 2F 65 78 70 6F 72 74 2F .s... / export/	000005CC 00 73 10 00 00 00 2E 2F 74 65 73 .s.../te s t
000005DA 68 6F 6D 65 2F 74 6F 6D 63 61 74 2E 70 79 74 04 home/tomcat.pyt.	000005CC 2F 74 6F 6D 63 61 74 2E 70 79 74 04 /tomcat.pyt.
0000064A 00 00 00 28 00 00 00 00 73 16 00 00 00 2F ...(...s... /	00000638 00 00 00 28 00 00 00 00 73 10 00 00 00 2E 2F 74 ...(...s.../t
00000658 65 78 70 6F 72 74 2F 68 6F 6D 65 2F 74 6F 6D 63 export/home/tomc	00000648 65 73 74 2F 74 6F 6D 63 es t /tomc

pip安装的库pyc中出现的路径差异, 该差异当前无法消除, 建议不要打包pyc文件或者通过

`python -m compileall`重新编译pyc文件

150 07 63 65 72 74 69 66 69 7A 0A 63 61 63 65 72 74 2E 70 65 6D 29 05 DA 0C .certifz.cacert.pem).0.	00000150 07 63 65 72 74 69 66 69 7A 0A 63 61 63 65 72 74 2E 70 65 6D 29 05 DA 0C .certifz.cacert.pem).0.
160 5F 43 41 43 45 52 54 5F 50 41 54 48 DA 08 67 65 74 5F 70 61 74 68 DA 08 _CACERT_PATH0.get_path0.	00000160 5F 43 41 43 45 52 54 5F 50 41 54 48 DA 08 67 65 74 5F 70 61 74 68 DA 08 _CACERT_PATH0.get_path0.
170 5F 43 41 43 45 52 54 5F 43 54 58 DA 03 73 74 72 DA 08 5F 5F 65 6E 74 65 _CACERT_CTX0.str0__ente	00000160 5F 43 41 43 45 52 54 5F 43 54 58 DA 03 73 74 72 DA 08 5F 5F 65 6E 74 65 _CACERT_CTX0.str0__ente
180 72 5F 5F A9 00 72 0A 00 00 00 00 00 00 33 2F 74 6D 70 2F 70 69 _0.r.....03/tmp/pi	00000180 72 5F 5F A9 00 72 0A 00 00 00 00 00 00 33 2F 74 6D 70 2F 70 69 _0.r.....03/tmp/pi
190 70 20 74 61 72 67 65 74 20 77 30 65 78 78 35 6C 35 2F 6C 69 62 2F 70 p-target-w @pex18/lib/p	00000180 70 20 74 61 72 67 65 74 20 77 30 65 78 78 35 6C 35 2F 6C 69 62 2F 70 p-target-w @pex18/lib/p
1C7 79 74 68 6F 6E 2F 63 65 72 74 69 66 69 2F 63 6F 72 65 2E 70 79 74 05 77 ython/certifz/core.py0.w	000001C7 79 74 68 6F 6E 2F 63 65 72 74 69 66 69 2F 63 6F 72 65 2E 70 79 74 05 77 ython/certifz/core.py0.w
1F7 68 65 72 65 11 00 00 73 68 00 00 00 00 00 00 0A 01 8C 02 72 8C 00 here.....f..	000001F7 68 65 72 65 11 00 00 73 68 00 00 00 00 00 00 0A 01 8C 02 72 8C 00 here.....f..
10F 00 00 DA 05 61 73 63 69 69 63 83 00 00 00 00 00 04 00 00 00 00 .0.ascii.....	000001F7 00 00 DA 05 61 73 63 69 69 63 83 00 00 00 00 00 04 00 00 00 00 .0.ascii.....
10F 00 00 43 00 00 73 24 00 00 00 74 04 81 83 00 64 01 7C 02 64 02 8D C...\$.t.t.f.d;.d..	0000020F 00 00 43 00 00 73 24 00 00 00 74 04 81 83 00 64 01 7C 02 64 02 8D C...\$.t.t.f.d;.d..

