УДК 004.05
doi:10.21685/2307-4205-2021-3-7

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КОНТРОЛЯ ФАЙЛОВОЙ ИЗБЫТОЧНОСТИ И ПОДТВЕРЖДЕНИЯ ПОЛНОТЫ ИСХОДНЫХ ТЕКСТОВ НА УРОВНЕ ФАЙЛОВ

В. В. Самаров

ООО «16 НИИЦ», Мытищи, Россия
samarov_vladimir@mail.ru

Аннотация. *Актуальность и цели*. Описан пользовательский интерфейс и порядок работы с разработанным программным модулем, осуществляющим контроль полноты и отсутствия файловой избыточности в исследуемом, в рамках сертификационных испытаний, программном изделии. *Материалы и методы*. При проведении анализа сравниваются два множества: перечень файлов с исходными текстами; протокол сборки (компиляции) исследуемых исходных текстов в соответствующий дистрибутивный пакет (загрузочные модули), полученный при использовании системы аудита операционных систем семейства Linux (далее – ОС Linux). *Результаты и выводы*. Использование унифицированных протоколов аудита позволило реализовать программный алгоритм, при котором имеется возможность контроля файловой избыточности и установления полноты на уровне файлов, для исследуемых проектов любых объемов и структуры, а также вне зависимости от используемых для их сборки (компиляции) схем.

Ключевые слова: сертификационные испытания программных изделий, контроль полноты и отсутствия избыточности на уровне файлов, система аудита ОС Linux, автоматизация сертификационных испытаний

Для цитирования: Самаров В. В. Программная реализация контроля файловой избыточности и подтверждения полноты исходных текстов на уровне файлов // Надежность и качество сложных систем. 2021. № 3. С. 57–61. doi:10.21685/2307-4205-2021-3-7

SOFTWARE IMPLEMENTATION OF FILE REDUNDANCY CONTROL AND CONFIRMATION OF THE COMPLETENESS OF THE SOURCE TEXTS AT THE FILE LEVEL

V.V. Samarov

LLC "16 NIITS", Mytishchi, Russia
samarov_vladimir@mail.ru

Abstract. *Background.* describes the user interface and the procedure for working with the developed software module that monitors the completeness and absence of file redundancy in the investigated software product within the framework of certification tests. *Materials and methods*. During the analysis, two sets are compared: a list of files with source texts; the protocol of assembly (compilation) of the source texts under investigation into the corresponding distribution package (load modules) obtained using the audit system of operating systems of the Linux family (hereinafter referred to as Linux OS). *Results and conclusions*. The use of unified audit protocols made it possible to implement a software algorithm in which it is possible to control file redundancy and establish completeness at the file level for the projects under study of any size and structure, as well as regardless of the schemes used for their assembly (compilation).

Keywords: certification testing of software products, control of completeness and absence of redundancy at the file level, audit system of Linux OS, automation of certification tests

For citation: Samarov V.V. Software implementation of file redundancy control and confirmation of the completeness of the source texts at the file level. *Nadezhnost' i kachestvo slozhnykh sistem = Reliability and quality of complex systems.* 2021;(3):57–61. (In Russ.). doi:10.21685/2307-4205-2021-3-7

Использование механизмов, предоставляемых системой аудита ОС Linux, начиная с версии 2.6 [1], позволило разработать алгоритм [2] по формированию унифицированного протокола (рис.1) компи-

ляции (сборки), исследуемого в рамках сертификационных испытаний программного изделия, проводимых в системе сертификации Минобороны России.

```
File Report
============================================
# date time file syscall success exe auid event
============================================
1. 22.03.2021 18:38:38 /home/sam/test/ open yes /bin/bash sam 218
2. 22.03.2021 18:38:39 /home/sam/test/ open yes /bin/bash sam 219
3. 22.03.2021 18:38:39 /home/sam/test/ open yes /bin/bash sam 220
4. 22.03.2021 18:38:38 /home/sam/test/ open yes /bin/bash sam 217
5. 22.03.2021 18:38:42 /home/sam/test/ open yes /bin/bash sam 222
6. 22.03.2021 18:38:42 /home/sam/test/ open yes /bin/bash sam 221
7. 22.03.2021 18:38:54 makefile open yes /usr/bin/make sam 224
8. 22.03.2021 18:38:54 . open yes /usr/bin/make sam 223
9. 22.03.2021 18:38:54 hello.c open yes /usr/lib/gcc/x86_64-linux-gnu/6/cc1 sam 225
10. 22.03.2021 18:38:55 main.c open yes /usr/lib/gcc/x86_64-linux-gnu/6/cc1 sam 226
```

