

Question

Imagine that we have 2**48 text files. Explain how can we find which files are the same.

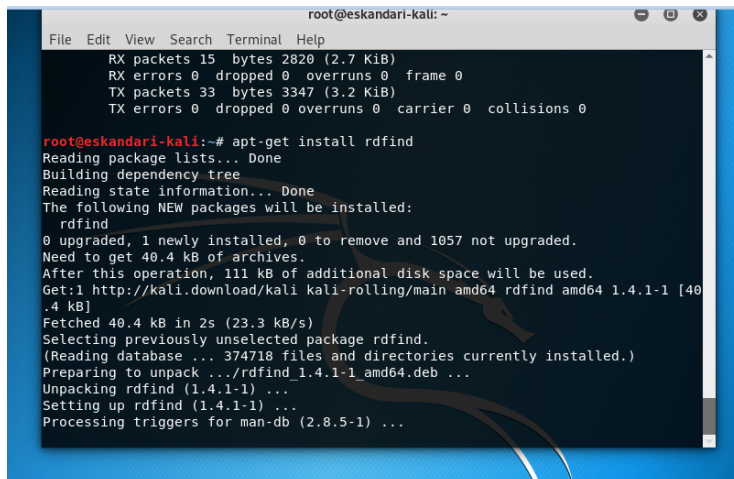
Method 1-Short Answer

Rdfind comes from redundant data find. It is a free tool used to find duplicate files across or within multiple directories. It uses checksum and find duplicates based on file contains not only names.

Rdfind uses algorithm to classify the files and detects which of the duplicates is the original file and considers the rest as duplicates.

Method 1-Long Answer

1- install rdfind in Linux

A screenshot of a terminal window titled 'root@eskandari-kali: ~'. The terminal shows the output of the command 'apt-get install rdfind'. The output includes network statistics at the top, followed by the installation process: reading package lists, building a dependency tree, and installing the 'rdfind' package. It shows that 40.4 kB of archives are needed and that the package is being unpacked and set up. The terminal text is as follows:

```
root@eskandari-kali: ~
File Edit View Search Terminal Help
RX packets 15  bytes 2820 (2.7 KiB)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 33  bytes 3347 (3.2 KiB)
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

root@eskandari-kali:~# apt-get install rdfind
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  rdfind
0 upgraded, 1 newly installed, 0 to remove and 1057 not upgraded.
Need to get 40.4 kB of archives.
After this operation, 111 kB of additional disk space will be used.
Get:1 http://kali.download/kali kali-rolling/main amd64 rdfind amd64 1.4.1-1 [40.4 kB]
Fetched 40.4 kB in 2s (23.3 kB/s)
Selecting previously unselected package rdfind.
(Reading database ... 374718 files and directories currently installed.)
Preparing to unpack .../rdfind_1.4.1-1_amd64.deb ...
Unpacking rdfind (1.4.1-1) ...
Setting up rdfind (1.4.1-1) ...
Processing triggers for man-db (2.8.5-1) ...
```

Now, we run rdfind on a directory, by simply typing rdfind and the target directory. Here is an example:

```
File Edit View Search Terminal Help
root@eskandari-kali:~#
root@eskandari-kali:~# ls /pulse/e666276dd8ad47defb9a4acd4478107b-default-sink
db.sh  Documents  first.txt  hello.c  Pictures  second.txt  Videos  default-source
Desktop Downloads hello     Music    Public   second.txt  Templates
root@eskandari-kali:~# rdfind -c /etc/ld.so.conf.d/ldconfig.conf -r /usr/share/doc/ldconfig/ldconfig.conf
Now scanning "/usr/share/doc/ldconfig/ldconfig.conf", found 671 files.
Now have 671 files in total.
Removed 0 files due to nonunique device and inode.
Total size is 111324173 bytes or 106 MiB.
Removed 488 files due to unique sizes from list.183 files left.
Now eliminating candidates based on first bytes:removed 107 files from list.76 files left.
Now eliminating candidates based on last bytes:removed 37 files from list.39 files left.
Now eliminating candidates based on sha1 checksum:removed 10 files from list.29 files left.
It seems like you have 29 files that are not unique.
Totally, 55 KiB can be reduced.
Now making results file results.txt
root@eskandari-kali:~# ls
db.sh  Documents  first.txt  hello.c  Pictures  results.txt  Templates  Videos
Desktop Downloads hello     Music    Public   second.txt  Templates  Videos
root@eskandari-kali:~# vi result.txt
root@eskandari-kali:~#
```