消除方案1:

在python编译时使用相对路径, 构建命令可以参考如下形式:

```
cd ${py_source_path}
python -m compileall ./
```

消除方案2:

CI构建python时编译脚本和源码构建是独立目录。修改compilepy.py编译脚本, 引入参数ddir解决路径的一致性问题的。

```
import sys
import compileall
#ddir 替换路径统一替换为". "
result = compileall.compile_dir(dir=sys.argv[1],ddir=".")
#python 源码中返回 1 表示成功 返回 0 表示失败
if result == 0:
    print "[ERROR]Failed to compile python scripts.For more details, please
    seach SyntaxError"
    sys.exit(1)
    sys.exit(0)
```

修改前后的对比如下:

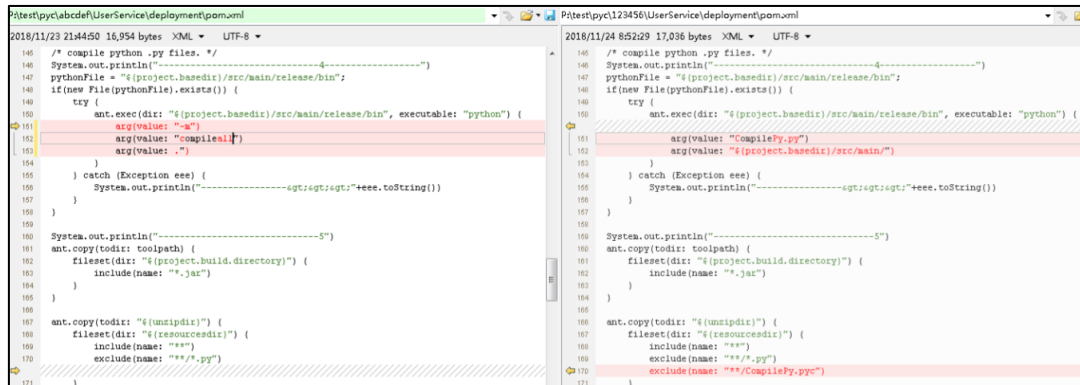
2017/3/2 16:58:14 99 字节 Python 脚本 ANSI UNIX	2017/3/2 16:58:14 99 字节 Python 脚本 ANSI UNIX
1 import compileall	1 import compileall
2 import sys	2 import sys
3 ret = compileall.compile_dir(sys.argv[1])	3 ret = compileall.compile_dir(dir=sys.argv[1],ddir=sys.argv[2])
4 if ret == 0:	4 if ret == 0:
5 sys.exit(1)	5 sys.exit(1)

消除方案3:

Pom直接构建python源码，将脚本CompilePy.py作为工具使用，归档到源码包中。

- 1) 将CompilePy.py归档到src/main/release/bin目录，输入一个参数用来指定py文件位置；
- 2) 打包时删除由compilepy生成的pyc文件；

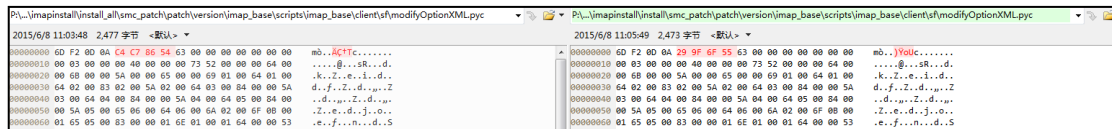
pom文件修改示例如下：



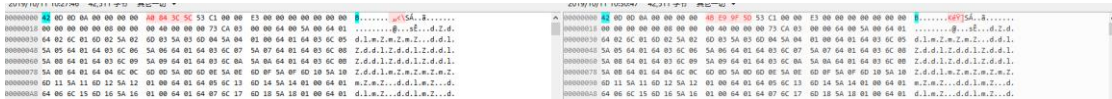
pyc 时间戳差异消除指导

差异现象：

Python<3.7



Python>=3.7



- 1、pyc文件是为了加快py文件加载的产物。在第一次运行python代码时，代码中import的库会自动生成pyc文件，py文件更新后会生成新的pyc文件；
- 2、为了判断pyc文件是否需要重新生成所以在pyc文件中加入了py文件的时间戳
- 3、Python3.7之前版本时间戳在pyc文件的4到7字节，python3.7版本开始做了可重复构建的优化，在编译过程中加指定参数就不会写入时间戳了，pyc格式也做了修改，默认编译的pyc文件是包含时间戳的在文件的8到11字节。
- 4、在代码同步或者移动过程中py文件的时间戳发生了变化，导致生成的pyc文件也发生了变化。

消除方案：

升级到Python3.7及以后的版本。

Python3.7及以后版本可利用新版引入的参数进行消除配置环境变量
SOURCE_DATE_EPOCH=1

python 2to3 差异消除指导

差异现象：

在编译python解释器时，执行make install生成的Grammarx.y.z.final.0.pickle文件存在差异



在编译python解释器执行make install时会把lib2to3/Grammar.txt文件转换成pickle文件，生成Grammarx.y.z.final.0.pickle文件。在转换过程中，make_dfa函数中将NFASState对象作为键名放入字典中，python中字典的顺序是按照键名的hash进行排序的，对象计算hash时和对象的内存地址有关，由于内存地址的随机性导致字典顺序随机。后面的一些转换会在对象中记录该字典的顺序，最终导致每次生成的pickle文件不同。

