**Raw-cloner Factory and Acceptance Test**

*Mike Mountrakis* [*mike.mountrakis@gmail.com*](mailto:mike.mountrakis@gmail.com)

*Author: Minos Saltiel <your mail>*

*Version 1.00 Status: Draft*

***Abstract****: raw-cloner is a Linux/Windows command line utility for cloning/copying files and disks. The present document is the Factory and Acceptance Test that summarizes the QoS of the program raw-cloner created and maintained by Apelait Ldt.*

Contents

[Abbreviations 3](#_Toc85263883)

[Introduction 4](#_Toc85263884)

[Alpha Test 5](#_Toc85263885)

[System Integration 5](#_Toc85263886)

[Configuration Test 7](#_Toc85263887)

[Basic Functional Test 8](#_Toc85263888)

[Robust and Durability Test 9](#_Toc85263889)

[Performance Test 10](#_Toc85263890)

## Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Definition - Meaning** |
| CPU | Central Processing Unit |
| RAM | Random Access Memory – the media: disk / file |
| OS | Operating System |
| S/W | Software |
| QoS | Quality of Service |
| CLI | Command Line Interface |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Introduction

raw-cloner – **the application** under test - is a Linux/Windows command line utility for cloning/copying files and disks. The present document is the Factory and Acceptance Test that summarizes the QoS of the program raw-cloner created and maintained by **Apelait Ldt**.

In the following pages, the author will summarize the procedures for Factory and Acceptance of the application until it meets the standards set by **Apelait Ldt** for QoS. The main topics covered in this document will cover the following topics:

* **Alpha Test**
  + **System Integration Test** – compilation under various O/S Windows or Linux
    - Linux
    - Windows
  + **Configuration Test** – CLI parser of user defined options
  + **Basic Functional Test**
    - Test Copy Forward
    - Test Copy Forward to Image
    - Test Copy Backwards
    - Test Features: log correct format, stats, program messages, everything
  + **Robust/Durability Test**
    - Test user malfunction configuration. If user wants to break the program.
* **Beta and Performance Test**
  + Compilation with Compiler Optimization Flags - O3
  + Disk Clone Forward
  + Disk Clone Forward to Image
  + Disk Copy Backwards
  + Performance Features

## Alpha Test

Alpha Test is the elementary test operation of the application. It is the first series of tests that the QoS performs in order to check if the system/application under test meets what is supposed to do. It tests all functional units (integration, configuration, use cases) to prove that all operations that will be performed by the end user will be free of malfunctions and there will be no problem.

### System Integration

During the System Integration Test procedure, our aim is to ensure the application integrates correctly with the O/S and other 3rd party systems it cooperates. For the given application, you must try to compile the application on various platforms using the directives given by the application engineer’s documentation and provide feedback.

#### Compile under Windows

|  |  |
| --- | --- |
| **Label** | **Integration 1.1** |
| **O/S Version** | Microsoft Windows 10 Enterprise  Version 10.0.19042 Build 19042 |
| **Compiler Info** | gcc version 9.2.0 (tdm64-1) |
| **Method** | Makefile.win |
| **Required Tools** | Gxx.exe make.exe / Ming suite |
| **Tester**  **Comments** | Compiles without any problem, not even warning. Produces x86\_64-w64-mingw32 executable  Static compilation produces **raw-cloner.exe** |
| Result | PASSED |

#### Compile under Linux

Compile under various Linux distribution, to verify that the application is portable

|  |  |
| --- | --- |
| **Label** | **Integration 1.2** |
| **O/S Version** | ubuntu1~18.04  146-Ubuntu SMP Tue Apr 13 01:11:19 UTC 2021 |
| **Compiler Info** | 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04) |
| **Method** | Makefile |
| **Required Tools** | Gcc, ld, make GNU Suite |
| **Tester**  **Comments** | Produces  ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,  Linux Dynamic compilation produces **raw-cloner** |
| **Result** | **PASSED** |

Write your own compilation test case for the Linux distribution you test.

### Configuration Test

Configuration Test proves that all system parametrization enters the application correctly and it is used by the application correctly. For the given application, all parameters provided by the user to the CLI are correctly passed to and used by the application.

|  |  |
| --- | --- |
| **Label** | **Configuration 2.1** |
| **Scenario** | Example: Test copy forward from a start offset to end offset using a buffer size |
| **O/S Version** |  |
| **Compiler Info** |  |
| **Method** | CLI |
| **Required Tools** | - |
| **Command** | Put the command you have used |
| **Tester**  **Comments** | Warnings, problems... |
| **Result** | PASSED or FAILED |

|  |  |
| --- | --- |
| **Label** | Configuration 2.2 |
| **Scenario** | Example: your scenario |
| **O/S Version** |  |
| **Compiler Info** |  |
| **Method** | CLI |
| **Required Tools** | - |
| **Command** | Put the command you have used |
| **Tester**  **Comments** | Warnings, problems... |
| **Result** | PASSED or FAILED |

### Basic Functional Test

Ensure the application delivers all functionality it promises and works as expected. To do so you must test the application with a **small and easy to read file** and examine if the copy produced is valid, if the logs produced are accurate, if the statistics produced are ok. Some test scenarios:

* Copy an ASCII text test file forward with a small buffer size
* Copy an ASCII text test file forward with a small buffer size from a given start offset
* Copy an ASCII text test file forward with a small buffer size to a given end offset
* Copy an ASCII text test file forward with a small buffer size from a given start offset to a given end offset
* Copy an ASCII text test file forward with a small buffer and some retries
* Copy an ASCII text test file backwards
* All the above cases but using BINARY FILES this time
* Use your logic and imagination

Write a test case like this:

|  |  |
| --- | --- |
| **Label** | Basic Functional 3.1 |
| **Scenario** | Example: Copy an ASCII test file forward with a small buffer size |
| **O/S Version** | Ubuntu1~18.04 |
| **Compiler Info** | gcc version 7.5.0 |
| **Method** | CLI |
| **Required Tools** | - |
| **Command** | raw-cloner -i ascii.txt -o ascii-copy.txt -l log.txt -b 7 |
| **Input Description** | base64 /dev/urandom | head -c 100 > ascii.txt  head ascii.txt  wFMelZILVXhVfwNAoUG0yMmHMua1u2pWCiGxYVPEDEI4vCKm5jKcb8RELakpSlDE2ITCfcOSEdnP  hmrkT9B56LGnjhbysd3D/Z0gbWwVayV8/LnrkwEwhP7BofnU9E8ob5lkH5r44+IYP1tFK1Q4tCNL  root@dynatrace-dev:~# ls -l ascii.txt  -rw-r--r-- 1 root root 1000 Oct 14 19:20 ascii.txt |
| **Tools used** | Diff. Cmp, vi |
| **Tester**  **Comments** |  |
| **Result** | PASSED or FAILED |

### Robust and Durability Test

The aim of Robust and Durability Test procedure is to ensure that the application understands the misuse or program abuse and does not break. See the example and use your imagination how one can abuse and try to break the application.

Example

|  |  |
| --- | --- |
| **Label** | Durability 4.1 |
| **Scenario** | Example: Copy an ASCII test file forward with a small buffer size start\_offset, end\_offset but  end\_offset < start\_offset. |
| **O/S Version** | Ubuntu1~18.04 |
| **Compiler Info** | gcc version 7.5.0 |
| **Method** | CLI |
| **Required Tools** | - |
| **Command** | raw-cloner -i ascii.txt -o asci-copy.txt -s 500 -e 200 -b 100 |
| **Input Description** | base64 /dev/urandom | head -c 100 > ascii.txt  head ascii.txt  wFMelZILVXhVfwNAoUG0yMmHMua1u2pWCiGxYVPEDEI4vCKm5jKcb8RELakpSlDE2ITCfcOSEdnP  hmrkT9B56LGnjhbysd3D/Z0gbWwVayV8/LnrkwEwhP7BofnU9E8ob5lkH5r44+IYP1tFK1Q4tCNL  root@dynatrace-dev:~# ls -l ascii.txt  -rw-r--r-- 1 root root 1000 Oct 14 19:20 ascii.txt |
| **Tools used** | Diff. Cmp, vi |
| **Tester**  **Comments** |  |
| **Result** | PASSED or FAILED |

## Performance Test

At this point, you know that the application works as expected since in the previous section, you have tested all the application's functionality. From now on, you test the application with real use cases. To do so you first have to recompile the program with performance flags. Edit the Makefile, and replace -g with -O3. Use: make clean. Then: make; make install. This will recompile and install the application with maximum optimization, ready to serve real use cases.

You have to test the application with a **real drive or USB that does not have problems first and then with a faulty drive** and examine if the copy produced is valid, if the logs produced are accurate, if the statistics produced are OK. Some test scenarios:

* Copy /dev/sdb to /dev/sdc forward with a buffer of 1 MB. Copy is valid? All files with same sizes there?Logs OK, stats OK? Time required? Observe the system load – is the platform affected?
* Copy /dev/sdb to /dev/sdc forward with a buffer of 512 MB. Copy is valid? All files with same sizes there?Logs OK, stats OK? Time required?
* Copy /dev/sdb to /dev/sdc backwards. Copy is valid? All files with same sizes there?Logs OK, stats OK? Time required? Observe the system load – is the platform affected?
* ,,,,
* ,,,,
* Find a scenario to test thermal protection.

Example

|  |  |
| --- | --- |
| **Label** | Performance 4.1 |
| **Scenario** | Example: Copy of /dev/sdb to /dev/sdc with 1MB buffer |
| **O/S Version** | Ubuntu1~18.04 |
| **Compiler Info** | gcc version 7.5.0 |
| **Method** | CLI |
| **Required Tools** | Find, ls, |
| **Command** | raw-cloner -i /dev/sdb -o /dev/sdc -l log.txt -b 1048576 |
| **Input Description** | # df -h /dev/sdb  Filesystem Size Used Avail Use% Mounted on  devtmpfs 2.0G 0 2.0G 0% /dev  # df -h /dev/sdc  Filesystem Size Used Avail Use% Mounted on  devtmpfs 2.0G 0 2.0G 0% /dev  Both disks are SSD... |
| **Tools used** | find, dd... |
| **Tester**  **Comments** |  |
| **Result** | PASSED or FAILED |