Рис. 1. Формат унифицированного протокола компиляции (сборки)
на примере фрагмента соответствующего протокола для тестового проекта

В свою очередь, возможность получения протокола компиляции (сборки), имеющего унифицированную форму для любых (не зависящих от используемых схем и средств сборки) компилируемых (собираемых) из исходных текстов программных изделий, позволила коллективу испытательной лаборатории ООО «16 НИИЦ», под руководством и при непосредственном участии автора статьи, разработать программу, осуществляющую контроль файловой избыточности и устанавливающую факт полноты анализируемого пакета с исходными текстами.

Действительно, имея унифицированный формат протокола сборки (рис. 1), имеется возможность выполнять данный этап проверок с учетом следующих условий:

– наличия в анализируемом проекте файлов исходных текстов с одинаковыми именами, но находящихся в разных каталогах (актуально для больших проектов);

– исключения из анализа «ложных» срабатываний (событий, непосредственно не связанных со сборкой проекта, но формирующих в протоколе сборки семантический «след» того или иного файла с исходными текстами), т.е. анализ только успешных событий по сборке;

– регистра имен файлов и соответствующих к ним путей в соответствии с особенностью файловых систем семейства типа extfs[1] (ext2–ext4) [3].

В целях осуществления платформенной совместимости программа была разработана на языке Python, версии 3.4.3 [5], что позволяет без модификации использовать ее в ОС Windows (версий XP и выше), а также на большинстве операционных систем семейства Linux, имеющих соответствующие сертификаты (заключения) о соответствии безопасности информации, выданные Минобороны России.

Программа позволяет:

– успешно обрабатывать «смешанные» входные данные, т.е. на вход программы может быть подан файл, представляющий собой перечень файлов проекта (исходных текстов), сформированный любым удобным способом как в ОС семейства Windows, так в Linux системах;

– формировать информативные и удобочитаемые отчеты в текстовом формате, а также в формате html;

– успешно и быстро обрабатывать входные данные значительных объемов.

Алгоритм работы с программой:

1. Подготовить исходные данные, в том числе:

– произвести построение списков с файлами исходных текстов (для чего воспользоваться любым удобным средством, например ПС «Total Commander» или любым другим). Файл с перечнем исходных текстов может быть в любой стандартной кодировке и должен представлять собой тексто-

[1] Ввиду того, что в ядро ОС Linux для файловой системы ext4 была включена поддержка работы без учета регистра символов [4], то для корректной работы программного модуля была предусмотрена возможность такого анализа.

вый файл. Формат входного файла с перечнем исходных текстов представляет собой массив со строками[1] (рис. 2);

– сформировать протокол компиляции анализируемого проекта, для чего воспользоваться алгоритмом [1] по его (протокола) созданию с помощью системы аудита ОС семейства Linux;

– при необходимости установить на сборочную систему соответствующую службу (auditd);

– осуществить постановку на контроль (аудит) всех обращений к объектам (файлам с анализируемыми исходными текстами), находящихся в соответствующем каталоге (команда: `$ sudo auditctl – w /Dest/for/src –p rwxa`, где */Dest/for/src* – абсолютный путь к каталогу с анализируемыми исходными текстами);

– произвести штатную сборку (в соответствии с действиями, описанными в соответствующей программной документации программного изделия) исследуемого проекта;

– с помощью утилиты aureport сформировать соответствующий (см. рис.1) протокол сборки (команда: `$ sudo aureport –f –i – success > log_file,` где *log_file* – имя файла, содержащего записи, протоколирующие процесс компиляции (сборки) тестового проекта).

```
J:\exmp\TEST\assembly_TEST\data\app\data\Application.cpp
J:\exmp\TEST\assembly_TEST\data\app\data\application.h
…….
J:\exmp\TEST\assembly_TEST\data\app\data\main.cpp
```

Рис. 2. Пример представления перечня анализируемых файлов исходных текстов

2. В зависимости от используемой ОС запустить соответствующий интерпретатор командной строки (например, cmd – для ОС Windows или bash терминал – для ОС Linux).