消除方案1：

利用python的预加载机制和模块运行时替换功能，动态地为NFASState添加__hash__方法。创建~/local/lib/pythonX.Y/site-packages/ usercustomize.py文件(X.Y为python版本号2.6或2.7)，写入如下代码，若文件已经存在也可以直接将下面代码追加进文件中。

```
import sys
import lib2to3.pgen2.pgen
def hash_patch(self):
    return 1
sys.modules['lib2to3.pgen2.pgen'].NFASState.__hash__ = hash_patch
```

消除方案2：

先通过find / -name sitecustomize.py 查找环境中是否存在sitecustomize.py文件，并且sitecustomize.py 文件在python的sys.path路径中

例如：

```
/usr/lib/python2.x
/usr/lib/python2.x/site-packages/
/usr/lib64/python2.x
/usr/lib64/python2.x/site-packages
/usr/local/lib/python2.x/
/usr/local/lib/python2.x/site-packages
/usr/local/lib64/python2.x/
/usr/local/lib64/python2.x/site-packages
```

如果存在则在文件后面添加下面代码

```
import sys
import lib2to3.pgen2.pgen
def hash_patch(self):
    return 1
```

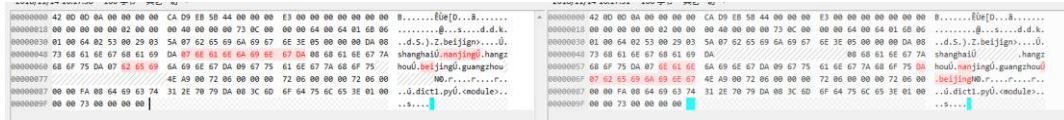
```
sys.modules['lib2to3.pgen2.pgen'].NFAStruct.__hash__ = hash_patch
```

如果不存在则创建/usr/lib64/python2.X/site-packages/sitecustomize.py文件，写入上面的代码。

pyc 集合(set)差异消除指导

差异现象：

Pyc文件中出现顺序差异



Python3代码中如下使用set时，编译出的pyc文件会出现差异，例如：

```
'beijing' in {'beijing', 'hangzhou', 'nanjing', 'shanghai', 'guangzhou'}
```

消除方案1：

使用frozenset固定集合顺序

```
'beijing' in frozenset({'beijing', 'hangzhou', 'nanjing', 'shanghai', 'guangzhou'})
```

消除方案2：

将集合赋值给变量再，使用in判断元素时不会出现差异

```
city = {'beijing', 'hangzhou', 'nanjing', 'shanghai', 'guangzhou'}
'beijing' in city
```

消除方案3：

在编译环境中设置环境变量：PYTHONHASHSEED=0

同一 py 文件编译 pyc 的大量差异消除指导

差异现象：

一般编译py文件使用compileall这个接口，不会出现这种差异，但是有些自定义的编译脚本使用的是compiler这个接口编译时就会产生这样的差异

Total|24


```

1  #define TYPE_NULL.....'0'
2  #define TYPE_NONE.....'N'
3  #define TYPE_FALSE.....'F'
4  #define TYPE_TRUE.....'T'
5  #define TYPE_STOPITER.....'S'
6  #define TYPE_ELLIPSIS.....'.'
7  #define TYPE_INT.....'i'
8  /* TYPE_INT64 is not generated anymore.
9  [...] Supported for backward compatibility only. */
10 #define TYPE_INT64.....'I'
11 #define TYPE_FLOAT.....'f'
12 #define TYPE_BINARY_FLOAT.....'g'
13 #define TYPE_COMPLEX.....'x'
14 #define TYPE_BINARY_COMPLEX.....'y'
15 #define TYPE_LONG.....'l'
16 #define TYPE_STRING.....'s'
17 #define TYPE_INTERNEDED.....'t'
18 #define TYPE_REF.....'r'
19 #define TYPE_TUPLE.....'('
20 #define TYPE_LIST.....'['
21 #define TYPE_DICT.....'{'
22 #define TYPE_CODE.....'c'
23 #define TYPE_UNICODE.....'u'
24 #define TYPE_UNKNOWN.....'?'
25 #define TYPE_SET.....'<'
26 #define TYPE_FROZENSET.....'>'
27 #define FLAG_REF.....'\x80' /* with a type, add obj to index */
28
29 #define TYPE_ASCII.....'a'
30 #define TYPE_ASCII_INTERNEDED.....'A'
31 #define TYPE_SMALL_TUPLE.....')'
32 #define TYPE_SHORT_ASCII.....'z'
33 #define TYPE_SHORT_ASCII_INTERNEDED.....'Z'

