Securely wipe disk/Tips and tricks

< Securely wipe disk

See **Securely wipe disk** for the main article.

This article describes alternative wiping methods to the specialized utilities that can speed up wiping.

Warning: **Reserved blocks** will not be wiped by creation of the files but they can be disabled with utility `tune2fs`.

Contents

Wipe a single file

Warning: Wiping of single files is less effective, if the partition has been **defragmented**, resized or moved, or the files have been **duplicated** on the same device before. It is much harder to recovery if the whole **encrypted data** container part was wiped but encryption will lower **disk performance** much, if it is not **hardware based** encryption that has almost no impact on the performance. See also **Securely wipe disk#Data remanence**.

Note: To preserve file access and modification time you can use `touch(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/touch.1)` command to change time and `stat(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/stat.1)` command to read time information before accessing the file. See also **comparison of file systems** for support of metadata and timestamps by them.

Wiping of a single file consists of two basic and one advanced anti-forensic time consumed method that can be done only with specialized tools, the last one method will not be covered in this article.

- Overwrite with random data before deletion or replace content with another one without changing file size.
- Wipe file and meta-data stored by the filesystem with specialized tools
  - Search the whole disk for the deleted left-over parts of the file and wipe them too without making any changes to other files and their traces.

Overwriting of the file without changing its size can be done with common Linux utilities:

- Invoking `shred -x file` will replace content of *file* with pseudo-random data without changing the filesize (`-x`). Using the `-u` option will remove *file* after overwriting it.
- With `mkfs` you can convert file into the filesystem that will alter everything in it, mount and fill in with any other content for a better overwriting.
- The `dd` will create a file with preset size and content of your chose, if destination file name exist then it will become overwritten. With `dd` command you can replace the whole file or only a part in it with another content by combining `skip` and `seek` options. You need to know size of the file to avoid expand of the file, to do it can use `du -b file_name | cut -f1` or `stat -c "%s" file_name`. It is mandatory to use `iflag=fullblock` option to make it work correct with the file.
- Replace content in a file with a single symbol to avoid size changing you can do with **perl (https://www.archlinux.org/packages/?name=perl)** utility.

To wipe meta-data you can fill in partition with files that makes file system replace old entries about files with new or use specialized utilities for that, see **wipe free space** section below.

It is up to you how to combine all of Linux file creation and conversion tools to prevent recovery of files and/or mislead recovery tools and them who uses it by rewriting with random or replace with predefined content.

Note:

- Simple overwriting has a chance to leave dislocated parts of file if free space is available.
- To rewrite content of specific files without changing their location on the disk you can first create a file or files to make partition full and then rewrite or replace content in files you want to hide with preferred utilities. It consumes much useful time if free space is very big and files are very few.

Examples:

Perl command that will replace everything in the file with `.` :

```
$ perl -p -i -e 's\[^*]\.\g' file_name
```

dd:

```
$ source_content | dd bs=size_in_bytes count=1 iflag=fullblock of=destination_file seek=0
```

Or by using *stdout* redirection that works a slightly faster for creation but you will not be able to use `seek` option for skipping some parts in the destination:

```
$ source_content | dd bs=size_in_bytes count=1 iflag=fullblock > destination_file
```

Tip: If source file is lower than destination then you will need to combine it with the `while` loop described in the section below.

See also:

- **convert charset in text files (http://stackoverflow.com/questions/64860/best-way-to-convert-text-files-between-character-sets)**
- **Advanced Bash-Scripting Guide (http://tldp.org/LDP/abs/html/here-docs.html)** - describes more advanced ways about how to create files from a bash script
- **sed alternatives (http://www.computing.net/answers/unix/alternative-for-sed-i/7800.html)** - a perl example that works better than sed for replacing content in a file

Overwrite the target

Prevent wiping mounted partitions

Choosing the device to be wiped needs extra care; a simple typo can be enough to damage the system. To minimize these risks, you can use a simple script to wrap your favourite wipe tool. For example:

```
if [[ -e "$1" && -b "$1" ]];then
NOT_safe="$(lsblk -o "NAME,MOUNTPOINT" ${1//[0-9]/} | grep -e / -e '\]')";
 if [[ -z "$NOT_safe" ]];then
# Here you can use any of your favourite wiping tools
# to wipe destination passed on command line and stored in variable "$1"
#
  else
  echo 'Not allowed to destroy if any of the partitions is mounted: '"$NOT_safe"
 fi
fi
```

dd - advanced example

An alternative is to randomize the drive/partition using a randomly-seeded AES cipher from OpenSSL (displaying the optional progress meter with `pv (https://www.archlinux.org/packages/?name=pv)`). For example:

```
# openssl enc -aes-256-ctr -pass pass:"$(dd if=/dev/urandom bs=128 count=1 2>/dev/null | base64)" -nosalt </dev/zero \
    | pv -bartpes <DISK_SIZE> | dd bs=64K of=/dev/sd"X"
```

where the (optional) total disk size in bytes (`DISK_SIZE`) may be obtained via:

```
# blockdev --getsize64 /dev/sd"X"
```

```
250059350016
```

The command above creates a 128 byte encryption key seeded from `/dev/urandom`. AES-256 in CTR mode is used to encrypt `/dev/zero`'s output with the urandom key. Utilizing the cipher instead of a pseudorandom source results in very high write speeds and the result is a device filled with AES ciphertext.