3. Перейти в каталог с файлами программы, после чего выполнить команду вида

```
C:\Test\files_cmpr\>python app.py --first=data/SRC_test.txt --
second=data/log_file --dest_path= assembly_TEST --reg=1 --exts="cpp|h" // –
```

для ОС семейства Windows;

```
user@deb:~$ python3 app.py --first=data/ SRC_test.txt --
second=data/log_file --dest_path= assembly_TEST --reg=1 --exts="cpp|h" // –
```

для ОС семейства Linux,

где `first` – путь[2] к файлу с перечнем файлов исходных текстов; `second` – путь[3] к файлу с протоколом компиляции проекта; `dest_path`[3] – имя каталога (верхнего каталога, включая подкаталоги) с анализируемыми исходными текстами; `reg` – параметр, отвечающий за правило сравнения (1 – учитывать регистр символов в названии файлов; 0 – не учитывать регистр символов в названии файлов); `exts` – маска поиска (через символ "|" перечисляются расширения файлов, по которым осуществляется анализ).

4. Перейти в каталог «results» программы и ознакомиться с сформированными отчетными материалами, в том числе:

– с файлами Rep_equals.txt (отчет по одинаковым файлам), Rep_file1_only.txt (отчет по избыточным файлам), Rep_file2_only.txt (отчет по файлам, не отвечающим требованиям по полноте исходных текстов) (рис. 3);

– со сводным html отчетом (рис. 4).

[1] Для корректной работы программы требуется указание абсолютных путей к соответствующим файлам исходных текстов.

[2] В приведенном примере указан относительный путь к сравниваемым файлам, находящимся в каталоге «data/» программы.

[3] Для получения релевантных результатов анализа необходимо, чтобы имя каталога с анализируемыми исходными текстами из файла с исходными текстами коррелировало с именем каталога с исходными текстами, участвовавшего в сборке проекта.