```

如果该字符串被引用过，该类型的编码会和0x80进行或运算

```

>>> hex(ord('Z')|0x80)
'0xda'

```



```
static int
w_ref(PyObject *v, char *flag, WFILE *p)
{
    ..._Py_hashtable_entry_t *entry;
    ...int w;

    ...if (p->version < 3 || p->hashtable == NULL)
    ...    return 0; /* not writing object references */

    .../* if it has only one reference, it definitely isn't shared */
    ...if (Py_REFCNT(v) == 1)
    ...    return 0;

    ...entry = _Py_HASHTABLE_GET_ENTRY(p->hashtable, v);
    ...if (entry != NULL) {
    ...    .../* write the reference index to the stream */
    ...    ..._Py_HASHTABLE_ENTRY_READ_DATA(p->hashtable, entry, w);
    ...    .../* we don't store "long" indices in the dict */
    ...    ...assert(0 <= w && w <= 0x7fffffff);
    ...    ...w_byte(TYPE_REF, p);
    ...    ...w_long(w, p);
    ...    ...return 1;
    ...} else {
    ...    ...size_t s = p->hashtable->entries;
    ...    .../* we don't support long indices */
    ...    ...if (s >= 0x7fffffff) {
    ...    ...    PyErr_SetString(PyExc_ValueError, "too many objects");
    ...    ...    goto err;
    ...    ...}
    ...    ...w = (int)s;
    ...    ...Py_INCREF(v);
    ...    ...if (_Py_HASHTABLE_SET(p->hashtable, v, w) < 0) {
    ...    ...    Py_DECREF(v);
    ...    ...    goto err;
    ...    ...}
    ...    ...*flag |= FLAG_REF;
    ...    ...return 0;
    ...}
err:
    ...p->error = WFERR_UNMARSHALLABLE;
    ...return 1;
}
```

所以为DA的文件在编译时该处的内容被引用过，如果为5A表示未被引用过。

消除方案：

1. 在编译环境中设置环境变量：PYTHONHASHSEED=0
2. 比较一下两个环境中python安装目录，保证两个环境中python安装的第三方库一致，python的标准库一致，pyc文件数量一致。
3. 两个环境中python安装目录的权限一致，不要出现一个环境上是root用户，另一个环境上是huawei用户。

Python 字符串驻留导致差异消除指导

差异现象：

		2020/6/22 23:44:06	2,503 bytes	Everything Else
DA	01 78 FA 01 7C 7A 02	.@=ú.[ú.]ú.^z.^=ú.{ú. z.		
00	00 00	=ú.}ú.^c.....		

环境信息：

Python3.7以上版本中随机出现

差异分析：

结合上一节中关于数据类型的介绍这里可以推断出存在差异的两个数据类型DA代表数据类型为TYPE_SHORT_ASCII_INTERNERD，FA代表数据类型为TYPE_SHORT_ASCII。

```
>>> hex(ord('Z')|0x80)
'0xda'
>>> hex(ord('z')|0x80)
'0xfa'
```

TYPE_SHORT_ASCII_INTERNERD就表示这个字符串被intern过了，

TYPE_SHORT_ASCII表示没有intern过。

从差异上来看出现差异的字段后面一般都是‘{或者}’，搜索python源码发现了下面这段代码

```
static int
maybe_init_static_strings(void)
{
    ....if (!_str_open_br &&
    ....!(_str_open_br = PyUnicode_InternFromString("{}"))){
    ....return -1;
    ....}
    ....if (!_str_dbl_open_br &&
    ....!(_str_dbl_open_br = PyUnicode_InternFromString("{{}"))){
    ....return -1;
    ....}
    ....if (!_str_close_br &&
    ....!(_str_close_br = PyUnicode_InternFromString("}"))){
    ....return -1;
    ....}
    ....if (!_str_dbl_close_br &&
    ....!(_str_dbl_close_br = PyUnicode_InternFromString("{}"))){
    ....return -1;
    ....}
    ....return 0;
}

static PyObject *
expr_as_unicode(expr_ty e, int level)
{
    ...._PyUnicodeWriter writer;
    ...._PyUnicodeWriter_Init(&writer);
    ....writer.min_length = 256;
    ....writer.overallocate = 1;
    ....if (-1 == maybe_init_static_strings() ||
    ....-1 == append_ast_expr(&writer, e, level))
    ....{
    ....    ...._PyUnicodeWriter_Dealloc(&writer);
    ....    ....return NULL;
    ....}
    ....return _PyUnicodeWriter_Finish(&writer);
}

PyObject *
_PyAST_ExprAsUnicode(expr_ty e)
{
    ....return expr_as_unicode(e, PR_TEST);
}
```