As you can see rdfind will save the results in file called results.txt located in the same directory from where you ran the program. The file contains all the duplicate files that rdfind has found.

```
Application Workstation
Open results.txt To release input, press Ctrl+Shift. Save
# Automatically generated
# duptype id depth size device inode priority name
DUPTYPE FIRST OCCURRENCE 36 2 1 2049 941110 1 ./config/pulse/e666276dd8ad47defb9a4acd4478107b-default-sink
DUPTYPE WITHIN SAME_TREE -36 2 1 2049 941131 1 ./config/pulse/e666276dd8ad47defb9a4acd4478107b-default-source
DUPTYPE FIRST OCCURRENCE 20 2 11 2049 1119960 1 ./cache/tracker/db-locale.txt
DUPTYPE WITHIN SAME_TREE -20 2 11 2049 1119972 1 ./cache/tracker/ldconfig.conf
DUPTYPE FIRST OCCURRENCE 109 5 16 2049 312514 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/allow-flash-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312639 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/block-flash-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312620 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/test-harmful-simple.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312617 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/base-track-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312537 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/except-flash-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312536 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/test-unwanted-simple.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312531 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/test-track-simple.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312525 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/test-phish-simple.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312521 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/mozplugin-block-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312519 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/mozstd-trackwhite-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312523 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/block-flashsubdoc-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312511 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/except-flashallow-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312510 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/test-block-simple.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312508 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/test-trackwhite-simple.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312512 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/except-flashsubdoc-digest256.pset
DUPTYPE WITHIN SAME_TREE -109 5 16 2049 312507 1 ./cache/mozilla/firefox/0c1tyb2q.default/safebrowsing/test-malware-simple.pset
DUPTYPE FIRST OCCURRENCE 6 0 56 2049 789934 1 ./first.txt
DUPTYPE WITHIN SAME_TREE -6 0 56 2049 789940 1 ./hello.c
DUPTYPE WITHIN SAME_TREE -6 0 56 2049 789941 1 ./second.txt
DUPTYPE FIRST OCCURRENCE 145 5 3162 2049 312096 1 ./cache/mozilla/firefox/0c1tyb2q.default/startupCache/urlCache-current.bin
DUPTYPE WITHIN SAME_TREE -145 5 3162 2049 312357 1 ./cache/mozilla/firefox/0c1tyb2q.default/startupCache/urlCache-bin
DUPTYPE FIRST OCCURRENCE 102 4 5267 2049 312113 1 ./mozilla/firefox/0c1tyb2q.default/sessionstore-backups/previous.jsonlz4
DUPTYPE WITHIN SAME_TREE -102 4 5267 2049 312532 1 ./mozilla/firefox/0c1tyb2q.default/sessionstore-backups/upgrade.jsonlz4-20181211232904
DUPTYPE FIRST OCCURRENCE 92 4 47904 2049 312043 1 ./local/share/fonts/pcaendarFonts/Vazir.woff
DUPTYPE WITHIN SAME_TREE -92 4 47904 2049 312032 1 ./local/share/gnome-shell/extensions/PersianCalendar@oxygens.com/fonts/Vazir.woff
# end of file
```

Method 2- Short Answer:

We should store the hash value of each file in a database and then query the database for duplicate hash values. The duplicated files will be found in an easy manner, along with some other distinct files which their hash values collide with each other. Then, we could prune the result to remove collision cases.

Method 2- Long Answer:

- 1- Make a table in a database, for example, table Files_Info with three columns (ID, Path , Hash)

- 2- for each file, first we should find its hash value (for example, by running one of the MD5 or SHA1 hash functions), and then Insert the file path and its Hash to the aforementioned table. SHA1 is preferred in which the probability of collision occurrence is less than MD5.

Commands in Linux:

For example, you can download [coreutil](#) via commands below: (it is a library which contains most of hash functions)

```
% dpkg -L coreutils | grep '[0-9]sum$'
```

Now, you have these hash functions:

```
/usr/bin/sha224sum
/usr/bin/sha512sum
/usr/bin/md5sum
/usr/bin/sha1sum
/usr/bin/sha256sum
/usr/bin/sha384sum
```

Choose one of them to hash files. We prefer the SHA1SUM.

Step 2, make the aforementioned table in MySQL and then use the following bash to store the hash value of each file in the table.

```
#!/bin/bash
DB_USER='razieh';
DB_PASSWD='eskandari';

DB_NAME='test';
TABLE='Files_Info'
for filename in /Directory/*.txt; do
    Hash=`sha1sum ${filename}`
    mysql --user=$DB_USER --password=$DB_PASSWD $DB_NAME << EOF
    INSERT INTO $TABLE (id, Path, hash) VALUES (NULL, "$filename",
    "$Hash");EOF
done
```

- 3- Step 3: finally, running this query in MySQL would return the duplicate files:

```
SELECT Hash,COUNT(Hash)
FROM Files_Info
GROUP BY Hash
HAVING COUNT(Hash) > 1;
```

- 4- Step 4: since it is probable that the above hash function could have a collision among 2^{24} hash values, we should elaborate more on results. So, we check files obtained in step 3 once again to ensure duplication. Some of them may be eliminated in this step.

The algorithm used for this step is not very important, because the number of candidate files is not too many.