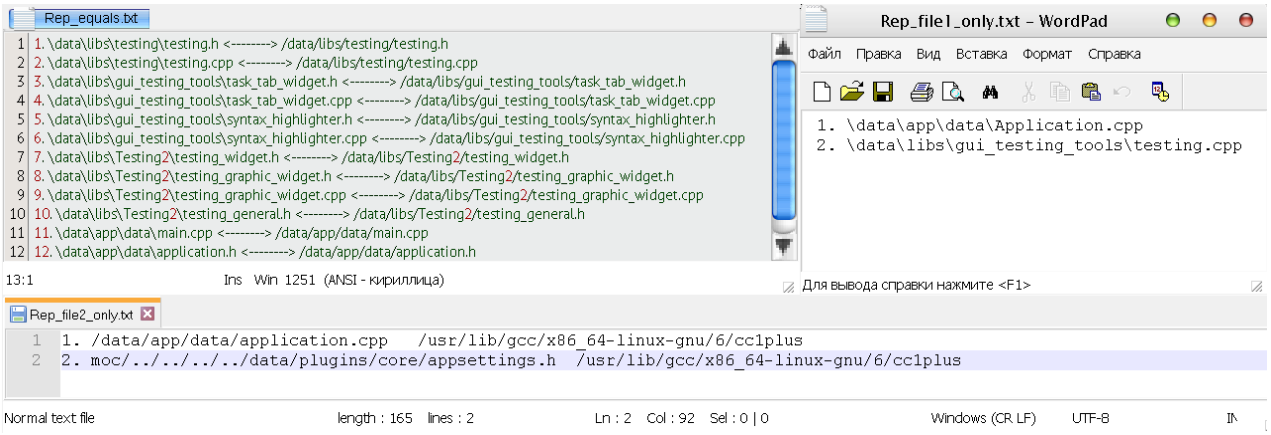

Рис. 3. Видеокадр сформированных по результатам анализа тестового примера файлов

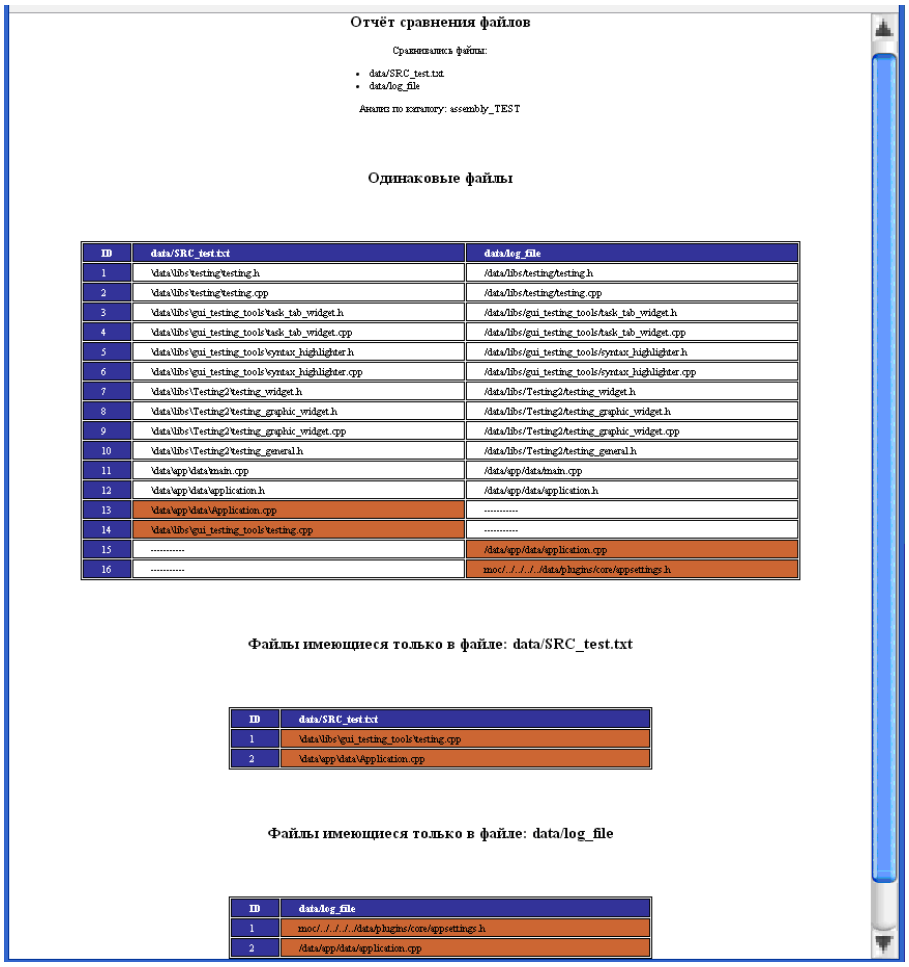

Рис. 4. Видеокадр сводного html отчета по файловой избыточности и полноте тестового примера

Заключение

В заключении хотелось бы отметить, что описанный в настоящей статье программный модуль по контролю полноты и отсутствия избыточности на уровне файлов успешно апробирован специалистами испытательной лабораторией ООО «16 НИИЦ» при проведении ими сертификационных испытаний программных изделий.