当使用python3.7新引入的<https://www.python.org/dev/peps/pep-0563/>语法时，例如下面的代码会调用该处代码进行解析。使用了类型注解并且from __future__ import annotations

```

from __future__ import annotations
USING_STRINGS = True

# dataclass_module_1.py and dataclass_module_1_str.py are identical
# except only the latter uses string annotations.

import dataclasses
import typing

T_CV2 = typing.ClassVar[int]
T_CV3 = typing.ClassVar

T_IV2 = dataclasses.InitVar[int]
T_IV3 = dataclasses.InitVar

@dataclasses.dataclass
class CV:
    ... T_CV4 = typing.ClassVar
    ... cv0: typing.ClassVar[int] = 20
    ... cv1: typing.ClassVar = 30
    ... cv2: T_CV2
    ... cv3: T_CV3
    ... not_cv4: T_CV4 # When using string annotations, this field is not recognized as a ClassVar.

@dataclasses.dataclass
class IV:
    ... T_IV4 = dataclasses.InitVar
    ... iv0: dataclasses.InitVar[int]
    ... iv1: dataclasses.InitVar
    ... iv2: T_IV2
    ... iv3: T_IV3
    ... not_iv4: T_IV4 # When using string annotations, this field is not recognized as an InitVar.

```

分析Python3.8.2的标准库，只有在编译dataclass_module_2_str.py dataclass_module_1_str.py dataclass_textanno.py这三个文件的时候会调用到maybe_init_static_strings。

由于python3.8的Makefile中调用compileall.py去编译标准库的时候相对python3.7的Makefile多了-j0参数，默认启用了多进程编译。编译过程中编译文件顺序就出现了随机性，如果先编译dataclass_module_2_str.py等会调用maybe_init_static_strings函数的文件再编译token.py、tokenize.py等运行python时会自动调用的pyc时，这些pyc文件中的‘{’、‘}’就是被驻留的，然后编译其他py文件时，如果代码中存在‘{’、‘}’就会被驻留pyc中显示DA。如果先编译token.py、tokenize.py后编译dataclass_module_2_str.py这些文件就不会驻留，pyc中显示FA。

消除方案1:

在两个构建环境上重新编译装python，在编译python时配置环境变量

PYTHONHASHSEED=0 、SOURCE_DATE_EPOCH=1，

并通过下面命令进行安装如下:

```
make install PYTHON_FOR_BUILD='./$(BUILDPYTHON)'
```

如果是python3.8安装前先执行sed -i "s/-j0 -d/-d/g" Makefile把多线程编译标准库关掉编译自研代码时使用单线程编译。

生成的pyc文件可能会随着python版本的不同，有的版本下是DA有的版本下是FA。

python3.8.2编译出的pyc为DA。

消除方案2:

在编译前删除掉python安装目录的lib/python3.8/__pycache__/token.cpython-38.*或

lib/python3.7/__pycache__/tokenize.cpython-37.*如果还不行可以将lib/python3.8/__pycache__

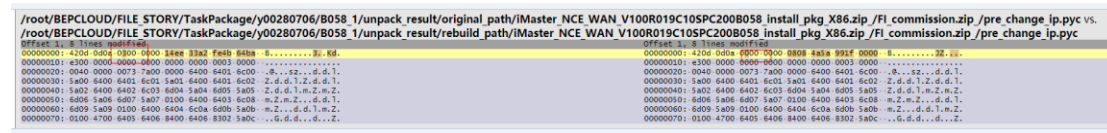
或lib/python3.7/__pycache__整个目录删除。

编译自研代码时使用单线程编译。

重新生成的pyc文件会统一变成FA。

环境变量不统一差异

差异现象：



差异分析：

个别产品构建环境不统一，由于SOURCE_DATE_EPOCH的设置不同，导致Python3.7/3.8的pyc文件出现差异

配置了SOURCE_DATE_EPOCH环境变量，编译出来的pyc显示03，没配置环境变量的情况下编译出来的就是00

消除方案3：

两个环境都配置SOURCE_DATE_EPOCH环境变量

4 源代码版本控制工具

SVN

差异现象：

使用SVN做为版本控制工具，从配置库下载源代码到本地后，源代码的修改时间和创建时间属性会被修改为本地实际，而不是配置库上的时间。SVN库上的文件时间如下图所示：

	Extension	Revision	Author	Size	Date	Lock	Lod
		82045			2018/11/19 9:09:58		
		80832			2018/10/27 17:16:32		
	.h	78820		788 字节	2018/9/11 20:38:45		
	.c	82543		35.0 KB	2018/11/24 14:08:57		
i.sh	.sh	82541		10.7 KB	2018/11/24 12:53:29		
icons.py	.py	82045		8.87 KB	2018/11/19 9:09:58		
	.c	78820		8.56 KB	2018/9/11 20:38:45		
y.h	.h	62125		26.0 KB	2017/12/7 15:15:55		
m.c	.c	82384		13.4 KB	2018/11/22 19:47:55		
.c	.c	79564		2.19 KB	2018/9/25 10:16:20		
re.conf	.conf	65971		89 字节	2018/3/16 15:51:38		
	.sh	82041		1.48 KB	2018/11/18 17:51:51		
		68063		2.52 KB	2018/4/11 17:22:52		
	.c	78820		1.51 KB	2018/9/11 20:38:45		
		81466		3.22 KB	2018/11/8 14:58:40		
:	.c	68063		101 字节	2018/4/11 17:22:52		
c	.c	65397		134 字节	2018/3/6 17:12:05		
r.c	.c	68063		218 字节	2018/4/11 17:22:52		
it	.txt	56947		717 字节	2017/8/31 15:46:31		

使用SVN工具将代码下载到本地后，所有文件和文件夹的修改日期属性会被修改成当前时间，如下图所示：

or time 02-Source code			搜索 02-Source code
助(H)			
新建文件夹			
	修改日期	类型	
.	2018/11/24 15:45	文件夹	
.	2018/11/24 15:45	文件夹	
.h	2018/11/24 15:45	H 文件	
r.c	2018/11/24 15:45	C 文件	
r_ci.sh	2018/11/24 15:45	Shell Script	
r_scons.py	2018/11/24 15:45	Python File	
:	2018/11/24 15:45	C 文件	
eue.h	2018/11/24 15:45	H 文件	
idom.c	2018/11/24 15:45	C 文件	
ce.c	2018/11/24 15:45	C 文件	
dtime.conf	2018/11/24 15:45	CONF 文件	
	2018/11/24 15:45	Shell Script	
Log	2018/11/24 15:45	文件	
.c	2018/11/24 15:45	C 文件	
e	2018/11/24 15:45	文件	
r.c	2018/11/24 15:45	C 文件	
54.c	2018/11/24 15:45	C 文件	
54_r.c	2018/11/24 15:45	C 文件	
s.txt	2018/11/24 15:45	Notepad++ Doc...	

环境信息：

使用SVN同步代码的构建场景。

差异分析:

每次代码编译都需要从配置库下载代码，则源代码的修改时间属性每次都会发生变化，部分编译器和构建工具会将文件的修改时间属性写入二进制文件中，从而导致每次构建生成的二进制文件都会有时间戳差异。

例如python编译器在将.py文件编译成.pyc或.pyo时，就会写.py的修改时间记录到.pyc或.pyo中。

消除方案:

使用SVN工具自带参数，使源码下载后本地时间戳与配置库时间戳一致。

传统命令:

```
svn checkout ${svn} ${local_path}
```

增加参数后命令:

```
svn checkout ${svn} --config-option config:miscellany:use-commit-times=yes ${local_path}
```

Git

差异现象:

使用Git从代码库下载代码，下载后的源代码修改时间戳会被重置为当前时间（与SVN类似）。

环境信息:

使用Git同步代码的构建场景。

消除方案:

使用Git工具自带参数，使源码下载后本地时间戳与配置库时间戳一致。

传统命令:

```
git clone ${git} -b branch
```

增加参数后命令:

```
git ls-files | while read file; do echo $file; touch -d $(git log --date=local -1 --format="%ct" "$file") "$file"; done
```

5 ar生成的.a静态链接库文件

文件修改时间属性差异消除指导

差异现象:

ar工具会将obj文件的修改时间写入到.a文件中导致差异。虽然这种时间戳不会被链接到产品的二进制文件中，但是，如果直接交付.a文件的场景下，第一次构建没有消除时间戳差异，重构建时消除差异的技术难度和工作量都很高。

消除方案1:

```

linux-v07s:/opt # ar --help
Usage: ar [emulation options] [-]{dmpqrstx}[abcDfilMNOpsStuvV] [--plugin <name>] [member-name] [count]
] archive-file file...
    ar -M [<mri-script>]

commands:
d      - delete file(s) from the archive
m[ab] - move file(s) in the archive
p      - print file(s) found in the archive
q[f]   - quick append file(s) to the archive
r[ab][f][u] - replace existing or insert new file(s) into the archive
s      - act as ranlib
t      - display contents of archive
x[o]   - extract file(s) from the archive

command specific modifiers:
[a]     - put file(s) after [member-name]
[b]     - put file(s) before [member-name] (same as [i])
[D]     - use zero for timestamps and uids/gids
[U]     - use actual timestamps and uids/gids (default)
[N]     - use instance [count] of name
[f]     - truncate inserted file names
[P]     - use full path names when matching
[o]     - preserve original dates
[u]     - only replace files that are newer than current archive contents

generic modifiers:
[c]     - do not warn if the library had to be created
[s]     - create an archive index (cf. ranlib)
[S]     - do not build a symbol table
[T]     - make a thin archive
[v]     - be verbose
[V]     - display the version number
@<file> - read options from <file>
--target=BFDNAME - specify the target object format as BFDNAME

optional:
--plugin <p> - load the specified plugin

emulation options:
No emulation specific options
ar: supported targets: elf64-x86-64 elf32-i386 elf32-x86-64 a.out-i386-linux pei-i386 pei-x86-64 elf64
4-llom elf64-klom elf64-little elf64-big elf32-little elf32-big plugin srec symbolsrec verilog tekhex
binary ihex
Report bugs to <http://www.sourceware.org/bugzilla/>