The block size is set to 64K above as it is usually faster than the default 512 bytes, and yields nearly optimal transfer rates across a range of hardware: **[1] (http://superuser.com/questions/234199/good-block-size-for-disk-cloning-with-diskdump-dd)** and the references therein.

See also **Dm-crypt/Drive preparation#dm-crypt wipe on an empty disk or partition** for a similar approach.

Using a template file

Instead of zeros you can use a bunch of files you want to be found or partition prints made by `mkfs` formatted files but you should mount and fill it up with content or with any other repeated output from utilities of your choice.

One way is to wipe until device ends, but this type of redirection is not recommended because you have to use **stop keys** to break the `while` loop when errors about device end will show up:

```
$ while [ 1 -lt 2 ];do cat file1-to-use.as-template file2-to-use.as-template /tmp/templatefiles/* ;done > /dev/sd"XY"
```

With *dd* you can safely wipe repetitively without out-of-space-errors, if size to be wiped is set up correctly with options. By using *dd* inside the while loop for *stdout* you will be able to chose which part of the file you want to restore by combining the `skip` and `seek` options with random or fixed values e.g. restore only partition start or end from a file, related are `head(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/head.1)` and `tail(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/tail.1)` commands for output of the file parts to *stdout*.

```
while [ 2 -gt 1 ]; do
 if [ -z "$(pidof dd)"  ];then
  break ;
 fi;
cat file1-to-use.as-template file2-to-use.as-template /tmp/templatefiles/* ;
done | dd of=/dev/sd"XY" seek=start_sector bs=sector_size count=sectors_to_wipe
```

Tip: For repeatedly rewriting of the same area. You can also use variables from second loop around `while ... done | dd ...` to change destination of the source template files to use on each new loop.

Note: If you have a real big amount of predefined data to use for overwriting then you can use the ability of **SquashFS (http://tldp.org/HOWTO/SquashFS-HOWTO/creatingandusing.html)** to compress files and folders. Compressing them into a mounted **ramdisk (http://www.linuxscrew.com/2010/03/24/fastest-way-to-create-ramdisk-in-ubuntulinux/)** will minimize the amount of physical reads but will higher CPU usage. Or create the compressed archive, copy into the **ramdisk (http://www.linuxscrew.com/2010/03/24/fastest-way-to-create-ramdisk-in-ubuntulinux/)** and use a file extraction utility that can extract to stdout. Downsides are that it is harder to navigate between folders and files, if you do not want to use all and same content for each run. Upsides are that you can use even raw parts of the archive, if it was split into volumes upon creation.

See also:

- **sector size (http://flashdba.com/4k-sector-size/)** - file creation and sector sizes

Wipe free space

Warning:

- It is not appropriate to use before **preparations for block device encryption**.

You can wipe free space by several ways:

- **Redirect output** into a file instead of partition or device.
- Create multiple file copies by using e.g. `cp` command in loops with random file names or destination directories until no free space will be left.
- Use an utility that creates encrypted files with random password and file names.

- Use a specialized program for the free space wiping such as wipefreespace (https://aur.archlinux.org/packages/wipefreespace/)AUR.

Some of the file compression utilities use to have many types of the compression methods, file types and can even split file into volumes of the preset size upon creation. By using all options at random and into the loop you will be able to fill the whole free space up with a useless encrypted data that will overwrite everything else that was deleted. Just do not forget to remove created files and run it many times to ensure that even forensic specialists will get harder to restore very sensitive data on it.

7zip

```
Password="$(dd if=/dev/urandom bs=128 count=1)"
DestinationFile="$((${RANDOM/0/1}$(date "+%s")/${RANDOM/0/1}))"
7z a -t7z -mhe=on -p"${Password}" -mx=0 -v1m ${DestinationFile} source
```

See also 7z(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/7z.1) for description of used options.

The `source` can be a predefined file with random data or a device, e.g. `/dev/urandom` or another block device or partition on it, e.g. `/dev/sd"XY"`, with data you are not afraid to be found then even deleted files on it will be compressed to the destination.

Note:

- It is not necessary to set level of compression, enough to store data for minimizing CPU usage and a faster fill free space up.
- If you are using a single file as a source then you can put it into the **RAM disk (http://www.vanemery.com/Linux/Ramdisk/ramdisk.html)** or the `/tmp` folder, because it uses **tmpfs** that allocates some amount of RAM because it will speed up reading.

Create multiple files with help of the *timeout* command

The `timeout` command with randomized waiting time used it in a **loop (http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-7.html)** will break the command that will leave a file with random size. This is a slow method but is one of the possible alternatives. You can also use an array with predefined file names before the random part of it.

```
AA=${RANDOM/0/1};
timeout $((AA/100)) cat /dev/urandom > filename${RANDOM}.tmp;
```

See also:

- **limits** for the file creation on the different file systems.
- **meta-data (http://www.r-tt.com/Articles/File_Recovery_Basics/)** in the filesystem may keep information about file after it was deleted.
- **forensic software (http://security.stackexchange.com/questions/58515/how-does-forensic-software-detect-deleted-files)** uses meta-data to recovery and what need to do for

wiping of the meta-data.

Retrieved from "https://wiki.archlinux.org/index.php?title=Securely_wipe_disk/Tips_and_tricks&oldid=492999"

- This page was last edited on 12 October 2017, at 15:04.
- Content is available under GNU Free Documentation License 1.3 or later unless otherwise noted.