Вместе с тем интерпретация выходных данных программного модуля в части файлов, которые по результатам анализа помечены как не соответствующие требованиям по полноте исходных текстов (отчет – Rep_file2_only.txt (см. рис. 3) в третьей таблице в сводном html отчете (рис. 4)), требует от специалистов, проводящих испытания, достаточно серьезных знаний в современных методах и

способах компиляции исходных текстов (например, принципов работы метаобъектного компилятора на основе библиотеки qt [6] или метаобъектных протоколов (например, Google Protobuff [7]), а также зачастую значительных временных затрат. Таким образом, по мнению автора, целесообразно проработать вопрос о расширении функционала программного модуля в части добавления в него механизмов по предоставлению дополнительной информации, использование которой позволит упростить принятие решения по таким (генерируемым в процессе компиляции) файлам.

Список литературы

1. Пингвин под колпаком: Аудит ситемных событий в линукс // Хакер. URL: https://xakep.ru/2011/03/30/54897 (дата обращения: 26.02.2021).
2. Самаров В. В. Использование системы аудита операционных систем семейства Linux при проведении сертификационных испытаний программных изделий // Надежность и качество сложных систем. 2021. № 1. С. 144–150.
3. Типы файловых систем для Linux. URL: https://losst.ru/tipy-fajlovyh-sistem-dlya-linux (дата обращения: 02.04.2021).
4. В ядро Linux для ФС Ext4 включена поддержка работы без учета регистра символов. URL: https://www.opennet.ru/opennews/art.shtml?num=50581 (дата обращения: 02.04.2021).
5. Python Release 3.4.3. URL: https://www.python.org/downloads/release/python-343 (дата обращения: 02.04.2021).
6. Использование метаобъектного компилятора (Meta-Object Compiler, moc). URL: http://doc.crossplatform.ru/qt/4.6.x/moc.html (дата обращения: 02.04.2021).
7. Language Guide – Protocol Buffers – Google Developers. URL: https://developers.google.com/protocol-buffers/docs/overview (дата обращения: 03.04.2021).

References

1. Pingvin pod kolpakom: Audit sitemnykh sobytiy v linuks. *Khaker = "Penguin under the hood: Audit of system events in Linux" // Hacker*. (In Russ.). Available at: https://xakep.ru/2011/03/30/54897 (accessed 26.02.2021).
2. Samarov V.V. The use of an audit system for Linux operating systems during certification tests of software products. *Nadezhnost' i kachestvo slozhnykh system = Reliability and quality of complex systems*. 2021;(1):144–150. (In Russ.).
3. *Tipy faylovykh sistem dlya Linux = Types of file systems for Linux*. (In Russ.). Available at: https://losst.ru/tipy-fajlovyh-sistem-dlya-linux (accessed 02.04.2021).
4. *V yadro Linux dlya FS Ext4 vklyuchena podderzhka raboty bez ucheta registra simvolov = The Linux kernel for FS Ext4 includes support for case-insensitive operation*. (In Russ.). Available at: https://www.opennet.ru/opennews/art.shtml?num=50581 (accessed 02.04.2021).
5. *Python Release 3.4.3*. Available at: https://www.python.org/downloads/release/python-343 (accessed 02.04.2021).
6. *Ispol'zovanie metaob"ektnogo kompilyatora (Meta-Object Compiler, moc) = Using the Meta-Object compiler (Meta-Object Compiler, moc)*. (In Russ.). Available at: http://doc.crossplatform.ru/qt/4.6.x/moc.html (accessed 02.04.2021).
7. *Language Guide – Protocol Buffers – Google Developers*. Available at: https://developers.google.com/protocol-buffers/docs/overview (accessed 03.04.2021).

Информация об авторах / Information about the authors

Владимир Владимирович Самаров
заместитель начальника испытательной лаборатории,
ООО «16 НИИЦ»
(Россия, Московская обл., г. Мытищи,
Новомытищинский просп., 19)
E-mail: samarov_vladimir@mail.ru

Vladimir V. Samarov
deputy head of test laboratory,
LLC "16 NIITS "
(19 Novomytischinskiy avenue, Mytischi,
Moscow region, Russia)

Авторы заявляют об отсутствии конфликта интересов /
The authors declare no conflicts of interests.

Поступила в редакцию/Received 19.06.2020
Поступила после рецензирования/Revised 21.07.2021
Принята к публикации/Accepted 13.09.2021