```

uid 和 gid 差异消除指导

在编译器生成.obj文件时uid和gid属性是会随着编译用户的不同发生变化，ar工具会将该uid和pid写入.a文件中，从而导致.a文件会产生uid和gid差异。该差异不会被链接到产品的二进制文件中，如果直接交付.a文件的场景下仍会造成二进制差异。

消除方案1:

第22页,共24页Page 22.

```
linux-v07s:/opt # ar --help
Usage: ar [emulation options] [-]{dmpqrstx}[abcDfilMNoPsTuvV] [--plugin <name>] [member-name] [count]
] archive-file file...
    ar -M [<mri-script>]
commands:
d      - delete file(s) from the archive
m[ab]  - move file(s) in the archive
p      - print file(s) found in the archive
q[f]   - quick append file(s) to the archive
r[ab][f][u] - replace existing or insert new file(s) into the archive
s      - act as ranlib
t      - display contents of archive
x[o]   - extract file(s) from the archive
command specific modifiers:
[a]     - put file(s) after [member-name]
[b]     - put file(s) before [member-name] (same as [i])
[D]     - use zero for timestamps and uids/gids
[U]     - use actual timestamps and uids/gids (default)
```

文件权限属性差异消除指导

差异现象:

ar工具文件打包生成.a文件时，会将被打包文件的权限属性一直写入.a文件中。

000000f0 6f 64 65 5f 70 00 63 6f 70 79 69 6e 2e 6f 2f 20	ode_p.copyin.o/	000000f0 6f 64 65 5f 70 00 63 6f 70 79 69 6e 2e 6f 2f 20	ode_p.copyin.o/
00000c00 20 20 20 20 20 20 31 35 33 36 32 38 39 37 39 33	1536289793	00000c00 20 20 20 20 20 20 31 35 33 36 32 38 39 37 39 33	1536289793
00000c10 20 20 30 20 20 20 20 20 20 20 20 20 20 31 30	0 0 10	00000c10 20 20 30 20 20 20 20 20 20 20 20 20 20 31 30	0 0 10
00000c20 30 37 35 35 20 20 39 33 34 37 32 20 20 20 20 20	0755 93472	00000c20 30 37 37 37 20 20 39 33 34 37 32 20 20 20 20 20	0777 93472

消除方案:

在使用ar工具打包前，设置被打包文件权限，保证两次构建时文件属性完全一致。如
`chmod 755 *.o`

6 压缩、打包工具

tar

差异现象:

SUSE 12操作系统自带的tar程序的默认的pax-option是posix，在生成的压缩文件中增加一个paxheader的目录，目录名会有随机数差异。

名字 (N)	大小 (Z)	修改 (M)	名字 (N)	大小 (Z)	修改 (M)
Bep_Env_For_Linux	1,139,546	2018/9/14	PaxHeaders.61867	90	2018/9/14
PaxHeaders.61890	90	2018/9/14	Bep_Env_For_Linux	90	2018/8/28
Bep_Env_For_Linux	90	2018/8/28	Bep_Env_For_Linux	1,139,546	2018/9/14

该目录下会生成一个文件用于保存文件的创建时间、修改时间和访问时间。

2018/8/28 14:17:24	90 字节	<默认>	ANSI	UNIX	2018/8/28 14:17:24	90 字节	<默认>	ANSI	UNIX
30	mtime=1535437043.695769228				30	mtime=1535437043.695769228			
30	atime=1536926912.757993015				30	atime=1535459651.303769228			
30	ctime=1535459601.755769228				30	ctime=1535459601.755769228			

解决方案:

使用tar自带的参数--format=gnu，tar程序将不再生成paxheader文件信息。如:

`tar zcvf output.tar.gz *.o --format=gnu`

cpio

差异现象：

cpio用于创建、解压归档文件，也可以对归档文件执行拷入拷出的动作，即向归档文件中追加文件，或从归档文件中提取文件。cpio工具会将工具压缩打包时的时间戳写入生成的二进制文件，同时还会把文件的inode信息写入二进制文件，造成二进制差异。

03830070 31 43 30 30 30 30 31 45 44 30 30 30 30 30 1C000081ED000000	03830070 31 44 30 30 30 30 31 45 44 30 30 30 30 30 1D000081ED000000
038415E0 30 37 30 37 30 31 30 30 31 39 34 30 31 44 30 30 0707010019401D00	038415E0 30 37 30 37 30 31 30 30 31 39 34 30 31 45 30 30 0707010019401E00
038448A0 31 39 34 30 31 45 30 30 30 30 38 31 45 44 30 30 19401E000081ED00	038448A0 31 39 34 30 31 46 30 30 30 30 38 31 45 44 30 30 19401F000081ED00
0385ADF0 30 31 30 30 31 39 34 30 31 46 30 30 30 30 38 31 010019401F000081	0385ADF0 30 31 30 30 31 39 34 30 32 30 30 30 30 38 31 0100194020000081
0385E040 32 31 30 30 30 30 38 31 45 39 30 30 30 30 30 21000081E9000000	0385E040 32 32 30 30 30 30 38 31 45 39 30 30 30 30 30 22000081E9000000
0386BE00 30 37 30 37 30 31 30 30 31 39 34 30 32 30 30 0707010019402000	0386BE00 30 37 30 37 30 31 30 30 31 39 34 30 32 31 30 30 0707010019402100
038A1C50 32 33 30 30 30 30 38 31 45 44 30 30 30 30 30 23000081ED000000	038A1C50 32 34 30 30 30 30 38 31 45 44 30 30 30 30 30 24000081ED000000
038A2500 32 32 30 30 30 30 38 31 43 39 30 30 30 30 30 22000081C9000000	038A2500 32 33 30 30 30 30 38 31 43 39 30 30 30 30 30 23000081C9000000
03CD6C40 31 35 30 30 30 30 38 31 45 44 30 30 30 30 30 15000081ED000000	03CD6C40 31 36 30 30 30 30 38 31 45 44 30 30 30 30 30 16000081ED000000
03CD6F60 30 31 30 30 31 39 34 30 31 34 30 30 30 30 38 31 0100194014000081	03CD6F60 30 31 30 30 31 39 34 30 31 35 30 30 30 30 38 31 0100194015000081
045CE9A0 31 39 34 30 30 46 30 30 30 30 38 31 45 44 30 1940 0F000081ED00	045CE9A0 31 39 34 30 31 30 30 30 30 30 38 31 45 44 30 194010 000081ED00
045D061F 30 31 30 30 30 30 30 31 45 44 30 30 30 30 30 01000081ED000000	045D061F 30 31 31 30 30 30 30 38 31 45 44 30 30 30 30 30 011000081ED000000
045D228F 0A 30 37 30 37 30 31 30 30 31 39 34 30 31 32 30 .070701001940120	045D228F 0A 30 37 30 37 30 31 30 30 31 39 34 30 31 33 30 .070701001940130

解决方案1：

cpio2.12版本提供了--device-independent, --reproducible和--ignore-devno选项用于构建一致性，在生成cpio文件时忽略设备编号。用法如下：

```
find ./ *.docx | cpio --reproducible -ov -F a.cpio
```

gzip

差异现象：

gzip命令用来压缩文件。gzip是个使用广泛的压缩程序，文件经它压缩过后，其名称后面会添加“.gz”扩展名。gzip工具会将时间戳写入生成的文件中。两侧第5~8个字节是时间戳信息。

D:\...04_ATIA_Teamwork\09---二进制分析技术\gzip\afia\p2010_ulmage_atia - 副本	D:\...003_ATIA\04_ATIA_Teamwork\09---二进制分析技术\gzip\master\p2010_ulmage_master - 副本
2017/7/21 16:41:51 1,873,420 字节	2017/7/21 16:42:25 1,873,419 字节
00000000 1F 88 08 08 5F 41 68 59 02 03 76 6D 6C 69 6E 75AhY..vmlinu	00000000 1F 88 08 08 13 6E 37 59 02 03 76 6D 6C 69 6E 75n7Y..vmlinu
00000010 78 2E 62 69 6E 2E 33 32 34 35 30 00 E4 5A 0D x.bin.32450 .缀.	00000010 78 2E 62 69 6E 2E 31 35 30 36 00 E4 5A 0D x.bin.1 506 .缀.

解决方案1：

gzip提供了-n参数，去掉压缩生成的文件中的时间戳信息。

```
find ./ *.obj | gzip -n -f -9 > output.gz
```